Formalisme, outils et éléments méthodologiques pour la modélisation et la simulation multi-agents

Fabien Michel

Numéro d'identification :

Académie de Montpellier

Université Montpellier II

— **Sciences et Techniques du Languedoc** —

Thèse

présentée à l'Université des Sciences et Techniques du Languedoc
pour obtenir le diplôme de DOCTORAT

Spécialité	: Informatique
Formation Doctorale	: *Informatique*
Ecole Doctorale	: *Information, Structures, Systèmes*

Formalisme, outils et éléments méthodologiques pour la modélisation et la simulation multi-agents

par

Fabien Michel

Soutenue le 21 décembre 2004 devant le Jury composé de :

M. Alexis Drogoul, Professeur, Université Paris VI, ... Rapporteur
M. David R.C. Hill , Professeur, Université Blaise Pascal, Clermont-Ferrand, ... Rapporteur
M. Henry Van Dyke Parunak , Chief Scientist, Altarum Institute, Ann Arbor, MI, USA, ... Rapporteur
M. Yves Demazeau , Chargé de recherche HDR, Institut IMAG, Grenoble, ... Président
M. Jean-Pierre Müller , Directeur de recherche HDR, CIRAD, Montpellier, ... Examinateur
M. Jacques Ferber , Professeur, Université Montpellier II, ... Directeur de Thèse

A mon père.

Remerciements

Je remercie respectueusement mon directeur le thèse, le Professeur Jacques Ferber, pour ses conseils, ses encouragements, sa confiance, tout ce par quoi il a su orienter ma liberté de recherche.

Mes respects et ma gratitude vont également aux membres de mon jury qui m'ont fait l'honneur de juger ce travail en venant pour certains de fort loin et qui tous, par leur disponibilité, leurs observations et leurs rapports m'ont permis de l'enrichir.

∞

Mener à terme un doctorat comporte essentiellement deux aspects. Le premier concerne bien sûr la rédaction du document de thèse. Le deuxième a trait à l'organisation de l'événement si particulier que constitue le jour de sa soutenance. Faire l'une ou l'autre de ces deux choses sans l'aide et le soutien d'autres personnes est certainement impossible. Merci à Kadda, Greg et José pour avoir immensément contribué à chacun de ces deux aspects. Merci à Olivier Simonin pour toute l'aide qu'il m'a apportée de nombreuses fois durant toutes ces années. Merci à Danny Weyns de m'avoir fait participer à une activité de recherche si passionnante et inspirante (thank you Danny). Merci à Jérôme, Che, Didier, Christo, Marc, Mehdi, Simon et Céline pour leur disponibilité lors des pré soutenances. Merci à Lylia pour son immense contribution à l'organisation du jour J. A ce propos, je remercie Nicole Olivet, Nadine Tilloy et Florence Picone pour leur précieuse aide. Je tiens à remercier quelques-uns des membres du *SeT*, Abder Koukam, Vincent Hilaire, Sebastian Rodriguez et Nicolas Gaud pour m'avoir si amicalement accueilli et pour les précieux conseils qu'ils m'ont donnés pour la soutenance. Merci à Pierre Bommel pour ses nombreux coups de main (recherche et autres). Merci à Olivier Gutknecht pour m'avoir fait l'amitié d'être présent le jour de ma soutenance.

Mener à terme un doctorat suppose aussi bien sûr que les choses de la vie vous en aient donnés l'occasion. Je pense à tous mes professeurs sans lesquels ce travail n'existerait pas et plus spécialement à Mme Salima Hassas qui, à Lyon, m'a orienté vers les systèmes multi-agents. J'ai une pensée pour Jean-Marc Fouet qui m'a fait découvrir le domaine de l'Intelligence Artificielle. Merci à Elhadi Tchekiken pour m'avoir fait partager son expérience du doctorat alors que j'étais en maîtrise. Je remercie Pascale Tchekiken pour m'avoir permis de faire mes premières armes dans l'enseignement en tant que tuteur à l'Université Claude Bernard-Lyon1.

Mener à terme un doctorat est aussi une expérience humaine. Je remercie tous ceux qui m'ont aidé dans cette "aventure". Merci à Nacéra, Vincent, Julien et Eve, Denis et Karine, Rami, Toufik et Habiba. Merci à Noël pour avoir été à mes côtés le jour où il le fallait et avoir bien représenté la *lyonnaiserie* : Salem et Fatima, Alex, Hubert, Mylène, Béa et Zaïre, Ludo, Patrice, Mathieu, Christelle, Jean-Louis, Christophe, Catherine et Laurent.

Je remercie Danièle et Bernard, les parents d'Isabelle.

Je remercie ma mère et je remercie mon père.

Merci à Isabelle.

« *The most incomprehensible thing about the universe is that it's comprehensible.* »

Albert Einstein

Sommaire

Chapitre 1

Introduction

La modélisation et la simulation des systèmes complexes constituent aujourd'hui un enjeu majeur dans de nombreux domaines de la société humaine. L'actualité de la course à la puissance de calcul permet de s'en convaincre. En effet, depuis maintenant un peu plus de deux ans[1], l'ordinateur le plus puissant au monde n'est autre que *The Earth Simulator*[2]. Comme l'indique le directeur de ce projet, Tetsuya Sato, ce monstre de puissance est le fruit d'une ambition qui vise à exécuter des modèles dont la complexité est comparable à la réalité du système terrestre :

> "*With Earth Simulator, we are now able to search in territories where no intellectual creation of human kind was ever possible, being able to understand the Earth with all factors entangling together simultaneously, from micro process of how clouds or snow has been formed, to macro process of atmospheric circulation, as just the way Earth is. I call it the Holistic Simulation.*"

Si l'étude de l'écosystème terrestre constitue aujourd'hui une source intarissable de modèles complexes destinés à être simulés par ordinateur, c'est bien la communauté scientifique dans son ensemble qui utilise la simulation informatique à des fins multiples et variées. La simulation d'une explosion nucléaire, d'un robot explorant la planète Mars, de la mécanique d'un fluide, des échanges boursiers en sont quelques exemples. La simulation informatique est ainsi utilisée comme un outil scientifique à part entière. Elle permet de tester des hypothèses, de les transmettre, de les exposer et d'en formuler de nouvelles a posteriori. Cet outil constitue ainsi pour les scientifiques un moyen d'investigation unique, quel que soit le domaine considéré.

1.1 Contexte de la thèse

C'est dans ce contexte, celui de la modélisation et de la simulation par ordinateur des systèmes complexes, que cette thèse s'inscrit. Plus précisément, le travail ici rapporté se situe dans le domaine des systèmes multi-agents. Approche de recherche relativement récente, âgée d'environ une vingtaine d'années, la modélisation multi-agents repose sur l'idée qu'il est possible de représenter directement le comportement et les interactions d'un ensemble d'individus autonomes évoluant dans un environnement commun [Ferber, 1999]. Au contraire

[1] Selon le classement effectué par le projet *Top500* www.top500.org.

[2] www.es.jamstec.go.jp. Situé à Yokohama au Japon, cet ordinateur construit par NEC développe une puissance de calcul d'environ 40 Teraflops : 40 mille milliards d'opérations par seconde.

des approches de modélisation basées sur la définition d'équations mathématiques, où la dynamique d'un système est définie a priori par des relations fonctionnelles entre entités, la simulation multi-agents se propose de modéliser explicitement les comportements des entités et considère que la dynamique globale d'un système, au niveau macroscopique, résulte directement de l'interaction des comportements, au niveau microscopique [Parunak *et al.* , 1998]. Autrement dit, l'une des différences fondamentales qui existent entre ces deux approches est le niveau d'abstraction auquel le système est modélisé. Par exemple, dans le cadre d'une étude liée à l'évolution des populations dans un système proies/prédateurs, une approche classique consiste à modéliser cette évolution en établissant des relations fonctionnelles entre les différentes données observables du système, le nombre d'entités de chaque type par exemple. Les fonctions ainsi obtenues décrivent macroscopiquement l'évolution de ces différentes données observables au cours du temps. L'approche multi-agents est différente. Il s'agit dans ce cas de modéliser chaque individu, son comportement et les interactions qui découlent de la mise en commun de ces entités. La dynamique globale du système est alors issue de l'ensemble de ces interactions.

1.2 Problématiques abordées

De la même manière que pour une simulation informatique classique, la démarche scientifique sous-tendue par l'élaboration d'un modèle de simulation multi-agents repose sur une expérimentation qui comporte trois aspects fondamentaux :

- le *phénomène réel* (ou virtuel) que l'on souhaite étudier.
- le *modèle* de ce phénomène.
- la *simulation* de ce modèle par ordinateur.

Dans cette thèse, nous aborderons quelques-unes des problématiques liées à l'élaboration des modèles multi-agents et à l'implémentation des simulateurs permettant de les exécuter. Nous verrons notamment que la simulation multi-agents repose sur un processus expérimental dont les principes de conception restent aujourd'hui mal définis. Le premier objectif de cette thèse est de mettre en évidence l'une des conséquences les plus critiques de cet état de fait : la quasi impossibilité de reproduire fidèlement les expériences qui sont proposées dans la littérature. Les spécifications des modèles multi-agents sont en effet le plus souvent insuffisantes et ne permettent pas une implémentation non ambiguë du simulateur. Cela alors que, idéalement, l'implémentation du simulateur doit être neutre et un même modèle devrait toujours donner les mêmes résultats, quelle que soit la manière dont il est exécuté. Ce n'est pas le cas pour les modèles multi-agents dont de très nombreuses parties, mal spécifiées, peuvent être différemment interprétées. La neutralité du simulateur est alors perdue. De fait, l'implémentation d'un unique modèle peut donner des résultats très différents suivant l'interprétation qui en est faite. C'est ce que nous désignerons par le *phénomène de divergence implémentatoire*. Lié en grande partie à la complexité et à l'hétérogénéité des modèles considérés, nous verrons que de très nombreux facteurs interviennent dans la cause de ce problème et qu'il n'existe pas de solution évidente à ce dernier. Cette difficulté pose le problème de la vérification, de la validation et de l'exploitation des résultats des simulations qui sont réalisées. Difficulté qui est aussi à l'origine du manque de crédibilité dont les simulations multi-agents peuvent parfois souffrir.

La problématique précédente peut être résumée par le constat suivant : il existe un manque flagrant de correspondance entre les spécifications des modèles multi-agents et les structures informatiques qui permettent de les exécuter. L'objectif global de cette thèse est de **contri-**

buer à la mise en correspondance de ces deux parties. Pour cela, nous avons considéré qu'il était possible d'envisager le problème sous deux angles différents suivant que l'on se concentre sur l'une ou l'autre de ces deux parties.

Le premier est de considérer qu'un effort doit être fait pour que les structures informatiques que nous utilisons correspondent mieux aux modèles proposés. La première démarche que nous adopterons consistera ainsi à proposer des outils de conception de simulateur qui visent à rendre le fonctionnement d'un simulateur plus explicite et mieux adapté à l'implémentation d'un modèle multi-agents. Aujourd'hui, la plupart des simulateurs multi-agents disponibles reposent sur une encapsulation du moteur de simulation censée faciliter la conception. Cependant, il est alors difficile, voire impossible, de maîtriser l'ensemble des paramètres qui participent à l'implémentation de la dynamique du modèle dans son ensemble. Ce qui contribue largement au manque de correspondance entre le modèle et son implémentation. Il n'est pas rare qu'un modèle soit dénaturé par les contraintes imposées par les principes de fonctionnement d'un simulateur. De façon duale, l'encapsulation de l'implémentation engendre naturellement un manque de spécification du modèle en ce qui concerne les mécanismes qui sont dissimulés.

Le deuxième angle sous lequel le problème peut être abordé concerne la qualité des spécifications qui sont aujourd'hui utilisées pour modéliser un système multi-agents. Autrement dit, il s'agit ici de considérer que c'est sur les outils de spécification que l'effort doit être porté. A ce propos, nous proposerons une réflexion globale sur le paradigme agent et nous essaierons de déterminer les raisons pour lesquelles il est aujourd'hui si difficile de spécifier la dynamique d'un système multi-agents. Nous verrons notamment que cette difficulté est entretenue par le fait que le domaine lui-même ne propose pas de définitions formelles pour les différents concepts qu'il manipule, notamment ce qui concerne la propriété d'autonomie d'un agent. L'un des objectifs de ce manuscrit consistera ainsi à identifier les principes de modélisation qui doivent être associés au paradigme agent. Dans cette deuxième partie nous prendrons position sur des problématiques conceptuelles liées à la modélisation de systèmes multi-agents. Pour cela, nous proposerons une réflexion de fond sur la représentation des actions et des interactions entre agents dans les modèles multi-agents. Nous nous appuierons notamment sur une étude détaillée du modèle de l'action proposée par [Ferber & Müller, 1996] et nous en proposerons une adaptation pour la simulation multi-agents. Ensuite, nous verrons quelles sont les conséquences de son utilisation en ce qui concerne la modélisation de l'interaction. Nous proposerons alors un modèle formel appelé MIC* qui permet de prendre en compte l'ensemble des contraintes de modélisation que nous aurons identifiées tout au long de notre réflexion. Nous verrons en quoi celui-ci constitue un moyen adéquat de mettre en œuvre les différents points de notre analyse.

1.3 Plan de la thèse

Dans le chapitre 2, nous présenterons le contexte de la simulation informatique. Nous donnerons tout d'abord quelques définitions informelles qui permettent de se faire une idée globale du processus expérimental qui correspond à cette discipline. Ce qui nous amènera à présenter de façon non exhaustive les différentes techniques de modélisation et d'implémentation couramment utilisées. Après cela, nous détaillerons les concepts fondamentaux proposés par la *théorie de la modélisation et de la simulation* [Zeigler *et al.* , 2000], concepts qui nous permettront de présenter informellement la problématique de cette thèse et sur lesquels s'appuie une grande partie de notre raisonnement.

Le chapitre 3 consiste dans une introduction générale au domaine de la simulation multi-agents. Pour cela, nous verrons tout d'abord quels sont les principaux concepts associés au paradigme puis nous exposerons succinctement les motivations qui poussent la communauté scientifique à l'utiliser. Nous présenterons ensuite quelques domaines où cette démarche est appliquée, ce qui nous permettra de donner quelques exemples de plates-formes multi-agents. Après cela, nous nous approcherons davantage du cœur du sujet en abordant les problématiques directement liées à l'élaboration d'une simulation multi-agents. Notre but sera d'attirer l'attention du lecteur sur les difficultés associées à ces expériences.

Dans le chapitre 4, nous présenterons le problème de la *divergence implémentatoire*. Cette problématique, qui est en grande partie à l'origine de nos travaux de recherche, est liée à la difficulté de répliquer les expériences de simulation multi-agents proposées dans la littérature. Nous verrons notamment que ce problème pose la question de la vérification et de la validation de ces expérimentations et nous tenterons d'identifier les raisons qui en sont à l'origine. Nous donnerons pour finir les directions de recherche que nous avons prises dans l'optique de minimiser les phénomènes de divergence implémentatoire.

Dans le chapitre 5, nous présenterons les outils de conception de simulateur multi-agents de la plate-forme MADKIT. Nous verrons que ces outils reposent notamment sur les possibilités pratiques qu'ils offrent pour explorer un modèle et son implémentation. Nous donnerons à ce propos les raisons historiques et les motivations conceptuelles qui ont amené à leur réalisation. Pour cela nous présenterons tout d'abord le modèle AGR (Agent/Groupe/Rôle) et la plate-forme MADKIT avant d'expliciter le principe de développement associé à ces outils logiciels. Nous donnerons des exemples d'applications et nous proposerons un *pattern organisationnel* pour la simulation basé sur l'utilisation de ces outils.

Le chapitre 6 constitue le point central de notre réflexion. Nous y présenterons le modèle *Influence/Réaction* de [Ferber & Müller, 1996] qui est une proposition élégante pour résoudre le problème de la modélisation des actions simultanées. Nous montrerons pourquoi, bien plus qu'un modèle, il constitue en fait un principe de modélisation qui suppose une réflexion en profondeur sur la modélisation des systèmes multi-agents en général, notamment ce qui concerne la *distinction esprit/corps* pour la modélisation d'un agent. Nous en proposerons une adaptation pour la simulation et nous relaterons une application qui a utilisé ce modèle.

Dans le chapitre 7, nous proposerons une réflexion sur la modélisation de l'interaction. Point essentiel d'un modèle multi-agents, nous mettrons en évidence l'intérêt conceptuel et pratique de considérer l'interaction comme un point de modélisation clairement distingué. Ce qui nous permettra d'étudier la question de la validité de sa modélisation. Pour ce faire, nous reviendrons en détail sur la propriété d'*autonomie* d'un agent et nous introduirons la notion de *cohérence paradigmatique* d'un modèle multi-agents. Il s'agira pour nous d'identifier certain prérequis associés à l'utilisation du paradigme agent pour la modélisation de systèmes complexes. Nous discuterons alors de la possibilité de distinguer plusieurs formes d'interaction dans les systèmes multi-agents.

Le début du chapitre 8 proposera un récapitulatif des différentes contraintes conceptuelles et pratiques que nous aurons identifiées tout au long des précédents chapitres, ce qui nous amènera ensuite à récapituler les solutions qui leur sont associées. Après cela, nous présenterons en détail le modèle MIC* ({*Mouvement Interaction Calcul*}*) et nous montrerons en quoi il permet une modélisation des systèmes multi-agents qui soit en adéquation avec les différents principes que nous aurons évoqués.

Dans le chapitre 9 nous mettrons en application le modèle MIC* et nous exposerons dans le détail la modélisation et la simulation d'une société artificielle d'agents. Nous présenterons ensuite quelques résultats expérimentaux en proposant une rapide exploration du modèle obtenu. Cela nous permettra de conclure sur les différents aspects de cette expérience de simulation ainsi que sur ses limites.

Finalement, dans le chapitre 10 nous dresserons les conclusions issues de ce travail de recherche et nous essaierons de donner quelques-unes des perspectives qui lui sont associées à court et moyen terme.

Chapitre 2

Simulation informatique

AVANT de rentrer dans le vif du sujet, celui de la simulation multi-agents proprement dite, il convient tout d'abord de poser le cadre plus général de la simulation des systèmes dynamiques. En effet, les simulations de systèmes complexes qui utilisent le paradigme agent s'inscrivent naturellement dans ce contexte. De fait, les définitions, les modèles, les théories et les techniques d'implémentation qui se rattachent à cette discipline nous concernent directement. L'ensemble de ce chapitre est donc voué à la présentation de ce contexte.

2.1 Définitions générales

Donner une définition exhaustive, pointue et définitive de ce qu'est la simulation par ordinateur est sans doute une tâche vouée à l'échec étant donné le nombre de domaines concernés et l'hétérogénéité des applications qui en sont issues. C'est pourquoi, plutôt que d'en donner une seule, nous proposons ici au lecteur deux définitions complémentaires qui permettent de se faire une idée du contexte dans lequel cette thèse s'inscrit.

La première définition que nous donnons ici est empruntée à Shannon [Shannon, 1976]. Bien que cette définition soit presque trentenaire, elle est aujourd'hui encore utilisée, telle qu'elle a été énoncée, dans des articles qui se veulent être une introduction au monde de la simulation comme par exemple [Ingalls, 2001, Shannon, 1998]. Selon Shannon, la simulation peut être définie de la façon suivante :

> "*the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behaviour of the system or of evaluating various strategies (within the limits imposed by a criterion or a set of criteria) for the operation of the system.*"

La problématique sous-tendue par une simulation digitale est donc d'étudier un **système réel** de manière à comprendre son fonctionnement interne et/ou à en prévoir son évolution sous certaines conditions. De plus, pour atteindre ces objectifs, cette étude se fait nécessairement à travers un **modèle** du système réel qui est utilisé pour réaliser les expérimentations. Les termes **système réel** et **modèle** sont les deux mots-clés de cette définition. Il est important de comprendre que le système réel ne désigne pas forcément un phénomène qui existe dans la nature. Il peut aussi être une construction intellectuelle d'un phénomène virtuel. Cette dénomination est en fait utilisée pour clairement distinguer le phénomène à étudier de son modèle qui est lui aussi considéré comme un système. Shannon ajoute qu'un processus de

simulation est fondamentalement constitué, d'une part par la construction du modèle et, d'autre part, par l'utilisation analytique qui est faite de celui-ci pour étudier le système. La simulation informatique est en effet indissociable du processus expérimental qui se rattache à son objectif. Ce qui nous amène naturellement au deuxième énoncé que nous avons choisi qui s'attache à définir la nature de ce processus expérimental.

Selon Fishwick [Fishwick, 1997], la simulation informatique peut être comprise de la manière suivante :

> "*Computer simulation is the discipline of designing a model of an actual or theoretical physical system, executing the model on a digital computer, and analyzing the execution output.*"

Fishwick illustre par ailleurs ses propos à l'aide du schéma suivant (figure 2.1) :

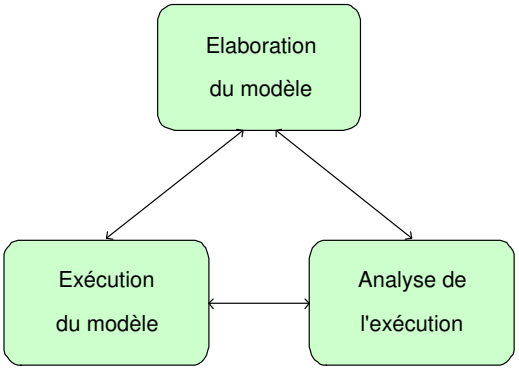

FIG. 2.1 – La simulation informatique selon [Fishwick, 1997].

Cette deuxième définition donne un aperçu global et intuitif de ce qu'est un processus de simulation par ordinateur. Fishwick définit ainsi cette discipline comme un processus itératif non linéaire composé de trois tâches fondamentales fortement interdépendantes :

1. l'élaboration du modèle.
2. l'exécution du modèle sur ordinateur.
3. l'analyse de l'exécution du modèle et des résultats obtenus.

Encore une fois, cette définition fait clairement apparaître l'importance du modèle, et donc de son élaboration, dans la conception d'une simulation. Par ailleurs, il est intéressant de remarquer que cet énoncé s'attache principalement à définir quelles sont les tâches fondamentales qui sont associées à cette démarche scientifique. En effet, au-delà des objectifs sous-tendus par ce type d'approche, il est dans un deuxième temps important de définir les différentes phases qui caractérisent sa mise en œuvre.

2.2 La simulation en tant que processus expérimental

La définition de Fishwick que nous avons donnée précédemment suffit la plupart du temps à illustrer clairement et simplement les différentes étapes qui constituent l'élaboration d'une simulation par ordinateur. Cependant, il peut être intéressant de rentrer dans le détail de ces étapes. Pour cela, on peut encore une fois se référer à Shannon [Shannon, 1976, Shannon, 1998] qui propose de distinguer les étapes de conception suivantes dans le processus de simulation :

1. Définition du problème : dans cette phase initiale, il s'agit de clairement définir les objectifs de l'étude. Quelles sont les questions auxquelles on souhaite apporter une réponse ?
2. Planification du projet : il s'agit ici d'être sûr que l'on disposera des ressources humaines et matérielles que nécessite l'étude entreprise.
3. Définition du système : dans cette phase, le but est de déterminer quels sont les aspects du système que l'on désire étudier de manière à pouvoir le définir de manière pertinente dans le cadre de l'expérience. Le modèle sera alors élaboré en fonction des objectifs fixés.
4. Formulation du modèle conceptuel : durant cette phase, un premier modèle est élaboré de manière graphique ou en pseudo code. Il s'agit de définir les différentes entités qui composent le système : composants, variables, interaction entre composants, etc.
5. Analyse préliminaire de l'expérimentation : il faut ici déterminer quels sont les critères qui permettront d'évaluer la qualité de l'expérimentation : quels sont les paramètres que l'on souhaite faire varier, avec quelle amplitude et sur combien d'exécutions. Combien d'expériences seront nécessaires à l'expérimentation dans son ensemble.
6. Constitution des paramètres initiaux : durant cette phase, il est question de déterminer et de collecter les données qui sont nécessaires à l'élaboration des valeurs initiales qui seront utilisées pour le paramétrage du modèle.
7. Transcription du modèle : cette étape consiste à convertir le modèle élaboré dans un langage de simulation de manière à permettre son implémentation sur machine.
8. Vérification et validation : il s'agit ici de vérifier dans un premier temps que le simulateur exécute correctement le modèle (debugging), pour dans un deuxième temps valider les résultats obtenus par celui-ci. Sont-ils acceptables et représentatifs du système que l'on souhaite étudier ?
9. Analyse finale de l'expérimentation : à ce stade de la conception, il convient de reconsidérer l'étape numéro cinq. En effet, il faut en mettre à jour ses conclusions étant donné que la connaissance du modèle s'est considérablement accrue.
10. Expérimentation : la simulation proprement dite est exécutée de manière à récupérer les résultats désirés et à effectuer une analyse de sensibilité du modèle aux paramètres initiaux.
11. Analyse et interprétation des résultats : une fois les simulations effectuées, il s'agit d'inférer des conclusions sur le modèle à partir des résultats obtenus.
12. Utilisation et documentation : outre les conclusions tirées de l'expérimentation, le modèle et son utilisation doivent être clairement documentés.

La figure 2.2, résume de manière schématique les principales étapes de conception liées à cette approche.

Encore une fois, il ne s'agit pas de considérer que cette décomposition constitue un dogme pour toute personne qui souhaite effectuer une expérience de simulation. Il existe d'ailleurs à ce sujet une critique très intéressante dans [Hymore, 1981] où il est considéré qu'une approche trop dirigée par les objectifs peut amener à dévoyer la définition du problème, et donc l'ensemble de l'expérimentation, en l'orientant vers une solution préconçue. Il nous a cependant semblé intéressant de la donner en exemple car elle constitue un bon aperçu de la manière dont ce genre d'expérimentation peut être conduit ; de la définition du problème jusqu'à la documentation des résultats et du modèle. Par ailleurs, elle introduit la question de la validation et de la vérification d'une simulation comme une étape fondamentale du processus expérimental. Question sur laquelle nous aurons l'occasion de revenir en détail tout au long de ce document.

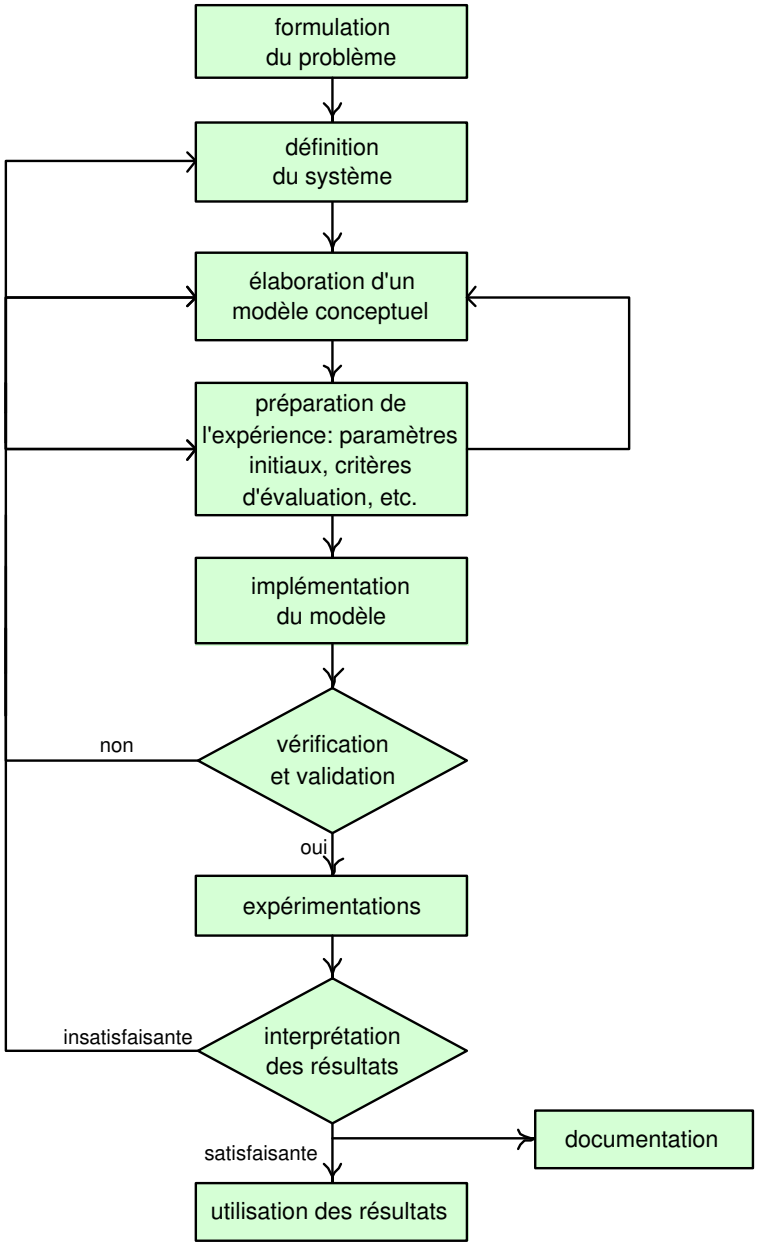

FIG. 2.2 – Etapes de conception d'une simulation informatique [Shannon, 1998].

2.3 Caractérisation des systèmes dynamiques

Maintenant que nous avons une idée générale de ce qu'est et en quoi consiste une simulation par ordinateur, il convient de de présenter les différents concepts qui sont manipulés pour modéliser des systèmes réels. Dans ce cadre, la notion de *système dynamique* joue un rôle central. Par système dynamique on entend [Rozenblit & Zeigler, 1993] :

> "*Any formal construct which provides general modeling concepts for various kind of disciplines.*"

Pour comprendre les différents concepts liés à la notion de système dynamique, il nous semble intéressant d'introduire, même succinctement, les bases de la *théorie des systèmes*. Elaborée dans les années 60, cette théorie fournie un cadre formel pour la représentation de systèmes dynamiques dans le but de proposer des méthodes et des outils pour la résolution de problèmes liés à l'étude de ces systèmes. La simulation informatique s'inscrivant naturellement dans ce cadre, la plupart des modèles utilisés pour représenter un système dynamique se fondent sur les abstractions proposées par cette théorie. Le principe de base de cette théorie est de considérer qu'un système peut être spécifié selon deux aspects fondamentaux :

- le *comportement* du système à ses bornes (comportement externe), c'est-à-dire les réactions observables du système depuis l'extérieur de celui-ci.
- la *structure interne* du système, c'est-à-dire son état interne et son fonctionnement intrinsèque (sa dynamique).

Ainsi, au plus haut niveau d'abstraction, un système dynamique est vu comme une boîte noire (black box) qui possède une entrée, une sortie et une structure interne comme le montre la figure 2.3.

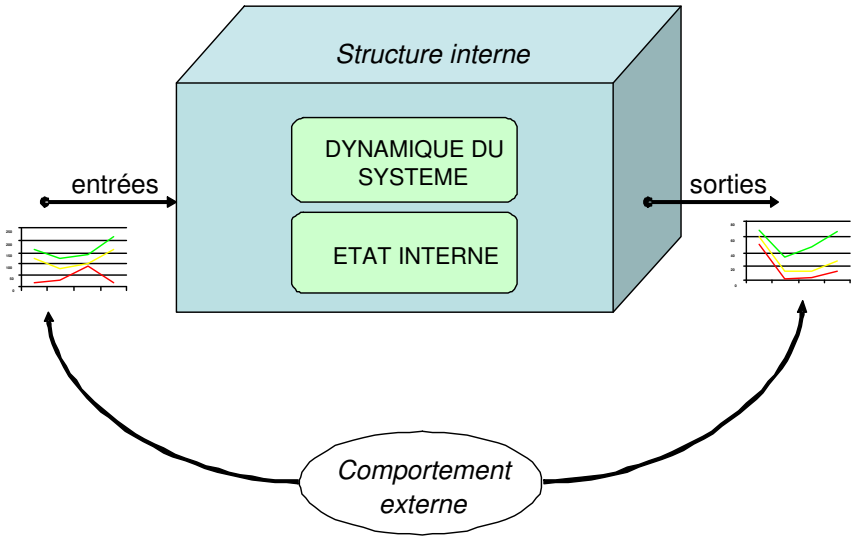

FIG. 2.3 – Représentation classique d'un système dynamique : la boîte noire.

Le comportement externe du système est alors défini par la relation qui existe entre l'historique des entrées et l'historique des résultats observés en sortie. Autrement dit, il caractérise la manière dont le système réagi du point de vue de l'observateur au niveau des entrées/sorties (E/S).

La structure interne du système est quant à elle définie selon trois paramètres :

- l'*état interne* du système (*system state*) qui est généralement représenté par une ou plusieurs variables appelées *variables d'état*.
- le *mécanisme de changement d'état* du système qui désigne la façon dont les variables d'état évoluent en fonction des entrées ou de manière autonome. Cet aspect de la dynamique du système est modélisé par ce qu'on appelle généralement la *fonction de transition d'état*.
- le *mécanisme de production* du système qui fait référence à la manière dont celui-ci produit un résultat en sortie en fonction de son état interne. On parle ici de la *fonction de sortie* du système.

Généralement, les deux derniers points sont implicitement regroupés lorsqu'on parle de la *dynamique du système* qui concerne les mécanismes d'évolution du système au cours du temps, par opposition à l'*état du système* qui fait référence à la situation dans laquelle le système se trouve à un instant précis dans le temps. Ainsi, le premier objet des formalismes qui sont issus de cette représentation est de spécifier les fonctions qui implémentent la dynamique du système : la fonction de transition d'état et la fonction de sortie. Par ailleurs, on désigne généralement par X le domaine des valeurs possibles pour les entrées, par Y le domaine des valeurs de sorties et par S le domaine des valeurs de l'état interne du système.

Lorsque le système peut être décomposé en plusieurs sous-systèmes couplés entre eux (en reliant les différents ports E/S), on parle de la *décomposition* du système en *composants système*. De la même manière, on parle de *composition* pour décrire la manière dont plusieurs systèmes existants peuvent être mis en relation pour former un plus grand système. Dans ces cas-là, on parle de *construction hiérarchique* du système. De plus, la théorie des systèmes est dite *fermée par composition* (*closed under composition*) ; la structure et le comportement d'une composition de systèmes peuvent encore être décrits selon les termes utilisés pour un système (état interne, fonction de transition d'état, fonction de sortie, etc.). Ainsi, le deuxième aspect des formalismes que nous avons évoqués consiste dans la spécification des systèmes composés.

Comme nous le verrons, le concept d'agent est lui aussi généralement représenté de manière abstraite par le schéma de la boîte noire. Seuls les termes changent (cf. section 3.1.1). Par conséquent, les systèmes multi-agents sont eux aussi souvent schématisés par une construction hiérarchique qui définit les accointances qui existent entre les différents agents.

Le point important qu'il faut ici retenir est la distinction que fait la théorie des systèmes entre les différents aspects d'un système dynamique : comportement et structure interne. Nous allons maintenant voir en quoi cette distinction permet d'identifier les principales problématiques auxquelles on peut être confronté suivant l'aspect sous lequel on étudie un système.

2.4 Principales problématiques

Suivant les connaissances que l'on a d'un système, sur sa structure interne et/ou sur son comportement externe, les objectifs liés à son analyse peuvent être très différents. Connaître la structure interne d'un système permet d'en déduire (analyser, générer, simuler) son comportement. En revanche, induire la structure interne d'un système de l'observation de son comportement n'a généralement pas une seule solution car, dans ce cas de figure, il peut exister plusieurs modèles valides, c'est-à-dire des structures capables de produire le comportement observé. Cette constatation nous amène à introduire la très intéressante notion de *niveaux de connaissance* d'un système telle qu'elle a été proposée par Klir [Klir, 1985]. Bien que celle-ci ne

se restreigne pas uniquement à l'étude des systèmes dynamiques, elle permet en effet de bien comprendre les principales problématiques qui peuvent être abordées lorsque l'on étudie un système suivant le degré d'information que l'on possède sur lui. Le tableau 2.1 décrit les quatre niveaux de connaissance de base identifiés par Klir. Chaque niveau identifie des connaissances supplémentaires sur le système qui ne se trouvent pas dans le niveau inférieur.

TAB. 2.1 – Niveaux de connaissance d'un système

Niveau	**Nom**	**connaissances que l'on a sur le système**
0	Source	l'ensemble des variables d'intérêt et le moyen de les observer
1	Donnée	un historique du comportement externe du système
2	Génération	un moyen de générer le comportement observé (par exemple une formule mathématique qui met en relation les E/S)
3	Structure	un modèle générateur décrit comme un ensemble de composants couplés entre eux (une construction hiérarchique)

L'idée fondamentale sous-tendue par cette hiérarchisation est que lorsque la structure est spécifiée, au niveau 3, la connaissance que l'on a sur le système est maximale dans le sens où l'on dispose de toutes les informations nécessaires à la déduction des informations de niveau inférieur. Autrement dit, on a la connaissance qui permet de générer les données observées au niveau 1. Pour Klir, descendre la hiérarchie ne revient qu'à rendre explicite ce qui est défini de façon implicite dans le niveau supérieur et aucune nouvelle connaissance n'est véritablement produite.

Ainsi, suivant le niveau de connaissance que l'on a sur un système, on peut définir les trois principales problématiques liées à l'étude d'un système suivant que l'on monte (dans l'ordre numérique) ou que l'on descende dans cette hiérarchie :

- *analyse du système* : dans ce cas, la structure du système réel est connue et l'objectif consiste à analyser (comprendre, prévoir) son comportement. Cette démarche correspond à une descente dans la hiérarchie des niveaux de connaissance du système. Par exemple, on peut posséder le modèle détaillé d'un véhicule et vouloir simuler son comportement en situation.
- *inférence sur le système* : dans ce cas, la structure du système réel est inconnue et il s'agit de découvrir cette structure à partir des observations que l'on peut faire sur lui (base de données comportementales). Il faut donc ici monter dans les niveaux pour induire des structures valides. On est généralement dans ce cas de figure lorsqu'on étudie des phénomènes naturels complexes comme le climat par exemple.
- *création du système* : dans ce cas, on désire créer un nouveau système possédant un comportement particulier. L'objectif consiste ici à déterminer la structure qui sera la mieux adaptée pour générer ce comportement. Encore une fois, il s'agit ici d'une montée dans les niveaux. Il peut par exemple être intéressant de faire la simulation d'un serveur informatique avant sa mise en place de manière à connaître le nombre optimal de machines suivant la qualité de service que l'on désire.

Il est parfois difficile de classifier d'une manière aussi simple les études qui sont réalisées. Certaines expériences sont à la frontière entre plusieurs de ces problématiques. Dans le cadre de la vie artificielle par exemple, les systèmes étudiés sont parfois créés de toutes pièces sans avoir forcément en tête un comportement particulier à atteindre. Une fois le système créé, son comportement est alors analysé pour en extraire des propriétés intéressantes. Ces propriétés peuvent à leur tour être utilisées pour essayer d'inférer de nouvelles structures qui permettent de les générer.

Il est intéressant de noter que la vision de Klir n'est pas universellement partagée par la communauté. En effet, il est tout à fait raisonnable et pertinent de penser que lorsqu'on rend explicite (au niveau inférieur), ce qui est considéré comme implicite par Klir (au niveau supérieur), on augmente ses connaissances sur le système dans le sens où certaines propriétés importantes peuvent apparaître alors qu'elles n'étaient pas apparentes, voire non déductibles, au niveau supérieur. Nous pensons ici par exemple à la propriété d'émergence[1] d'un système : même si l'on connaît la structure interne, par définition cette propriété ne peut être attribuée au système qu'une fois qu'elle a été observée. Cependant, nous pensons qu'il n'y a là pas de contradiction avec la vision de Klir. En fait, la *connaissance du système*, au sens de Klir, ne doit pas être comprise comme la *compréhension* du système mais plutôt comme la capacité à pouvoir générer les informations de niveau inférieur. Autrement dit, descendre dans les niveaux suggère un processus de déduction alors que monter dans les niveaux est un processus d'induction. Par exemple, même s'il faut attendre l'observation du comportement d'un système pour effectivement constater un phénomène émergent, lorsque sa structure est définie, on dispose alors de toutes les informations qui permettront d'observer ce phénomène : on pourra générer ce comportement. Au contraire, il peut arriver qu'on ne trouve jamais une structure qui permette de générer le comportement d'un système que l'on a observé (même si ce comportement est une simplification de la réalité). C'est pourquoi on a moins de connaissance sur lui, toujours au sens de Klir.

2.5 Les différents modèles temporels utilisés pour la représentation des systèmes dynamiques

Un système dynamique est avant tout défini par la façon dont il évolue au cours du temps. De fait, une des caractéristiques les plus importantes d'un modèle concerne la manière dont l'écoulement du temps est représenté. Ainsi, bien que les modèles puissent être fortement hétérogènes en regard des systèmes qu'ils représentent, ils peuvent être distingués en trois grandes catégories suivant la modélisation du temps qu'ils utilisent. Nous présentons dans cette section ces trois grandes familles de modèle : les modèles continus, les modèles discrets et les modèles événementiels.

2.5.1 Modèles continus

Les *modèles à temps continu* sont caractérisés par le fait que, dans un intervalle de temps fini, les variables d'état du système changent de valeur infiniment souvent, c'est-à-dire de

[1] Sans entrer dans le détail de cette notion complexe, l'émergence peut être comprise comme l'apparition inattendue, à partir d'un système complexe, d'un phénomène qui n'avait pas semblé inhérent aux différentes parties de ce système. Ces phénomènes émergeant ou collectifs montrent qu'un tout peut être supérieur à la somme de ses parties [Minsky, 1986].

manière continue. La figure 2.4 montre un exemple de l'évolution de la valeur d'une variable d'état x en fonction du temps dans un modèle continu.

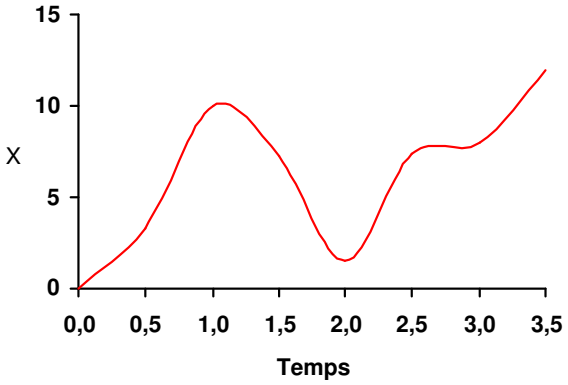

FIG. 2.4 – Evolution d'une variable dans un modèle continu.

Par exemple, la quantité d'eau x présente dans un récipient qu'on est en train de remplir peut être représentée par une variable d'état qui varie continuellement. Pour calculer l'évolution d'une telle variable, il est donc nécessaire de connaître son taux de variation au cours du temps, c'est-à-dire $dx(t)/dt$. Ce taux de variation est lui-même une fonction du temps (le débit $Deb(t)$ par exemple). Ce modèle très simple est donc représenté par l'équation différentielle suivante : $dx(t)/dt = Deb(t)$. Ainsi, tous les modèles continus sont représentés par un ensemble d'équations différentielles plus ou moins complexes selon les cas. Le formalisme de base associé à ce type de modèles est appelé *Differential Equation System Specification* DESS. Pour plus de détails sur ce type de modèles on peut se référer au livre de Cellier [Cellier, 1991a]. Bien que ce type de modèles soit naturellement adapté à la modélisation de phénomènes physiques réels (en utilisant directement les équations de la physique), ils sont aussi utilisés pour décrire toutes sortes de dynamiques. Un des modèles continus les plus connus est d'ailleurs celui qui a été proposé en 1926 par Volterra pour décrire la dynamique d'un écosystème de type proies/prédateurs à l'aide de deux équations différentielles.

2.5.2 Modèles discrets

Dans ce type de modèles, l'axe du temps est discrétisé suivant une période constante Δt appelée le *pas de temps*[2]. L'évolution des variables d'état du système se fait alors de manière discrète, c'est-à-dire instantanée, d'un instant t au suivant $t + \Delta t$. La figure 2.5 montre un exemple de l'évolution d'une variable d'état x en fonction du temps dans un modèle discret.

La construction de ce type de modèles implique de décrire les fonctions qui permettent de calculer l'état du système, pour un nouvel instant $t + \Delta t$ à partir de son état à l'instant t. De manière simplifiée, soit $\sigma(t)$ l'état global du système (état interne, entrée et sortie) à l'instant t, il existe une fonction ϕ de la forme suivante : $\sigma(t + \Delta t) = \phi(\sigma(t))$. On peut ainsi connaître l'état du système pour tout instant t en calculant successivement tous les états intermédiaires à partir d'un état initial.

[2]Le *pas de temps* Δt est généralement une valeur entière mais cela n'est pas une obligation. Le fait est que l'on peut toujours se ramener à une valeur entière étant donné que Δt est une constante.

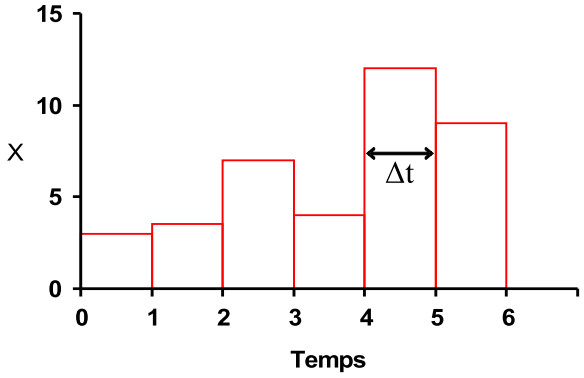

FIG. 2.5 – Evolution d'une variable dans un modèle discret.

Dans le détail, on distingue plusieurs types de modèles discrets suivant la nature des fonctions qui implémentent la dynamique du système. Par exemple, dans le formalisme de base associé à cette approche, qui est appelé *Discrete Time System Specification* DTSS, la dynamique d'un système est modélisée par deux fonctions. La première est la *fonction de transition d'état* $\delta : Q \times X \mapsto Y$. Elle définit comment est calculé le nouvel état interne du système en fonction d'une entrée. La deuxième, la *fonction de sortie* λ, peut être définie de plusieurs manières. Lorsqu'elle ne prend en compte que l'état interne pour calculer un résultat en sortie, on parle d'un système de *type Moore* : $\lambda : Q \mapsto Y$. Lorsqu'elle prend aussi en compte l'entrée courante, le système est de *type Mealy* : $\lambda : Q \times X \mapsto Y$. Il existe aussi des systèmes dits *sans mémoire* pour lesquels λ ne prend en compte que l'entrée courante : $\delta : X \mapsto Y$. Cette distinction est tout à fait cruciale dans le cas des systèmes composés où la sortie d'un composant peut constituer l'entrée d'un autre. Dans le cas d'un composant de type Mealy, le résultat observé en sortie est directement fonction de la valeur en entrée qui va ainsi influencer instantanément le composant suivant. Au contraire, dans un composant de type Moore, la sortie ne dépend que de l'état interne et l'influence de la valeur d'entrée est retardée d'un pas de temps. C'est par exemple le cas dans les automates cellulaires où l'ensemble des cellules est de type Moore. En effet, le nouvel état d'une cellule est déterminé en fonction de l'état courant de ses voisines. Ainsi, le changement d'état d'une cellule est tout d'abord stocké dans une variable temporaire qui sera validée uniquement lorsque le calcul de l'ensemble des nouveaux états des cellules est terminé. Le célèbre exemple du *Jeu de la vie* de Conway suit ce principe.

Cette représentation du temps a un franc succès dans le domaine de la simulation multi-agents et nous aurons l'occasion de revenir en détail sur celle-ci. Nous verrons que bien que les simulations multi-agents qui utilisent un modèle discret soit rarement formalisées, les techniques de simulations qu'elles emploient peuvent être rapprochées de ces deux principaux types de fonctionnements : les simulations de type Moore ou des variables tampons sont utilisées et les simulations de type Mealy où les actions sont directement validées.

2.5.3 Modèles à événements discrets

La troisième grande catégorie de modèles est celle des *modèles à événements discrets*. Paradoxalement, dans ce type de modèle, l'axe temporel est généralement continu, c'est-à-dire représenté par un nombre réel. Cependant, au contraire des modèles continus, les variables d'état du système changent de manière discrète (instantanée) à des instants précis qui sont

appelés *événements*. La figure 2.6 montre un exemple de l'évolution d'une variable d'état x en fonction du temps dans un modèle événementiel.

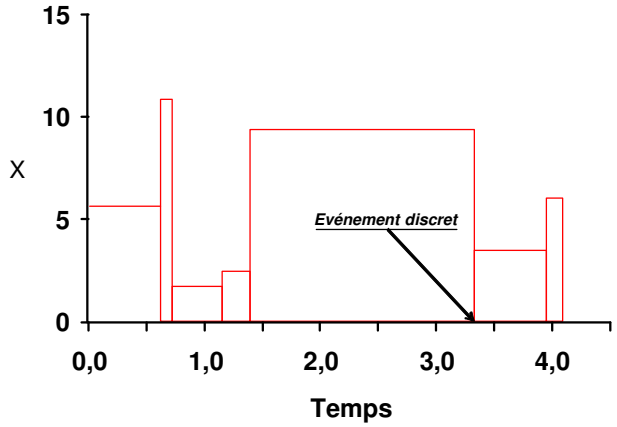

FIG. 2.6 – Evolution d'une variable dans un modèle événementiel.

Dans ce type de modèle, la dynamique du système est donc définie par la nature et la date des événements qui interviennent. On distingue deux types d'événements : les *événements externes* qui sont produits par l'environnement extérieur et pris en compte au niveau de l'entrée du composant système et les *événements internes* qui sont produits par le système lui-même et dont le résultat est observé en sortie du composant. La simulation de files d'attentes est un exemple classique de l'utilisation de modèles événementiels. Dans ce type de simulation, on distingue par exemple des événements tels que l'arrivée d'une nouvelle entité dans une file d'attente, le début d'un traitement et la fin d'un traitement.

Le formalisme de base utilisé pour la spécification des modèles événementiels est le *Discrete Event System Specification* qui est noté DEVS [Zeigler *et al.* , 2000]. Même rapidement, il nous faut ici en donner une courte description. D'une part parce que ce formalisme concrétise les concepts de la théorie des systèmes d'une manière simple et explicite. D'autre part parce qu'il constitue aussi une base qui peut être utilisée pour définir de nombreux autres formalismes, notamment *Parralel DEVS* dont nous parlerons dans le prochain chapitre. Dans sa forme la plus simple et la plus classique, une spécification DEVS est une structure $M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$ qui modélise un système dynamique telle que :

- X est le domaine des valeurs possibles en *entrée*.
- S est le domaine des valeurs possibles de l'*état interne*.
- Y est le domaine des valeurs possibles en *sortie*.
- $\delta_{int} : S \mapsto S$ est la fonction de *transition interne*
- $\delta_{ext} : Q \times X \mapsto S$ est la fonction de *transition externe* où $Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$ est le domaine de *l'état total* avec e le *temps écoulé* depuis la dernière transition d'état.
- $\lambda : S \mapsto Y$ est la *fonction de sortie*
- $ta : S \mapsto \mathbb{R}^+$ est la *fonction d'avancement du temps*

Très rapidement, cette structure définit un mécanisme de transition d'état où il existe deux cas de figure suivant qu'un événement externe survient ou non. Lorsqu'aucun événement externe ne survient, le système reste dans l'état s pendant un temps $ta(s)$ (temps de repos). Une fois ce temps écoulé, c'est-à-dire lorsque $e = ta(s)$, le système produit une valeur $\lambda(s)$ en sortie et passe dans un état s', calculé par la fonction de *transition interne*, tel que $s' = \delta_{int}(s)$. Lorsqu'un événement externe $x \in X$ intervient avant que $e = ta(s)$, le système passe dans un

état s' calculé par la fonction de *transition externe* tel que $s' = \delta_{ext}(s, e, x)$. Ce qui définit de fait un nouveau temps de repos $ta(s')$[3].

On trouve une description très complète de ce formalisme dans les récents travaux de Duboz [Duboz, 2004]. Nous donnons cette référence car elle propose par ailleurs une utilisation poussée de ce formalisme dans le contexte de la modélisation de systèmes multi-agents.

Par ailleurs, nous tenons à mentionner l'existence de formalismes comme *Discrete Event and Differential Equation System* DEV&DESS permettant d'élaborer des modèles hybrides [Praehofer *et al.* , 1993]. En effet, certains systèmes ne peuvent être modélisés de manière adéquate en utilisant une approche purement continue, ou à l'inverse purement discrète [Cellier, 1986].

2.6 Principales techniques d'implémentation

La structure algorithmique des simulateurs qui sont utilisés pour exécuter un modèle est en grande partie liée à la gestion du temps considérée. Ainsi, à chaque type de modèle temporel correspondent une ou plusieurs techniques génériques de simulation. Dans cette section nous allons grossièrement présenter les principes d'implémentation généraux qui sont utilisés pour simuler les différents modèles que nous avons présentés. La technique de simulation par intégration pour les modèles continus, la simulation à pas de temps constant dans le cas des modèles discrets et le principe d'échéancier pour les modèles événementiels.

2.6.1 Simulation des modèles continus

Il serait long et fastidieux de présenter ici dans le détail les techniques de simulation qui se rapportent à l'implémentation de modèles continus. D'autre part, nous verrons que l'approche multi-agents est par nature éloignée des modèles continus. En effet, le changement de comportement d'un agent est toujours un événement ponctuel, et donc discret. Ce qui ne veut pas dire pour autant que les agents ne puissent être plongés dans un environnement décrit par un modèle continu. Cela étant dit, il faut savoir que la simulation de modèles continus pose de nombreux problèmes dus à la nature digitale de l'ordinateur : il est tout simplement impossible de reproduire la continuité de la dynamique d'un système car celui-ci évolue infiniment souvent alors que la simulation digitale impose des calculs ponctuels. Les méthodes qui permettent de résoudre ce problème sont connues sous le nom de *méthodes d'intégration numérique*. Comme leur nom l'indique, il s'agit de méthodes mathématiques qui prennent en compte plusieurs paramètres (valeurs passées et/ou futures estimées, taux de changement, etc.) pour estimer la véritable valeur de l'état du système au plus près.

2.6.2 Simulation des modèles discrets

Etant donnée la représentation du temps utilisé, les techniques d'implémentation employées pour la simulation des modèles discrets sont sans aucun doute les plus simples à mettre en œuvre. En effet, lorsque les fonctions qui implémentent la dynamique du système sont

[3] On définit parfois des états tels que $ta(s) = 0$. Cela de manière à ce que le système produise immédiatement une valeur en sortie à la suite d'une transition externe par exemple. On dit alors que s constitue un *état transitoire*. De la même manière, certains états sont tels que $ta(s) = \infty$, on parle alors d'*état passif*.

clairement définies, il suffit de mettre en place un algorithme qui applique ces fonctions puis incrémente le temps d'une unité. La figure 2.7 illustre ce principe de simulation.

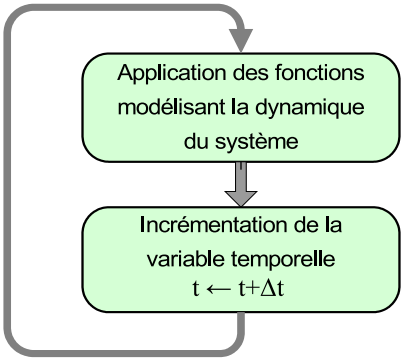

FIG. 2.7 – Principe de simulation d'un modèle discret.

Bien sûr, tout n'est pas aussi simple dans le cas d'un système composé. Le simulateur doit dans ce cas implémenter les différentes caractéristiques liées au type du modèle considéré (Moore, Mealy, etc.) et les algorithmes correspondants peuvent être très différents. Nous en verrons plusieurs exemples dans le contexte de la simulation multi-agents. C'est pourquoi nous n'en dirons ici pas plus.

2.6.3 Simulation par événements

Il existe trois politiques d'implémentation couramment utilisées pour la simulation de modèles à événement discret : par *ordonnancement d'événements* (*event sceduling*), par *activités* (*activity scanning*) et par *interaction de processus* (*interaction process*) qui est une combinaison des deux premières.

Ordonnancement d'événements

La simulation par ordonnancement d'événements est la technique la plus fréquemment utilisée. Elle consiste à déterminer au fur et à mesure de la simulation les futurs événements qui doivent intervenir sur le système. Ainsi, dans ce type de simulation, tous les événements sont déterminés au préalable. Les événements externes sont générés par ce qui peut être appelé un *générateur d'événements* et les événements internes sont produits par les composants système eux-mêmes. Chaque événement ainsi créé est placé dans une *liste d'événements* qui est ordonnée de manière chronologique. Au lancement de la simulation, l'horloge logique de la simulation est avancée jusqu'à la date de l'événement chronologiquement le plus proche qui est alors retiré de la liste et exécuté. Son exécution peut engendrer la planification d'un nouvel événement qu'il faut bien sûr intégrer dans la liste d'événements. La figure 2.8 résume l'algorithme qui correspond à ce principe de simulation.

La simulation de file d'attente est un exemple simple de ce type de programmation. Par exemple, pour simuler un système "station-service" on peut définir les événements E= {"arrivée d'une voiture dans la queue", "début de service", "fin de service"}. L'événement "arrivée d'une voiture" est un événement externe tandis que les deux autres sont des événements internes. Lorsqu'une voiture débute son service, la date de l'événement "fin de service" est immédiate-

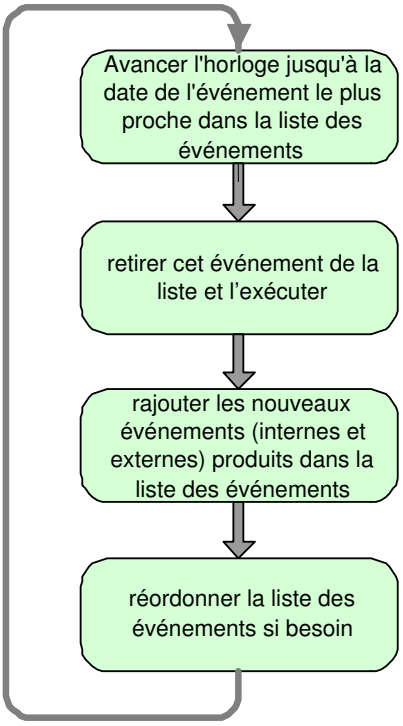

FIG. 2.8 – Principe de simulation par ordonnancement d'événements.

ment calculée et l'événement est alors planifié et sera exécuté à la date prévue. L'événement "fin de service" est donc une conséquence de l'événement "début de service".

Analyse d'activités

Dans le cas d'une simulation par activités, l'occurrence de certains événements est assujettie à une règle d'activation qui est conditionnée par l'état du système. Autrement dit, ces événements ne sont pas planifiés mais déclenchés, au fur et à mesure de la simulation, lorsque certaines conditions sont remplies. L'idée qui se cache derrière cette approche est de pouvoir définir des événements dont on ne peut pas connaître à l'avance la date de déclenchement. Par exemple, l'événement collision interviendra si deux véhicules rentrent en contact. Pour mettre en place cette technique, il suffit, pour chaque exécution d'un nouvel événement, de vérifier si celui-ci engendre un état du système qui vérifie les conditions de déclenchement de ce type d'événements.

Interaction de processus

Cette approche est une combinaison des deux premières dans le sens où les deux méthodes précédentes sont utilisées pour définir le comportement d'un composant. Celui-ci est alors spécifié par un ensemble de routines qui gèrent l'état du composant à la manière d'un programme séquentiel, d'où le terme processus : gestion des entrées, procédures conditionnelles, génération des sorties, etc. Dans le cas d'un système composé on a alors un ensemble de processus en interaction. Il est cependant important de ne pas confondre ce type de système avec un ensemble de programmes en interaction où le temps n'est pas modélisé. Dans le cadre d'une

simulation, ces processus sont soumis à une horloge globale commune et le rythme de leur évolution est complètement contrôlé.

Dans [Balci, 1988], on trouve une synthèse plus détaillée des différentes implémentations liées à l'utilisation d'un principe événementiel. Basée sur l'analyse des différentes méthodes qui peuvent être utilisées pour définir l'évolution temporelle du système (*Time Flow Mechanism* TFM), cette synthèse intègre par ailleurs une description du principe de simulation à pas de temps constant qui est bien sûr un cas particulier du principe de simulation par événements.

2.7 Nature des variables d'un modèle

La modélisation du temps utilisée n'est pas la seule caractéristique générique que l'on peut extraire d'un modèle. La deuxième aspect fondamental d'un modèle tient dans la nature des variables qui sont utilisées pour représenter l'état du système. On peut ainsi distinguer deux grandes catégories de variables : les *variables quantitatives* et les *variables qualitatives* [Cellier, 1991b].

2.7.1 Variables quantitatives

Une variable est dite quantitative lorsque sa valeur est exprimée par un réel. Généralement, les variables quantitatives sont utilisées pour modéliser des grandeurs physiques comme la distance entre deux entités par exemple.

2.7.2 Variables qualitatives

Une variable qualitative est une variable dont le domaine de valeurs est défini par des valeurs qualitatives comme D={"très froid"," froid"," tiède"," chaud"," très chaud"}. Plus précisément, on distingue aussi plusieurs catégories de variables qualitatives suivant le type des mesures auxquelles elles correspondent :

- une *mesure nominale* correspond à un domaine de valeurs non ordonnées qui s'excluent mutuellement. La couleur des cheveux en est un exemple.
- une *mesure ordinale* est utilisée pour des variables nominales dont les valeurs du domaine de définition peuvent être ordonnées comme c'est le cas pour le domaine D que nous avons vu précédemment. Par ailleurs, la condition d'exclusion mutuelle n'est pas toujours vérifiée, notamment lorsqu'on travaille avec des ensembles flous où certaines valeurs peuvent appartenir à deux ensembles en même temps.
- une *mesure par intervalles* correspond à une mesure ordinale où la distance entre deux valeurs peut toujours être calculée. Le quotient intellectuel d'une personne en est un exemple.
- une *mesure par ratio* est une mesure par intervalles où le domaine de définition contient un véritable zéro (avec des valeurs symétriques par rapport à ce zéro).

Ainsi, on distingue les modèles quantitatifs, qui opèrent sur des variables quantitatives, des modèles qualitatifs qui utilisent des variables qualitatives et définissent ainsi des *comportements qualitatifs*, c'est-à-dire des comportements définis par une suite ordonnée dans le temps des valeurs prises par une variable qualitative (scénario). On parle ainsi aussi de *simulation qualitative* par opposition aux *simulations quantitatives*. Par ailleurs, il est important

de mentionner qu'il est bien sûr possible de réaliser des modèles où les deux types de variables coexistent [Fishwick & Zeigler, 1992].

2.8 La théorie de la M&S (modélisation et simulation)

Bien que la nature d'une expérience de simulation soit indissociable du domaine de recherche pour lequel elle a été élaborée, il est cependant nécessaire d'établir une base de réflexion qui puisse être partagée par l'ensemble des acteurs qui participent à de telles expériences. Il est en effet fondamental de considérer la simulation informatique comme une discipline à part entière et, de fait, comme un véritable objet d'étude en soi qui doit être théorisé. Dans cette optique, il existe une contribution d'une importance non négligeable ; la *théorie de la modélisation et de la simulation* de Zeigler. Proposée au début des années 70 [Zeigler, 1972] et récemment réactualisée dans [Zeigler *et al.* , 2000], une partie de cette théorie a pour objectif de fournir une base méthodologique générale pour la conception de simulation, et en cela elle nous sera d'un grand intérêt. Plus important encore, une fois que nous aurons présenté ce cadre théorique, nous disposerons alors d'un vocabulaire précis qui nous permettra de situer, dans le contexte de cette théorie, la problématique que nous souhaitons aborder dans cette thèse. Il est d'ailleurs très étonnant que très peu de travaux portant sur les simulations multi-agents se positionnent par rapport à cette théorie[4]. Ainsi, en présentant quelques-unes des notions de cette théorie, nous espérons aussi contribuer un peu à sa promotion au sein de notre communauté.

2.8.1 Hiérarchie des spécifications d'un système

Dans le cadre de la M&S, toute la problématique de la modélisation d'un système dynamique repose sur l'élaboration d'une spécification acceptable de celui-ci. La nature de ces spécifications n'est pas la même suivant les connaissances que l'on a sur le système. De plus, un système peut être spécifié de nombreuses manières et avec des formalismes différents. C'est pourquoi il est important d'organiser ces spécifications, indépendamment du formalisme utilisé, suivant la qualité des informations qu'elles fournissent sur le système : du concept intuitif de boîte noire à la description fine de la structure interne, des composants système et de leur agencement. Zeigler propose de hiérarchiser les spécifications d'un système de la manière suivante (table 2.2) :

Bien que cette hiérarchie ne soit pas sans rappeler celle de Klir, elle se distingue cependant de cette dernière car elle intègre explicitement le modèle du temps dans la description des spécifications. Elle se focalise ainsi sur la modélisation de systèmes dynamiques où la représentation du temps est une composante fondamentale. Toute la problématique de la modélisation d'un système dynamique consiste à un naviguer entre ces différents niveaux de spécifications.

2.8.2 Notion de morphisme

L'avantage de la hiérarchie que nous venons de présenter ne tient pas seulement à l'aide qu'elle peut apporter lors de la construction d'un modèle. Comme le souligne Zeigler, bien que la modélisation soit le nerf de la guerre dans la M&S, la principale difficulté consiste à

[4]Nous modérons ici quelque peu notre propos en précisant que de plus en plus de travaux utilisent les formalismes qui sont issus de cette théorie. Il n'en reste pas moins que ses aspects méthodologiques sont la plupart du temps ignorés dans notre communauté.

Tab. 2.2 – Hiérarchie des spécifications d'un système dynamique

Niveau	**Nom de la spécification**	**connaissances**
0	Observation	Les variables E/S d'intérêt et la façon de les mesurer suivant une base temporelle.
1	Comportement E/S	Un ensemble de variables datées collectées à partir du comportement du système source (des paires de valeurs E/S).
2	Fonctions E/S	Sachant un état initial, chaque entrée produit un unique résultat en sortie. (relations entre les E/S).
3	Transition d'état	L'évolution de l'état interne du système en fonction des entrées. La sortie générée par un état interne. (la structure interne du système).
4	Couplage Componentiel	Les composants système et la manière dont ils sont agencés : liaison entre ports E/S. (Une structure hiérarchique).

établir des relations entre les spécifications qui sont données pour différents systèmes. Nous verrons notamment qu'il est important d'établir des relations non ambiguës entre le modèle et son simulateur. Zeigler introduit ainsi la relation de *morphisme* entre deux spécifications différentes. Un morphisme est une relation d'équivalence qui met en correspondance les éléments de deux spécifications différentes pour chaque niveau de la hiérarchie. Ainsi, deux systèmes possèdent des spécifications qui vérifient une relation de morphisme pour un certain niveau de la hiérarchie lorsque :

- au niveau 0, si les entrées, les sorties et la base temporelle peuvent être mise en correspondance.
- au niveau 1, si les paires de valeurs E/S peuvent être associées une à une.
- au niveau 2, si les états initiaux peuvent être mis en correspondance de telle sorte que les relations établies sur les E/S soient les mêmes.
- au niveau 3, si les systèmes sont homomorphiques. La relation d'homomorphisme est vérifiée si l'on peut mettre en correspondance l'historique des changements d'état et de production de sortie des deux systèmes. Autrement dit, les deux systèmes passent, dans le même ordre chronologique, dans des états équivalents qui produisent des sorties qui sont égales (pour avoir les morphismes de niveau 1 et 2).
- au niveau 4, si les composants système peuvent être mis en correspondance de telle sorte qu'il existe une relation de morphisme entre eux. De plus, les couplages E/S qui existent entre des composants similaires sont les mêmes.

De plus, l'existence d'une relation d'équivalence entre deux systèmes pour un certain niveau implique l'existence d'une relation de morphisme dans le niveau inférieur.

Comme nous allons le voir dans la section suivante, la notion de morphisme est tout à fait centrale en ce qui concerne la définition des relations qui existent entre les différentes entités qui constituent le processus de simulation dans son ensemble.

2.8.3 Les entités de la M&S et leurs relations

Un des aspects méthodologiques majeurs de la théorie de la M&S repose, d'une part sur l'identification des différentes composantes qui constituent une expérience de simulation, et d'autre part sur l'étude des relations qui existent entre ces différentes entités. Il s'agit par là de donner des définitions précises aux différents concepts qui sont manipulés dans le domaine de la simulation informatique.

Nous avons par exemple mentionné plusieurs fois le terme **modèle** et le lecteur doit en avoir une intuition. Cependant celui-ci n'a pas encore de consistance technique. Et si nous voulons clairement poser notre problématique nous ne pouvons nous contenter de l'intuition. Cette section est donc consacrée à la présentation des notions clés de la M&S telles qu'elles sont présentées dans [Zeigler *et al.* , 2000] :

- le *système source* et sa *base de données comportementale*.
- le *cadre expérimental*.
- le *modèle*.
- le *simulateur*.
- la *relation de simulation*.
- la *relation de modélisation*.

Le système source et sa base de données comportementale

Le *système source* correspond à l'environnement, réel ou virtuel, qu'on souhaite modéliser. Il doit être vu comme une source de données observables qui constituent ce qui est appelé la *base de données comportementale*. On se situe ici au niveau 0 et 1 de la hiérarchie des spécifications. Par exemple, pour l'étude d'une colonie de fourmis particulière, on peut collecter des informations relatives à leur nombre, leurs fonctions, leur durée de vie, la qualité et la quantité de phéromones produites, etc. Par ailleurs, l'ensemble de ces informations est indissociable du cadre expérimental qui est défini suivant les objectifs du modélisateur.

Le cadre expérimental

Le *cadre expérimental* est une spécification des conditions d'observation du système ainsi que des objectifs du projet de simulation. Par exemple, dans le cas d'une colonie de fourmis, suivant les objectifs poursuivis on peut être intéressé ou non par certaines caractéristiques des individus : dans certains cas la vélocité des insectes sera prise en compte alors que dans d'autres cas ce paramètre n'aura aucun intérêt. Il est ainsi évident que, suivant les choix que fait le modélisateur, plusieurs cadres expérimentaux peuvent être définis sur un même système (système source et modèle). De même, un cadre expérimental peut être appliqué à plusieurs systèmes, ce qui permettra de les comparer.

Il est important de noter que le cadre expérimental influence très fortement la formulation du système source et le design du modèle qui en sera issu. De plus, comme nous le verrons, la validation du modèle est toujours relative à ce même cadre. C'est pourquoi une définition claire du cadre expérimental est cruciale dans le processus de simulation. Outre la définition des conditions expérimentales et des objectifs de la simulation, il peut aussi définir de quelle manière est évaluée l'efficacité de la simulation sur le plan qualitatif (la simulation est-elle satisfaisante ?) et quantitatif (rapidité d'exécution par exemple).

Le modèle

En général le terme modèle est indifféremment utilisé pour désigner plusieurs choses différentes comme par exemple la description abstraite ou concrète du phénomène à simuler, la technique mise en œuvre pour le faire, etc. Dans le contexte de la théorie de la M&S, le terme *modèle* désigne plus précisément la spécification du système effectuée aux niveaux 3 et 4 de la hiérarchie des spécifications. Ainsi, le *modèle* correspond à l'ensemble des instructions, lois, équations ou contraintes qui définissent la manière dont est généré le comportement E/S d'un composant système (niveau 3) et la manière dont les composants système sont assemblés (niveau 4). Autrement dit, le *modèle* spécifie les mécanismes de changement d'état dont nous avons parlés dans la section 2.3.

Le simulateur

Maintenant que nous avons une définition précise de ce que représente le modèle dans le processus de simulation, c'est-à-dire la spécification de l'ensemble des instructions qui permettent de générer le comportement du système, il nous faut une entité capable d'exécuter ces spécifications. Le terme *simulateur* désigne cette entité. Un *simulateur* désigne ainsi tout système de calcul capable d'exécuter le modèle de manière à générer son comportement. A partir de cette définition, on pourrait croire qu'il n'existe pas une différence fondamentale entre le modèle et son simulateur dans le sens où il s'agit dans les deux cas de générer le comportement du système et que, de plus, le modèle représente déjà une spécification de niveau élevé. Cependant, comme le souligne Zeigler, séparer le modèle de son simulateur comporte plusieurs avantages :

- un même modèle peut être exécuté par différents simulateurs, ce qui définit ainsi sa portabilité à haut niveau d'abstraction.
- les algorithmes mis en œuvre dans les simulateurs peuvent être étudiés en tant que tels et leur validité peut donc être établie.
- les ressources nécessaires pour correctement simuler un modèle donnent une mesure de sa complexité.

Outre les avantages conceptuels présentés ici, cette distinction va s'avérer tout à fait fondamentale dans le cadre de la simulation multi-agents. En effet, très peu de travaux de simulation basés sur le paradigme agent considèrent le simulateur comme une entité de base du processus de simulation. La plupart du temps l'implémentation du modèle multi-agents est considérée comme une étape de conception dont la validité serait donnée a priori. Et il s'agit sans aucun doute d'un point faible de cette approche. Notamment car ce domaine manque cruellement d'outils formels permettant de spécifier les modèles qu'il se propose de simuler. Nous n'en dirons ici pas plus étant donné que ce problème constitue une large part de la problématique qui sera abordée au cours de cette thèse et nous reviendrons en détail sur celui-ci dans le chapitre 4.

La relation de modélisation : validité du modèle

La *relation de modélisation* désigne la relation entre le *modèle*, le *système source* et le *cadre expérimental*. Cette relation définit la notion de *validité* de l'expérience en soulevant la question de la correspondance entre le *modèle* (et le comportement qu'il génère) et le *système source* dans le **contexte** du *cadre expérimental*. Autrement dit, il s'agit ici de savoir si la

modélisation qui est faite du système est une simplification acceptable de celui-ci en fonction des critères qualitatifs choisis et des objectifs de l'expérimentation. Cette comparaison pouvant être effectuée sur plusieurs niveaux de la hiérarchie des spécifications d'un système, Zeigler distingue trois relations de validité différentes :

- la *cohérence reproductive* est vérifiée si, pour toutes les expériences qui peuvent être conduites dans le *cadre expérimental*, le comportement du modèle correspond à celui du système étant donné un certain seuil de tolérance. Cette comparaison est donc effectuée au niveau 1 (relation E/S) de la hiérarchie des spécifications.
- la *cohérence évolutive*, qui implique la propriété précédente, correspond à la capacité du modèle à prédire des comportements du système qui sont encore inconnus. Il s'agit donc ici d'une cohérence de niveau supérieur qui suppose qu'il existe une correspondance entre le modèle et le système source au niveau 2, sur les fonctions E/S, de la hiérarchie des spécifications. Évidemment cette propriété ne peut être vérifiée qu'a posteriori ou bien si la propriété de *cohérence structurelle* est vérifiée.
- la *cohérence structurelle* est une propriété qui implique les deux précédentes mais qui en plus vérifie une correspondance au niveau 3 (mécanismes de changement d'état) et/ou au niveau supérieur (agencement des composants système entre eux). Autrement dit, le modèle ne se contente pas de reproduire à l'identique le comportement observable du système mais il imite aussi son fonctionnement intrinsèque pas à pas pour reproduire le mécanisme de changement d'état du système lui-même.

Etant données les questions de cohérence et de validité qu'elle soulève, l'analyse de la relation de modélisation est d'une importance capitale. Elle sera ainsi au cœur de la problématique que nous souhaitons soulever dans cette thèse. Nous verrons notamment que dans le contexte de la simulation multi-agents, les modèles employés font souvent abstraction, à tort, du cadre expérimental que ce paradigme implique.

Nous introduirons alors un autre niveau de cohérence dans la relation de modélisation pour décrire plus précisément notre problème : la notion de *cohérence paradigmatique*. Celle-ci s'intéresse plus particulièrement à la cohérence qui existe entre le paradigme de modélisation utilisé et le modèle considéré. Autrement dit, il s'agit de savoir si le modèle élaboré correspond aux critères sous-tendus par l'approche considérée. Ainsi, nous montrerons pourquoi la relation de modélisation n'est la plupart du temps pas vérifiée dans le domaine des simulations multi-agents si on prend en compte cette cohérence.

La relation de simulation : validité du simulateur

La *relation de simulation* désigne une relation de validité du *simulateur* par rapport au *modèle*. La question est d'obtenir la garantie que le simulateur qui est utilisé pour exécuter le modèle génère correctement le comportement de celui-ci. Autrement dit, il s'agit d'être certain que le simulateur reproduit sans ambiguïté les mécanismes de changement d'état qui sont formalisés dans le modèle et qu'il n'introduit pas de biais dans ce processus.

Ce qui signifie qu'il doit exister une correspondance entre le modèle et le simulateur au moins au niveau 2 de la hiérarchie des spécifications, c'est-à-dire sur les fonctions E/S . Il est important de noter qu'étant donné qu'on possède toujours les spécifications du modèle et du simulateur à ce niveau-là, puisqu'ils sont forcément construits lors du processus de simulation, la validité du simulateur peut être prouvée si on peut établir une relation d'homomorphisme entre ces deux systèmes.

Par ailleurs, il est important de comprendre que même lorsque l'implémentation des spécifications d'un modèle est sans ambiguïté, la validité d'un simulateur n'est pas donnée a priori car certains points du modèle restent toujours implicites. Par exemple, lorsqu'un modèle contient des processus stochastiques, les nombres aléatoires utilisés sont supposés produits par un générateur parfait qui n'est généralement pas spécifié dans le modèle. Il convient alors de vérifier la qualité du générateur aléatoire utilisé dans le simulateur pour pouvoir être sûr de sa validité.

De façon duale, l'implémentation d'un modèle dans des structures informatiques concrètes implique de considérer les contraintes qui sont liées à cet outil : on ne passe pas impunément du modèle sur papier à du code machine et il est nécessaire d'étudier les effets de bord qui peuvent être liés à la structure des programmes et à la nature digitale de l'ordinateur. Un même modèle peut donner des résultats différents suivant la taille mémoire utilisée pour stocker les réels : le nombre de chiffres après la virgule peut avoir son importance. Dans ce cas de figure, il est important que les spécifications du modèle intègrent ce genre de paramètres.

Il est ainsi tout à fait fondamental d'élaborer des spécifications en gardant à l'esprit qu'elles vont être implémentées dans un programme informatique. Comme nous le verrons, cette remarque constitue le deuxième aspect de la problématique que nous souhaitons aborder en ce qui concerne les simulations de système multi-agents. Trop souvent, les modèles qui sont proposés n'intègrent pas les problèmes qui sont posés par leur implémentation. Autrement dit, les spécifications relatives à un modèle multi-agents comportent souvent un grand nombre d'ambiguïtés quant à la manière dont elles doivent être implémentées. La conséquence directe est qu'une même spécification peut être implémentée de plusieurs façons et conduire ainsi à des résultats expérimentaux différents. Ce qui remet en cause la validité de l'expérience. Dans le chapitre 4, nous identifierons cette problématique en illustrant par ce que nous appellerons des *phénomènes de divergence implémentatoire* et nous montrerons pourquoi de nombreux modèles multi-agents sont sujets à ce problème.

Récapitulatif

Dans cette section nous avons vu les entités de base de la M&S et les deux relations fondamentales qui existent entre elles. La figure 2.9 résume de manière schématique ces différentes notions.

Par ailleurs, cette figure montre que la problématique nous souhaitons aborder dans cette thèse se situe au niveau des deux relations fondamentales qui ont été définies : la relation de modélisation et la relation de simulation. Ainsi, comme nous l'avons suggéré dans le chapitre introductif, nous aborderons la question de la validité des modèles qui sont proposés dans les simulations multi-agents (relation de modélisation) et les problèmes liés à leurs implémentations (relation de simulation).

2.9 Résumé du chapitre

Dans ce chapitre voué à la présentation de la simulation informatique des systèmes dynamiques, nous avons tout d'abord présenté une vision globale de la nature des objectifs de ce processus expérimental. Nous avons donné une représentation générique des systèmes dynamiques (la boîte noire) et les principales problématiques liées à leur étude. Nous avons ensuite exposé les principales catégories de modèle employés pour la représentation du temps et nous

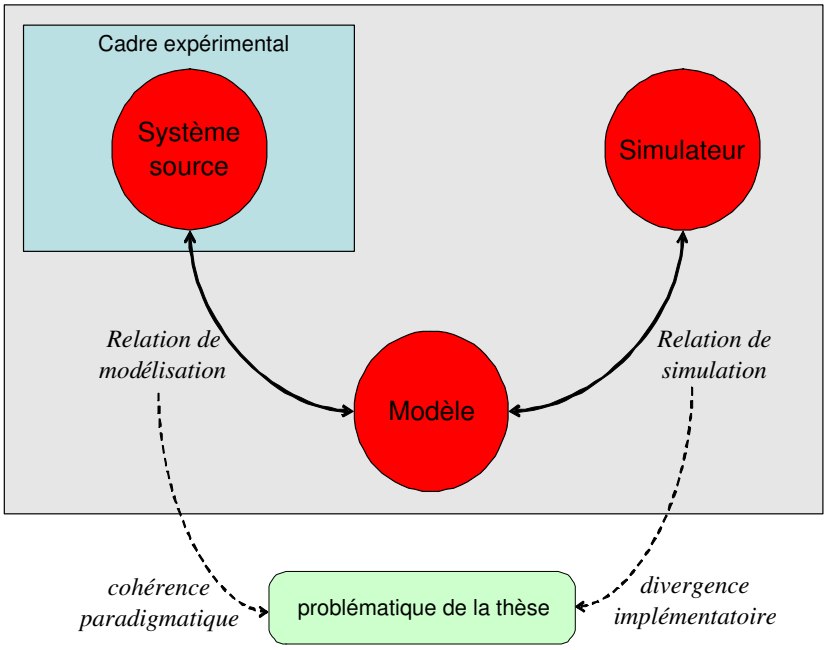

FIG. 2.9 – Les entités de base de la M&S et leurs relations.

avons succinctement abordé les principes d'implémentation qui leur correspondent. Nous avons aussi consacré une large part de ce chapitre à présenter quelques-unes des notions présentes dans la théorie de la M&S de Zeigler. Ce qui nous a permis de positionner informellement la problématique de notre thèse dans le cadre de cette théorie. Le chapitre suivant est consacré à la présentation des simulations multi-agents.

Chapitre 3

Les systèmes multi-agents et la simulation

DANS ce chapitre nous allons tout d'abord présenter le contexte du domaine de recherche dans lequel cette thèse se situe : celui des systèmes multi-agents. Nous verrons ensuite quelles sont les motivations qui se cachent derrière la modélisation et la simulation de ces systèmes. Après cela, nous donnerons des exemples de plates-formes à travers une description succincte de quelques domaines où le paradigme agent est utilisé. Dans les sections 3.4 et 3.5, nous exposerons les principales questions soulevées par l'implémentation d'un simulateur multi-agents. Notre but sera d'attirer l'attention du lecteur sur les nombreux paramètres qui participent à la complexité de ce type de logiciels.

3.1 Le paradigme agent

3.1.1 Définition générale

Qu'est-ce qu'un agent ? Répondre à cette question est aujourd'hui encore quelque peu embarrassant car il n'existe pas de définition formelle relative à l'implémentation, comme c'est le cas pour le monde des objets par exemple. Cette situation est d'ailleurs à l'origine du manque de crédibilité dont les systèmes multi-agents peuvent parfois souffrir auprès des autres domaines liés à l'informatique : il est en effet assez difficile d'exposer clairement quels sont les avantages techniques qui sont liés à cette approche.

La programmation orientée agent est donc aujourd'hui beaucoup plus une manière de penser un système informatique qu'une technique d'implémentation particulière. Ainsi, la plupart des définitions qui ont été proposées pour le terme agent s'attachent à décrire des principes généraux liés à cette approche [Ferber, 1999, Jennings & Wooldridge, 1999, Briot & Demazeau, 2001a]. Parmi ces principes, certains semblent aujourd'hui faire l'objet d'un consensus au sein de la communauté multi-agents. Nous utiliserons donc la définition exposée dans [Wooldridge & Jennings, 1995, Wooldridge & Ciancarini, 2000] qui résume ces principes[1]. Par le terme *agent*, on désigne un système qui possède les propriétés suivantes :

[1] Cette définition est aussi connue pour être ce qu'on appelle la *définition faible* d'un agent (weak definition) [Wooldridge & Jennings, 1995] dans le sens où, en ne rentrant pas dans les détails, elle est supposée englober l'ensemble des entités qui peuvent être taxées d'agent. Il est ailleurs assez amusant de se souvenir que cette définition était accompagnée d'une *définition forte* de la notion d'agent qui n'a pas eu du tout le même succès.

– *autonome* : un agent possède un état interne sur lequel il possède un contrôle total. Cet état interne est notamment **inaccessible** aux autres agents. De plus, l'agent prend des décisions qui sont basées sur cet état interne sans intervention extérieure (humaine ou d'un autre agent).
– *réactif* : un agent est *situé* dans un environnement. Il est capable de *percevoir* cet environnement et de *réagir* aux changements qui y interviennent par ses actions.
– *social* : un agent est une entité sociale dans le sens où il est capable d'*interagir* avec les autres agents via son environnement.
– *proactif* : un agent ne fait pas que réagir à son environnement mais il est aussi capable de produire de lui-même des actions motivées par des buts.

Bien que cette définition soit très générale, elle est cependant parfois inadaptée dans certaines situations, notamment en ce qui concerne la dernière propriété. Une grande partie de la communauté travaille par exemple sur une catégorie d'agent dits *réactifs*, par opposition à *cognitifs*, qui ne font que réagir à leur environnement sans avoir véritablement de buts explicitement motivés [Ferber, 1999]. Encore faut-il être d'accord sur la signification du terme *but*.

On voit ici toute la difficulté de donner une définition consensuelle au concept d'agent et celui-ci reste assez flou quant à son implémentation. Pour l'instant, nous nous contenterons de cette définition. Nous aurons l'occasion de revenir en détail sur l'interprétation de ces différentes propriétés lorsque nous aborderons les problèmes liés à la programmation de ces entités. Nous verrons notamment que la propriété d'autonomie, fondamentale au niveau conceptuel, soulève des questions importantes quant à son implémentation.

3.1.2 Un agent est un système dynamique

De la définition précédente, il est facile de voir qu'un agent est un système dynamique au sens de la théorie des systèmes. En effet, il possède un état interne qui change en fonction de ses perceptions (entrées) et on parle du comportement d'un agent pour qualifier la manière dont il réagit à son environnement (relation E/S). On le représente d'ailleurs souvent de manière abstraite à l'aide du schéma suivant (figure 3.1) [Wooldridge, 2000, Russell & Norvig, 2003] :

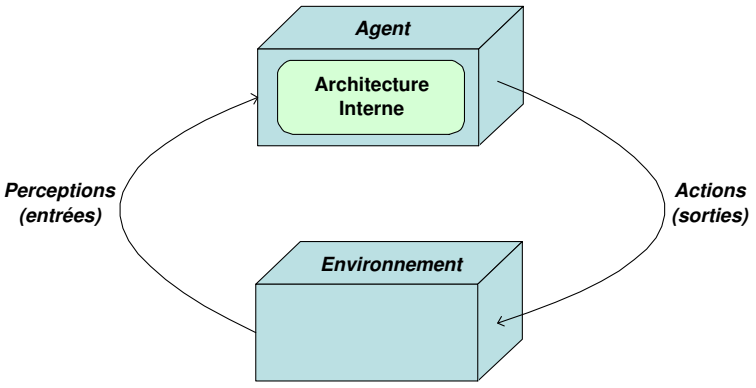

FIG. 3.1 – Représentation classique d'un agent et de son environnement.

Comme le montre cette figure, un agent est défini par un ensemble de *perceptions* (entrées), un ensemble d'*actions* (sorties) et on parle de l'*architecture interne* de l'agent pour désigner les mécanismes qui définissent sa dynamique intrinsèque. On utilise aussi fréquemment les

termes *senseurs* et *effecteurs* pour désigner les moyens par lesquels un agent interagit avec son environnement.

3.1.3 Pas d'agent sans environnement

Toujours concernant les propriétés conceptuelles que nous avons énoncées, et comme le suggère la figure précédente, on voit que le concept d'agent ne se suffit pas à lui-même : il est indissociable de celui d'environnement. En effet, l'ensemble des perceptions et des actions qu'un agent est susceptible de réaliser est entièrement défini par rapport à l'environnement où celui-ci va opérer. L'environnement définit ainsi les conditions d'existence de ce genre d'entités [Odell *et al.* , 2002]. Autrement dit, il n'est pas possible de parler d'agent sans parler d'environnement [Weyns *et al.* , 2005b].

Bien que cette opinion soit consensuelle, de nombreux travaux ont parfois tendance à négliger l'étude de l'environnement car ils considèrent des systèmes dans lesquels les agents ne font que communiquer par l'intermédiaire de messages de type courrier électronique[2] et où il n'existe pas à proprement parler d'environnement physique.

Cependant, même dans ce cas précis les échanges d'informations s'effectuent nécessairement par l'intermédiaire d'un média de communication. Ce média possède ses propres caractéristiques et constitue en soi un environnement dans lequel il existe des contraintes. De plus, l'envoi de messages et la consultation de la boîte aux lettres doivent être considérés respectivement comme des actions et des perceptions. En conséquence, l'architecture interne d'un agent est très fortement influencée par la nature de l'environnement et vice versa. Nous verrons dans la section 3.4 que cette influence mutuelle engendre bien sûr des contraintes importantes sur l'implémentation. Nous montrerons alors pourquoi il est fondamental de faire une analyse pointue des différents aspects de cette interdépendance.

Russel et Norvig ont proposé de distinguer plusieurs propriétés environnementales dont voici les plus importantes dans le contexte de la simulation [Russell & Norvig, 2003] :

- *accessible* ou *inaccessible* : un environnement est dit accessible lorsqu'il est possible à un agent d'obtenir la totalité des informations qui concernent son état. En général, un agent est toujours considéré comme une entité qui ne peut accéder que partiellement et localement aux informations qui décrivent l'environnement. De façon duale, les actions qu'il effectue n'ont en principe qu'une portée locale.
- *déterministe* ou *non déterministe* : dans un environnement déterministe, une même action aura toujours le même effet et il n'existe pas d'incertitude quant à son résultat. Ce qui n'est pas le cas pour les environnements non déterministes.
- *statique* ou *dynamique* : un environnement est dit statique[3] lorsqu'il ne change que sous l'impulsion des actions effectuées par les agents. Dans le cas contraire, l'environnement définit aussi ses propres processus d'évolution et son état peut être modifié sans qu'aucune action ne soit intervenue, sous l'effet de la gravitation par exemple. On parle alors de l'évolution endogène de l'environnement.
- *discret* ou *continu* : on parle d'environnement discret lorsqu'il existe un nombre fixé de perceptions et d'actions possibles sur lui. Dans le cas contraire il sera continu (si les distances entre individus sont modélisées à l'aide d'un réel par exemple).

[2] Ce type d'agent est souvent désigné sous les termes d'*agent purement communicant*.

[3] L'environnement reste cependant lui aussi un système dynamique dans la mesure où il évolue en fonction des entrées qui lui sont données.

3.1.4 Architecture interne d'un agent

Comme nous l'avons dit précédemment, l'architecture interne d'un agent fait référence aux mécanismes qui modélisent sa dynamique. Dans le domaine des systèmes multi-agents, on parle aussi souvent du *système décisionnel* ou *délibératif* de l'agent. Ainsi, un agent est généralement considéré comme un processus cyclique comportant trois phases successives : *perception*, *délibération* et *action* [Ferber, 1999] (figure 3.2).

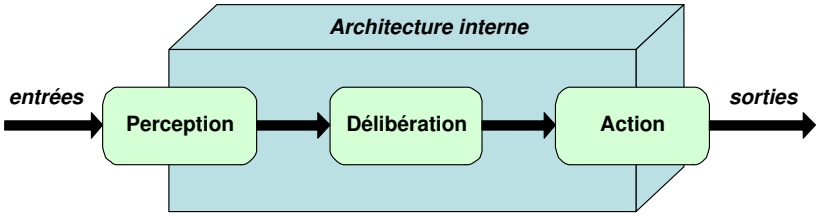

FIG. 3.2 – L'agent, un processus cyclique à trois phases : *perception*, *délibération* puis *action*.

Cette terminologie met en exergue la principale caractéristique d'un agent : son évolution n'est pas entièrement régie par les règles environnementales du milieu où il se trouve, au contraire des autres objets du monde qui ne font que subir ces règles. Autrement dit, un agent est un système dont l'évolution ne se résume pas à l'application de lois naturelles posées une fois pour toutes (comme les lois de la physique par exemple) : un agent délibère, c'est-à-dire qu'il prend des **décisions** suivant des règles **subjectives** susceptibles de changer au cours du temps. Ainsi, du point de vue de la théorie des systèmes par exemple, un agent est vu comme un système dynamique particulier dont la structure interne peut évoluer : c'est un *modèle à structure variable* [Zeigler & Ören, 1986]. En effet, un agent change de comportement de manière autonome et il peut donc ne pas réagir de la même façon aux mêmes stimuli. D'un point de vue génie logiciel, un agent peut être vu comme un système computationnel réflexif, c'est-à-dire un système qui a la faculté de représenter, contrôler et modifier son propre comportement [Ferber & Carle, 1991].

Le processus de délibération d'un agent constitue la partie la plus essentielle de son architecture interne dans la mesure où il définit ses caractéristiques comportementales. De fait, c'est sans conteste la partie la plus étudiée et pour laquelle il existe de nombreux formalismes. Ainsi, la nature de l'architecture interne d'un agent est souvent caractérisée par celle de son processus décisionnel, bien plus que par ses mécanismes de perceptions et d'actions. Dans ce contexte, on distingue de manière plus ou moins heureuse plusieurs types d'architecture interne suivant la nature et le degré de rationalité du processus décisionnel encapsulé dans un agent. Classiquement, la communauté multi-agents distingue deux grandes catégories : les architectures *cognitives* et les architectures *réactives*.

Architecture cognitive

Informellement, on dit d'un agent qu'il est cognitif si les décisions qu'il prend sont motivées par des buts explicites et qu'elles exhibent une certaine forme de rationalité basée sur une représentation symbolique ou logique du monde. Ce type d'agent est parfois qualifié d'*intelligent* ou de *rationnel*. Autant de termes dont on sait qu'ils doivent être manipulés avec précaution dans le domaine de l'informatique. L'une des architectures cognitives les plus connues est sans aucun doute l'architecture BDI (*Believe Desire Intention*) [Rao & Georgeff, 1992] qui, comme

son nom l'indique, consiste à élaborer un processus décisionnel qui utilise une logique basée sur les notions de croyances, de désir et d'intention. La figure 3.3 est un exemple d'architecture BDI. Celui-ci est tiré d'un cours en ligne interactif sur les systèmes multi-agents qui est accessible à l'adresse http ://turing.cs.pub.ro/auf2/[4].

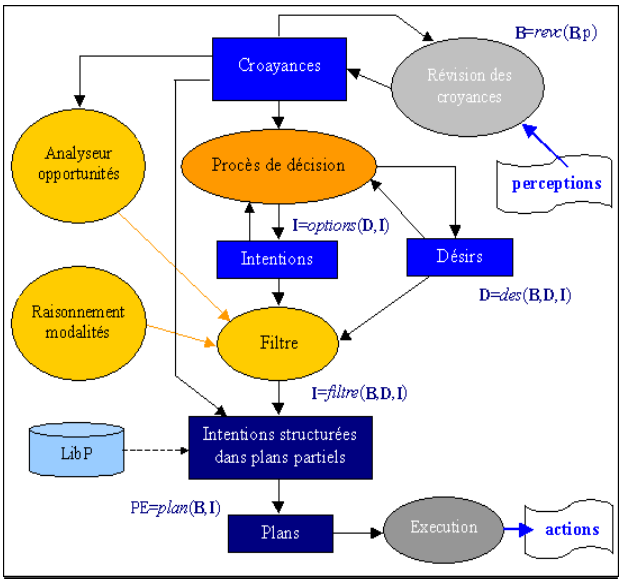

FIG. 3.3 – Un exemple d'architecture BDI.

Architecture réactive

Au contraire, les agents purement réactifs sont vus comme des entités qui ne font que réagir de manière mécanique aux stimuli qu'ils perçoivent. Ils n'ont pas de buts explicites. Pour autant, il ne faut pas en déduire que les agents réactifs ne peuvent exhiber un comportement qui, pour un observateur extérieur, ne peut être considéré comme intelligent. Le classique exemple de la fourmilière en est l'illustration : les fourmis n'ont a priori pas de capacités cognitives mais elles accomplissent cependant des tâches extrêmement complexes. Ainsi, l'idée de l'approche réactive est qu'il n'est pas nécessaire de concevoir des agents ayant une architecture interne complexe de haut niveau pour finalement obtenir un comportement que l'on puisse qualifier d'intelligent [Brooks, 1991, Drogoul, 1993]. Un exemple d'architecture couramment utilisée pour concevoir des agents réactifs est celui de l'architecture de *subsomption*[5] proposée par Brooks [Brooks & Connell, 1986]. Le principe de cette architecture est de concevoir le processus décisionnel en considérant un ordre de priorité entre des modules qui représentent les tâches pouvant être accomplies par l'agent. Les conditions d'activation (calculées en fonction des perceptions) de chaque module sont alors évaluées suivant cet ordre de manière à sélectionner la tâche à effectuer en priorité. Par exemple, il peut être vital pour un robot de se recharger en énergie si ses batteries faiblissent. Si tel est le cas, c'est tâche devient donc prioritaire. Dans le cas contraire, le robot évalue les conditions d'activation de la tâche de priorité immédiatement inférieure et ainsi de suite (figure 3.4. Figure issue du même cours en ligne que précédemment, voire note 4).

[4] Ce cours francophone est issu d'une collaboration entre deux universités roumaines (Bucarest et Suceava) et l'université Paris XIII. Les auteurs en sont Adina Florea, Daniel Kayser, Stefan Pentiuc et Amal El Fallah Segrounichi.

[5] Il est cependant possible de l'utiliser pour concevoir des agents cognitifs.

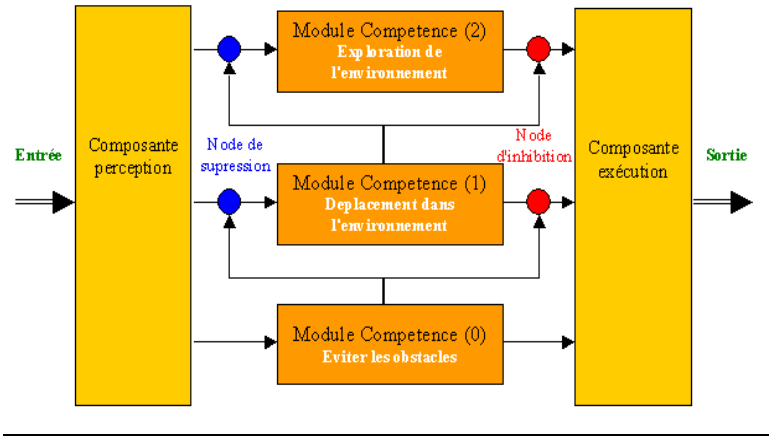

FIG. 3.4 – Un exemple de l'architecture de subsomption.

Approche formelle : agents *tropiques* et agents *hystérétiques*

Bien qu'il soit possible de distinguer facilement un agent purement réactif d'un agent purement cognitif, la frontière entre ces deux types d'architecture est parfois assez floue. Finalement, il s'agit plus d'une différenciation informelle entre deux approches de conception que d'un véritable moyen de distinguer une architecture interne par rapport à une autre. Pour faire une distinction claire entre différentes architectures internes, il est beaucoup plus intéressant de s'attarder sur les moyens concrets utilisés par l'agent pour faire sa décision.

Dans cette optique, Genesereth et Nilsson ont été les premiers à proposer une représentation algébrique de l'architecture interne d'un agent qui permette de distinguer formellement deux grandes catégories d'agent : les *agents tropiques* et les *agents hystérétiques* [Genesereth & Nilsson, 1987]. La première catégorie fait référence à des agents qui ne possèdent pas d'état interne et qui agissent donc de manière réflexe en fonction de leurs perceptions. Ils n'ont ainsi pas la capacité de mémoriser de l'information et donc de prendre des décisions en fonction d'un historique.

Plus formellement, soit Σ l'ensemble des états possibles du monde, P_a l'ensemble des perceptions d'un agent a et A_a l'ensemble des actions que l'agent a peut effectuer, le processus décisionnel d'un agent tropique $Behaviour_a : \Sigma \mapsto A_a$ se décompose en deux fonctions qui s'appliquent de manière séquentielle :

- la fonction de perception, $Percept_a : \Sigma \mapsto P_a$, qui associe un percept à un état du monde.
- la fonction réflexe, $Reflexe_a : P_a \mapsto A_a$, qui à chaque perception associe une action.

Au contraire, les agents hystérétiques possèdent un état interne qu'ils utilisent pour mémoriser des informations et pour prendre leurs décisions. Soit $s_a \in S_a$ l'état interne d'un agent, le processus décisionnel de l'agent peut ainsi être décomposé suivant les fonctions suivantes :

- la fonction de perception, $Percept_a : \Sigma \mapsto P_a$.
- la fonction de mémorisation, $Mem_a : P_a \times S_a \mapsto S_a$, qui en fonction de la perception et de l'état interne courants calcule le nouvel état interne de l'agent.
- la fonction de décision, $Decision_a : P_a \times S_a \mapsto A_a$, qui décide de l'action à effectuer en fonction du nouvel état interne et de la perception courante.

Il faut noter que le formalisme que nous venons de présenter ne correspond pas exactement à celui proposé par Genesereth et Nilsson où il est considéré que le comportement d'un agent modifie directement l'état du monde : $Behaviour_a : \Sigma \mapsto \Sigma$. A l'instar de [Wooldridge, 2000], nous préférons ici définir l'ensemble des actions qu'un agent peut effectuer. En effet, nous verrons plus tard qu'il existe plusieurs façons de définir le résultat d'une action et que celui-ci ne se traduit pas forcément par la modification directe de l'état du monde. Nous verrons ainsi une adaptation de ce formalisme dans le chapitre 6. Ceci n'est d'ailleurs ici pas très important pour comprendre la différence fondamentale qui existe entre les agents tropiques et les agents hystérétiques : les premiers ne possèdent pas d'état interne alors que les seconds sont capables de mémoriser de l'information, d'où une différence notable dans la manière de concevoir leur système décisionnel et donc leur architecture interne.

3.2 Qu'est-ce qu'un système multi-agents ?

3.2.1 Plusieurs agents

Dans une première approche, on peut considérer qu'un système multi-agents est tout simplement un ensemble d'agents partageant un environnement commun. On peut donc représenter schématiquement un système multi-agents à l'aide de la figure 3.5.

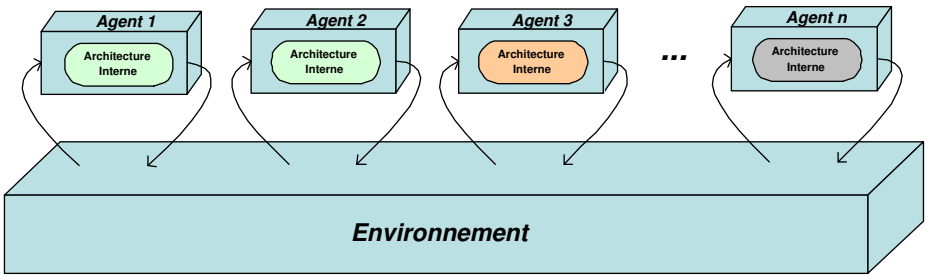

FIG. 3.5 – Représentation schématique d'un système multi-agents.

Ce schéma fait clairement apparaître le rôle fondamental de l'environnement dans un système multi-agents. En effet, si la notion d'environnement joue un rôle central dans la définition de ce qu'est un agent, elle prend une dimension encore supérieure lorsqu'on considère plusieurs individus. L'environnement doit non seulement définir les conditions d'existence de chaque entité, mais il met aussi en relation l'ensemble des agents : **il est le lieu de leurs interactions**. A ce propos, il est important de voir que le schéma précédent est valable du point de vue d'un observateur extérieur. Du point de vue d'un agent, tout ce qui ne concerne pas son architecture interne fait partie de l'environnement, notamment les autres agents. Nous verrons dans la section 3.4 toute l'importance de cette remarque et les implications qu'elle a sur l'implémentation. Nous justifierons alors en détail ce point de vue.

3.2.2 A + E + I + O : l'approche VOYELLES

Dire d'un système multi-agents qu'il se résume à la mise en commun d'un ensemble d'agents est bien sûr insuffisant. Cependant on peut difficilement se raccrocher à l'étude de leurs implémentations pour les définir avec plus d'acuité. En effet, de la même manière que nous avons des difficultés à définir ce que doit être le code d'un agent, il est très difficile de caractériser les

structures informatiques qui sont utilisées pour programmer des systèmes multi-agents. Pour aller plus loin dans la définition de ce qu'est un système multi-agents, il est plus intéressant de dégager certains aspects fondamentaux qui le caractérisent et le distinguent d'un système mono-agent.

Dans ce cadre, Demazeau propose de définir un système multi-agents comme un ensemble d'**agents** potentiellement **organisés** qui **interagissent** dans un **environnement** commun [Demazeau, 1995, Demazeau, 2001]. Cet énoncé fait clairement apparaître deux dimensions supplémentaires : l'interaction et l'organisation. L'approche VOYELLES consiste ainsi à considérer que l'analyse, le design, l'implémentation et le déploiement d'un système multi-agents peuvent être étudiés en fonction de quatre aspects fondamentaux : **A**gents, **E**nvironnement, **I**nteraction et **O**rganisation. En voici une description simplifiée :

- **A**gents : architectures internes des agents.
- **E**nvironnement : le milieu dans lequel évoluent les agents.
- **I**nteraction : les moyens par lesquels les agents interagissent.
- **O**rganisation : les moyens utilisés pour structurer l'ensemble des entités.

L'idée sous-tendue par cette approche est que chacune de ces briques définit une problématique particulière qui doit être explicitement étudiée dans chaque étape de la conception, de la phase d'analyse à l'implémentation. La plate-forme de développement VOLCANO est basée sur cette méthodologie [Ricordel & Demazeau, 2002]. Elle a par exemple été utilisée dans [da Silva & Demazeau, 2002] où les auteurs appliquent la décomposition VOYELLES pour l'étude de la coordination dans le contexte de la RoboCup.

Dans le cadre de notre thèse, nous adopterons nous aussi une telle démarche en décomposant les différents aspects liés à la modélisation et à la simulation d'un système multi-agents. Nous verrons alors qu'il convient de rajouter à cette décomposition un aspect lié au problème de la modélisation du temps.

Nous allons maintenant nous restreindre à la présentation des systèmes multi-agents dans le cadre de la simulation. Nous invitons le lecteur qui souhaite avoir un point de vue plus global du domaine à consulter des ouvrages tels que [Ferber, 1999, Weiss, 2000, Briot & Demazeau, 2001b] par exemple. Il est aussi intéressant de consulter des sites consacrés à ce sujet tels que www.agentlink.org ou multiagent.com, pour ne donner que ces deux-là.

3.3 Modélisation et simulation multi-agents

3.3.1 Motivations

Difficultés des modélisations classiques

Les premiers modèles mathématiques utilisés pour représenter des systèmes dynamiques modélisant une population d'entités individuelles, comme celui de Volterra, ont été critiqués pour l'étanchéité de leur niveau d'analyse. En effet, dans ce type de modèles l'ensemble du système est globalement représenté par des équations définies au niveau macroscopique qui ne permettent pas de prendre en compte les caractéristiques individuelles des entités qui composent le système [Ferber, 1999]. De plus, ce niveau d'analyse pose la question du réalisme et de la complexité des paramètres utilisés pour formuler les équations qui modélisent le système et met en cause la validité de l'approche [Buettner & Siler, 1976].

Modélisation *individus-centrée*

La première approche qui a tenté d'apporter une réponse à ce problème, en introduisant plusieurs niveaux d'analyse dans l'élaboration du modèle, est celle proposée par la micro-simulation [Orcutt, 1957]. On trouve une description des différentes techniques liées à cette approche dans [Gilbert & Troitzsch, 1999]. Le principe de base de cette technique consiste à intégrer le niveau microscopique en élaborant des équations stochastiques qui utilisent des caractéristiques individuelles pour modéliser des prises de décision. Bien que la microsimulation trouve son origine dans l'analyse des systèmes sociaux, cette démarche peut être considérée comme le point de départ des approches dites *individus-centrées* (dont on peut considérer que la simulation multi-agents est une sous-classe particulière) qui sont aujourd'hui utilisées dans d'autres domaines comme la biologie par exemple [Grimm, 1999].

Modélisation *multi-agents*

L'approche multi-agents se fonde elle aussi sur une démarche individus-centrée mais elle va plus loin. En effet, elle considère qu'il est possible de modéliser, non seulement les individus et leurs comportements, mais aussi les **interactions** qui se déroulent entre ces individus. Elle considère ainsi que la dynamique globale d'un système, au niveau macroscopique, résulte directement des interactions entre les individus qui composent ce système au niveau microscopique. Ainsi, alors que les modèles classiques modélisent les relations qui existent entre les différentes entités identifiées d'un système à l'aide d'équations mathématiques, l'approche multi-agents modélise directement les interactions engendrées par des comportements individuels [Parunak *et al.* , 1998, Klügl *et al.* , 2002].

3.3.2 Domaines d'application

Avec de telles prétentions, il n'est pas étonnant que les simulations multi-agents soient utilisées dans des domaines extrêmement variés. Dans cette section, nous allons présenter quelques-uns de ces domaines et quelques-unes des plates-formes de simulation qui s'y rattachent.

Robotique mobile collective RMC

Comme l'on pouvait s'y attendre, le domaine de la RMC est très demandeur de simulations multi-agents. En effet, dans ce domaine, la mise en œuvre d'expérimentations en conditions réelles peut s'avérer extrêmement longue et coûteuse. Il est alors très intéressant de disposer d'un outil de simulation qui permette une première analyse des algorithmes comportementaux qui seront implémentés dans les robots réels. De plus, la simulation permet de s'affranchir des contraintes techniques liées au monde réel et donc de se focaliser sur un problème particulier. Dans le cadre de la RMC, la problématique est ainsi le plus souvent la *création d'un système*. Il s'agit, suivant un objectif précis, d'élaborer l'architecture qui sera la mieux adaptée à la résolution d'un problème particulier. Parmi les problèmes classiques les plus étudiés par simulation, on peut citer le problème des robots fourrageurs[6] [Steels, 1990] qui a notamment été étudié par Drogoul [Drogoul, 1993] et plus récemment par Simonin [Simonin, 2001]. On peut aussi citer le problème des robots footballeurs qui a été étudié par Magnin qui propose

[6] Dans ce type de problèmes, les robots sont chargés de récupérer dans l'environnement des objets qu'il leur faut ramener dans un lieu précis.

un simulateur basé sur un principe événementiel appelé SIEME [Magnin, 1996]. Ce problème est aujourd'hui incarné par une compétition appelée RoboCup[7] [Kitano *et al.* , 1997]. Une des plates-formes les plus abouties est sans doute MISSIONLAB[8] qui a été développée par Arkin et son équipe [MacKenzie *et al.* , 1997]. Actuellement dans sa sixième version, il s'agit d'une plate-forme qui permet à la fois de réaliser des simulations et de contrôler des robots réels.

Ethologie

L'éthologie, discipline consacrée à l'étude du comportement animal, est un domaine qui se prête naturellement à l'utilisation des techniques de simulation multi-agents. Dans ce contexte, la problème consiste généralement à *inférer sur le système* que l'on souhaite étudier. Le plus souvent, on dispose de données d'observation sur le comportement d'un système social et l'objectif consiste à tenter de modéliser des mécanismes comportementaux qui produisent, par simulation, une dynamique qui corresponde aux observations. La thèse de Drogoul constitue une introduction exhaustive à cette problématique [Drogoul, 1993]. Cette thèse a d'ailleurs donné lieu à l'élaboration d'une plate-forme de simulation appelée MANTA [Drogoul & Ferber, 1992]. De plus, bien que cette plate-forme ait tout d'abord été conçue dans le but d'étudier une colonie de fourmis, elle a par la suite donné lieu à un principe générique de modélisation de comportements réactifs que Drogoul appelle l'*EthoModélisation*. Ce principe a ainsi utilisé dans d'autres domaines d'application, notamment la RMC avec le projet MICROBES[9] [Drogoul & Picault, 1999]. Il est d'ailleurs très intéressant de remarquer que l'utilisation de travaux liés à l'éthologie dans d'autres domaines est assez fréquente. Le règne animal constitue en effet une source d'inspiration intarissable pour de nombreux domaines comme le montre la très prisée conférence internationale SAB[10], *Simulation of Adaptative Behavior : From Animals to Animats*[11]. On peut aussi citer le travail de Travers qui propose une plate-forme de simulation basée sur un modèle discret qui s'inspire des problématiques liées à l'éthologie et à la modélisation d'animats [Travers, 1996].

Ecologie et biologie

Dans le monde d'aujourd'hui, les problématiques liées à la gestion de l'environnement prennent de plus en plus d'importance. Dans ce contexte, les techniques de simulations multi-agents permettent de modéliser directement les actions d'un ensemble d'acteurs (humains et/ou animaux) sur leur environnement. Le problème de la gestion de ressources renouvelables par l'homme est par exemple à l'origine de l'élaboration de la plate-forme de simulation multi-agents CORMAS[12] [Bousquet *et al.* , 1998]. Cette plate-forme a été utilisée pour de nombreux projets qui visent à étudier les interactions entre ressources et sociétés dont on trouve une liste sur le site Internet de la plate-forme. Parallèlement à cette problématique, l'écologie a aussi inspiré des architectures de simulation basées sur la notion d'écosystème comme par exemple la plate-forme ECHO[13] qui propose un modèle d'agent fondé sur des comportements liés à la survie dans un milieu contenant des ressources limitées (compétition, échange, re-

[7] www.robocup.org.
[8] www.cc.gatech.edu/ai/robot-lab/research/MissionLab.
[9] miriad.lip6.fr/microbes.
[10] www.isab.org.uk/confs.
[11] Le terme *animat* est souvent utilisé pour désigner un agent cybernétique ou informatique dont la conception est inspirée du monde animal.
[12] cormas.cirad.fr/index.htm.
[13] www.santafe.edu/projects/echo/echo.html.

production) [Hraber *et al.* , 1997]. Un peu plus générique, la plate-forme MOBIDYC[14] est un outil de simulation principalement dédié à des études liées à l'écologie, la biologie et l'environnement [Ginot & Page, 1998]. On peut aussi citer le projet GEAMAS[15] qui repose sur une architecture logicielle générique pour la modélisation et la simulation de systèmes complexes naturels [Marcenac & Giroux, 1998, Courdier *et al.* , 1998]. Objet de nombreuses évolutions [Soulié, 2001], cette plate-forme est aujourd'hui basée sur une architecture modulaire très intéressante qui fait une distinction nette entre la modélisation du système conatif de l'agent et la modélisation de l'interaction de celui-ci avec son environnement. Dans cette thèse, nous reviendrons à plusieurs reprises sur l'importance d'une telle distinction dans le cadre du paradigme agent.

Sciences sociales

Cela fait bien longtemps que le terme agent est utilisé dans le monde des sciences sociales et il est tout à fait naturel que les disciplines qui leurs sont associées se tournent vers les simulations multi-agents pour disposer d'un outil d'analyse supplémentaire. Dans les références qui suivent, on trouve les motivations et les principales problématiques liées à l'utilisation des systèmes multi-agents dans ce domaine [Conte *et al.* , 1998, Axtell, 2000b, Goldspink, 2002, Gilbert & Troitzsch, 1999]. Pour donner quelques exemples de cette utilisation, il existe des travaux qui concernent la théorie des jeux [Beaufils *et al.* , 1998], la modélisation de phénomènes urbains [Vanbergue *et al.* , 2000], la modélisation de la dynamique du changement d'opinion dans une population [Deffuant *et al.* , 2002] pour n'en citer que quelques-uns. On trouve de très nombreux autres exemples dans la revue électronique JASSS[16], *The Journal of Artificial Societies and Social Simulation*. En ce qui concerne les travaux qui proposent des modèles génériques de simulation liés aux sciences sociales, on peut citer les plates-formes de simulation MODULECO[17] [Phan, 2002], ASCAPE[18] [Parker, 2001] et le langage de modélisation SDML[19] qui propose une architecture agent basée sur des règles logiques [Moss *et al.* , 1998]. Il faut aussi noter l'existence de la très récente plate-forme REPAST[20] qui semble avoir de plus en plus de succès [Collier, 2002].

Enseignement et apprentissage des systèmes multi-agents

Dans le cadre de l'enseignement et de l'apprentissage des systèmes complexes, la simulation constitue indéniablement un outil pédagogique d'un grand intérêt. Ainsi, certaines plates-formes de simulation multi-agents sont explicitement conçues pour cet usage. L'idée est de fournir aux utilisateurs un monde virtuel et une architecture agent (un ensemble de primitives comportementales) prédéfinis de manière à ce que seuls les comportements de haut niveau des agents reste à définir. La simulation permet alors de constater immédiatement le résultat des interactions entre ces comportements. C'est dans cette optique que la plate-forme STARLOGO[21], basée sur le langage Logo, a été développée par Resnick au *Massachusetts Insti-*

[14] www.avignon.inra.fr/mobidyc.
[15] www2.univ-reunion.fr/∼courdier/mas2.
[16] jasss.soc.surrey.ac.uk/JASSS.html.
[17] www-eco.enst-bretagne.fr/∼phan/moduleco.
[18] www.brook.edu/es/dynamics/models/ascape/default.htm.
[19] sdml.cfpm.org.
[20] repast.sourceforge.net.
[21] education.mit.edu/starlogo

tute of Technology (MIT) [Resnick, 1994]. Le principe du langage Logo[22] consiste à manipuler un animat graphique, une *tortue*[23] (*turtle*), en lui donnant des commandes très simples via l'interface graphique. Par exemple, la commande `forward 10` a pour effet de faire effectuer un pas de 10 unités spatiales à la tortue. Ainsi, un programme Logo consiste généralement dans un ensemble de procédures très simples qui définissent les comportements primitifs de la tortue. Ces comportements sont ensuite composés pour obtenir un comportement plus complexe. Avec StarLogo, l'idée de Resnick était de fournir un environnement de simulation composé de plusieurs tortues évoluant simultanément dans un environnement commun, et ainsi de simuler un système multi-agents. De par la simplicité du modèle proposé, StarLogo a fait de nombreux émules, notamment la plate-forme NetLogo[24] et la plate-forme TurtleKit que nous avons développée et dont nous parlerons brièvement dans la section 5.3.5 (page 106).

Intelligence artificielle distribuée

En tant que domaine émergeant de l'intelligence artificielle distribuée (IAD), les systèmes multi-agents en ont tout de suite hérité la motivation : résoudre de manière répartie un problème complexe, que celui-ci soit naturellement distribué ou non (décomposition/distribution d'un problème, coordination/coopération, résolution de conflits par la négociation, planification, etc.). Deux exemples historiques sont la distribution et l'allocation de tâches à l'aide du protocole de coordination *Contract net protocol* [Smith, 1980] et la résolution de problèmes distribués telle qu'elle a été proposée dans le projet DVMT [Lesser & Corkill, 1983] pour le repérage de véhicules à partir d'un réseau de capteurs[25]. Il faut noter par ailleurs que DVMT a inspiré de très nombreux travaux de recherche dans divers domaines de l'intelligence artificielle distribuée durant toutes les années 80 (cf. [Decker, 1996]). Ce projet constitue assurément une étape très importante de l'histoire des systèmes distribués.

Pour implémenter, tester et analyser les diverses idées imaginées par cette communauté, il était bien sûr nécessaire de développer des plates-formes permettant de les expérimenter. La plupart des *testbeds* ainsi créés dans les années 80 n'étaient cependant pas génériques car développées autour d'une architecture de résolution prédéfinie ou pour un domaine particulier. L'équipe de Gasser rompit avec cette coutume en proposant une plate-forme multi-agents qui ferait date : MACE [Gasser *et al.* , 1987]. L'objectif de cette plate-forme était de fournir un environnement de développement, de simulation et d'analyse de systèmes multi-agents qui ne fasse aucune supposition sur la nature des problèmes envisagés et sur l'architecture interne des agents utilisés. L'idée force de cette approche était de considérer qu'un système multi-agents peut-être vu comme une organisation sociale qui doit être rendue explicite. Un agent ou un ensemble d'agents (cluster) sont alors partiellement définis par le rôle fonctionnel qu'ils tiennent au sein de l'organisation. MACE définissait ainsi un ensemble de facilités logicielles permettant aux agents d'utiliser la structure organisationnelle pour résoudre des problèmes complexes. A cela étaient ajoutés de nombreux outils permettant d'espionner le fonctionnement des systèmes construits afin d'évaluer leur efficacité.

La problématique de l'évaluation qualitative de ces systèmes d'intelligence artificielle distribuée, c'est-à-dire de leur capacité à résoudre des problèmes avec une quantité de ressources

[22] el.media.mit.edu/logo-foundation : le langage Logo est un dialecte de Lisp élaboré à la fin des années 60. Très innovant, ce langage avait déjà pour but de fournir un outil pédagogique basé sur un principe radicalement nouveau pour l'apprentissage de l'informatique.

[23] A l'origine, il s'agissait d'un petit robot, doté d'une sorte de carapace métallique, qui pouvait être dirigé en utilisant des commandes simples envoyées depuis un ordinateur.

[24] ccl.northwestern.edu/netlogo

[25] dis.cs.umass.edu/research/dvmt.

finie (puissance de calcul, mémoire, etc.), fut ainsi la question centrale de nombreuses recherches. Dans [Pollack & Ringuette, 1990] par exemple, Pollack et Ringuette ont proposé un système de simulation appelé *Tileworld* spécialement conçu pour étudier la qualité, en termes de temps de résolution, de différentes stratégies de raisonnement dans un environnement dynamique[26]. En offrant la possibilité de fortement modifier les différents paramètres de l'environnement, l'objectif de cette expérience était d'observer l'influence de la structure environnementale sur l'efficacité de diverses politiques comportementales. Il est en effet clair qu'un modèle de raisonnement n'est véritablement efficient que s'il est capable de s'adapter correctement à différents modèles environnementaux. La question de la qualité des expériences de simulation fut alors posée [Hanks *et al.* , 1993] : était-il possible de généraliser les résultats observés sachant qu'ils avaient été obtenus sur des plates-formes spécifiques et dans des contextes particuliers ?

Les aspects de de modélisation et de simulation de ces expériences prennent alors une plus grande importance et un sens nouveau. Jusqu'ici, les plates-formes multi-agents proposaient des outils de simulation dans le sens où elles fournissaient un ensemble de routines logicielles permettant d'exécuter un système multi-agents. Il leur faut maintenant intégrer des modèles du monde plus avancés qui rendent plus finement compte de la réalité des systèmes d'intelligence artificielle que l'on souhaite étudier : ressources système disponibles finies (mémoire, processeur, réseau, etc.), prise en compte de l'imperfection des capteurs et des effecteurs des agents, complexité du monde, génération d'événements imprévus, modélisation du temps cohérente, etc. Il s'agit ici d'être capable de modéliser le plus fidèlement possible les caractéristiques de l'**environnement** dans lequel les agents vont être amenés à évoluer. En d'autres termes, on se place à partir de ce moment véritablement dans le contexte de la théorie de la M&S car il est en effet maintenant question des deux relations fondamentales que nous avons présentées au chapitre précédent : la relation de modélisation et la relation de simulation (cf. 2.8.3). En se concentrant sur la modélisation du domaine, le langage de simulation et de modélisation d'environnement *TAEMS* [Decker & Lesser, 1993] fut l'un des premiers à focaliser les efforts de modélisation sur l'environnement plutôt que sur l'architecture interne des agents. La plate-forme MASS[27] en est la descendante directe [Vincent *et al.* , 2001]. TAEMS inspire encore actuellement de nombreuses expériences de simulations ([Wagner *et al.* , 2003] par exemple).

Pour conclure sur ce domaine, on trouve dans [Decker, 1996] une liste très complète des plates-formes qui ont été implémentées avant cette date. Pour citer quelques exemples plus récents, on peut noter l'existence des plates-formes suivantes : JAMES[28] [Uhrmacher & Schattenberg, 1998] sur laquelle a récemment été implémenté le problème du Tileworld [Uhrmacher, 2001a], *Sensible agents*[29] [Barber *et al.* , 2001] et la troisième version de la plate-forme MACE réactualisée sous le nom de MACE3J[30] [Gasser & Kakugawa, 2002].

3.3.3 Exemple d'approche générique : la plate-forme de simulation SWARM

Initialement développée au *Santa Fe Institute*, SWARM[31] est certainement actuellement la plate-forme de simulation multi-agents la plus connue et la plus utilisée. Ce succès est

[26] L'environnement étant modélisé par une grille à deux dimensions comportant des obstacles, un robot est chargé de pousser dans des trous des objets aléatoirement répartis. Les obstacles peuvent par ailleurs apparaître et disparaître dynamiquement au cours de la simulation.

[27] dis.cs.umass.edu/research/mass.

[28] www.informatik.uni-ulm.de/ki/james.html.

[29] www-lips.ece.utexas.edu/agents.html.

[30] www.isrl.uiuc.edu/amag/mace.

[31] www.swarm.org.

en grande partie lié au fait qu'elle ne vise pas un domaine d'application en particulier. Au contraire, elle a été conçue de manière à fournir des outils logiciels suffisamment génériques pour qu'ils puissent permettre d'implémenter toutes sortes de projets liés à l'utilisation du paradigme agent dans la simulation. Pour ce faire, cette plate-forme repose sur un principe de simulation simple et sur un ensemble de bibliothèques et de facilités logicielles orientées objet qui sont destinés à simplifier l'implémentation et l'exploration d'un modèle.

Le composant fondamental du principe de simulation proposé par cette plate-forme est un objet appelé un *swarm*. Un swarm est un ensemble d'agents auquel est associée une liste d'événements (des *activités*) définie a priori lors de la conception du modèle. Par exemple un swarm peut être, comme sur la figure 3.6, un ensemble de proies et de prédateurs associé à une suite d'activités prédéfinies. Dans cet exemple, les prédateurs mangent les proies qui tentent ensuite de fuir ces mêmes prédateurs [Minar *et al.* , 1996]. Un swarm peut à son tour être considéré comme un événement particulier qui pourra être utilisé dans une liste d'activités correspondant à un autre swarm. Ce qui permet de définir des modèles complexes basés sur des structures hiérarchiques.

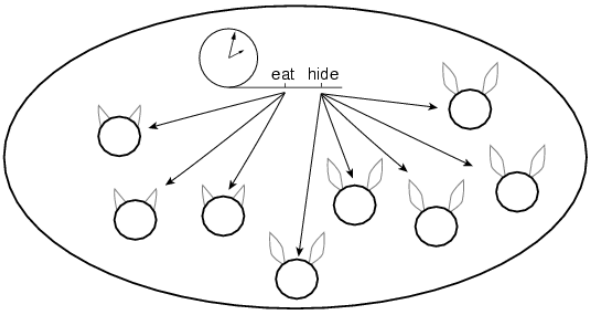

FIG. 3.6 – Un swarm : une liste d'activité sur un ensemble d'agents [Minar *et al.* , 1996].

A ce mécanisme d'exécution s'ajoute un ensemble d'outils dédiés à l'observation des différentes composantes du modèle. Il est ainsi possible de placer des sondes (*probes*) sur différentes propriétés de manière à extraire les données d'observation souhaitées. Sur la base de ces deux principes d'implémentation, la force de cette plate-forme est de proposer trois grands ensembles de librairies logicielles qui permettent de mettre en place les différentes parties du simulateur : une librairie *simulation* qui implémente l'ensemble des concepts de base de la plate-forme (swarm, activité, outils d'observation, etc.), une librairie *support* qui encapsule les tâches de fonctionnement du système (générateurs aléatoires, gestion des objets, etc.) et une librairie *modèle* qui propose tout un ensemble de modèles prédéfinis (environnement, agents, swarms, etc.) qui peuvent être réutilisés ou spécialisés suivant le domaine d'application visé. Pour résumer la philosophie de cette plate-forme, on peut dire qu'elle consiste dans un ensemble d'outils dédiés à supporter la **conception de simulateurs** multi-agents, au contraire de la majorité des plates-formes qui proposent des outils conçus pour implémenter un type de simulation particulier au regard d'un domaine d'application. Nous aurons l'occasion de revenir sur l'intérêt de cette démarche car cette plate-forme a été pour nous une grande source d'inspiration et les outils de conception que nous présenterons dans le chapitre 5 ont été élaborés dans le même esprit.

Nous allons maintenant nous intéresser aux problématiques qui sont liées à l'élaboration et à l'implémentation d'un modèle de simulation multi-agents. Comme nous allons le voir, la modélisation et l'implémentation de ces systèmes complexes soulèvent de nombreuses questions et posent encore aujourd'hui un certain nombre de difficultés non triviales. Nous allons

commencer par étudier les problèmes liés à l'implémentation des agents et de l'environnement et nous nous intéresserons ensuite à la manière dont la modélisation du temps est considérée dans les simulations multi-agents.

3.4 Problématiques liées à l'implémentation des agents et de l'environnement

3.4.1 Difficulté d'avoir un point de vue générique

D'un point de vue technique, les caractéristiques d'un simulateur multi-agents sont bien sûr en grande partie liées à l'architecture logicielle utilisée pour implémenter les agents et l'environnement. Dans ce cadre, les besoins et les contraintes qui portent sur celle-ci peuvent varier très fortement du fait de l'hétérogénéité des domaines où la simulation multi-agents peut-être appliquée. Les structures des modèles représentants les agents et l'environnement ne sont en effet pas les mêmes suivant le cadre expérimental dans lequel on se place (RMC, écologie, éthologie, etc.). L'architecture d'un simulateur multi-agents est donc bien sûr très fortement influencée par la nature du système source que l'on souhaite modéliser. C'est d'ailleurs sans aucun doute l'une des raisons pour lesquelles il n'existe pas à l'heure actuelle de méthodologie générique liée à la modélisation et à l'implémentation des systèmes multi-agents qui fasse l'objet d'un consensus. En conséquence, la nature de l'environnement (continu ou discret, statique ou dynamique, etc.), la granularité des actions/perceptions (fine ou grossière) et la complexité de l'architecture interne d'un agent sont autant de points sur lesquels le modélisateur est amené à faire des choix personnels qui sont fonction de ses besoins. Le nombre impressionnant de plates-formes existantes en est la manifestation.

3.4.2 Problématiques invariantes

Malgré cela, pour répondre aux objectifs que nous nous sommes fixés dans cette thèse, il nous faut ici essayer d'exhiber quelques invariants liés à l'implémentation du couple agents/environnement. Et ce, afin de clairement identifier les caractéristiques qui sont spécifiques aux simulations multi-agents. L'approche multi-agents impose en effet des contraintes, plus ou moins fortes, que toute implémentation, et a fortiori les propositions de cette thèse, doivent prendre en considération.

Nécessité d'une architecture modulaire

Tout d'abord, il est préférable que le design de l'architecture logicielle envisagée soit la plus modulaire possible. Les raisons en sont à la fois pratiques et conceptuelles. Pratiques dans le sens où il est nécessaire d'avoir un minimum de généricité afin de pouvoir implémenter facilement des variations sur les agents et/ou l'environnement sans remettre en cause l'ensemble du code source. Conceptuelles car le paradigme agent appelle naturellement une programmation modulaire car il est question de modéliser des entités individuelles et autonomes. De plus, ces deux aspects sont bien sûrs fortement liés : plus une implémentation est modulaire, plus elle a de chances d'implémenter des agents et un environnement conceptuellement corrects et vice versa. La plupart des plates-formes de simulation multi-agents intègrent ces aspects et proposent des architectures agent modulaires et/ou des environnements componentiels ([Lhuillier, 1998] par exemple). Pour ce faire, le modèle objet est aujourd'hui utilisé dans

la majorité des cas. Les agents et l'environnement sont alors représentés par une collection d'objets dont les interfaces accessibles définissent les moyens grâce auxquels ils interagiront[32].

Au-delà des contraintes liées à des considérations purement génie logiciel comme celle que nous venons d'évoquer, il est encore plus intéressant d'exhiber les contraintes génériques sous-tendues par le concept d'agent lui-même. Nous pensons qu'il existe deux contraintes fondamentales qui pèsent sur l'implémentation des agents et de l'environnement :

- la *contrainte de localité* : un agent est une entité dont les perceptions et les actions n'ont qu'une portée locale.
- la *contrainte d'intégrité environnementale* : un agent ne doit pas être en mesure de modifier directement les variables d'état de l'environnement.

Contrainte de localité

Cette première contrainte est incontournable. Comme nous l'avons dit précédemment, l'environnement n'est pas entièrement accessible pour un agent. Ce dernier n'a en principe qu'une perception locale du monde dans lequel il se trouve et ses actions sont limitées dans leur portée. Pour prendre en compte cet aspect, l'implémentation doit nécessairement fournir un moyen de définir le rayon d'action/perception d'un agent. Pour cela, deux solutions non exclusives sont possibles :

1. approche discrète (centrée environnement) ; où l'implémentation de l'environnement définit la granularité des perceptions/actions de par la nature du modèle utilisé.
2. approche continue (centrée agent) ; où la portée de chaque perception/action fait l'objet d'un traitement particulier fonction de sa nature et des caractéristiques de l'agent concerné.

Approche discrète. Lorsque l'environnement peut être facilement discrétisé en une juxtaposition de zones (les pièces d'une maison, les nœuds d'un réseau, le découpage d'une zone géographique, etc.), la première solution consiste à utiliser les caractéristiques du modèle environnemental pour définir la portée des perceptions/actions de manière générique. De la même façon que pour un environnement représenté par un objet unique, chaque zone définit une interface qui décrit les actions qui peuvent lui être appliquées. La dimension d'une zone est alors utilisée comme unité spatiale pour définir la portée des perceptions/actions. De plus, une relation de voisinage entre les différentes zones est utilisée pour définir la topologie de l'environnement. Dans le cas d'une maison cette relation peut être concrétisée par les portes qui existent entre les pièces par exemple.

L'exemple le plus courant d'une telle implémentation consiste dans une grille de cellules (ou patch) régulière (figure 3.7). Pour ce type environnement, les deux relations de voisinage les plus utilisées sont les suivantes :

- le voisinage de *Von Neumann* où une cellule est en relation avec les cellules adjacentes qui se trouvent aux quatre points cardinaux : Nord, Sud, Est et Ouest.

[32] Quelle que soit la nature de l'implémentation, pour que les agents puissent respectivement percevoir et agir sur leur environnement, celle-ci doit fournir des mécanismes qui leur permettent de consulter et de modifier, non nécessairement de manière directe, l'état de l'environnement. L'interface de l'environnement définit ainsi les actions et les perceptions que l'agent peut invoquer grâce à l'attribution d'une référence, que celle-ci soit encapsulée ou non, directe ou indirecte comme c'est le cas lorsqu'un objet intermédiaire tel qu'un effecteur est utilisé par exemple. De façon duale, l'interface de l'agent est utilisée par l'environnement pour concrétiser de potentielles modifications sur son état physique.

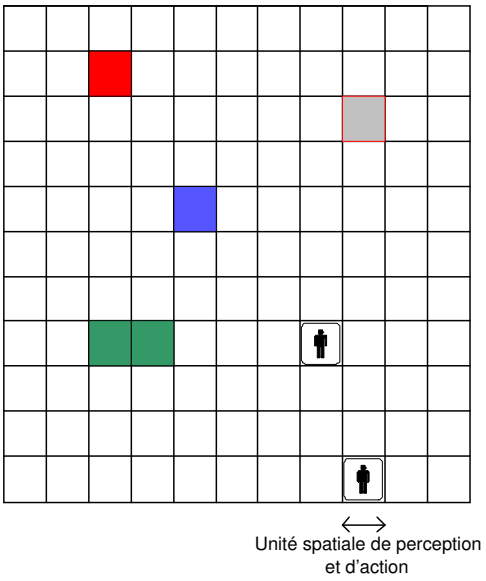

FIG. 3.7 – L'environnement comme une grille de cellules.

– le voisinage de *Moore* où les voisins d'une cellule sont les huit cases adjacentes.

Largement utilisée pour leur simplicité d'implémentation, les méthodes à base de grille ont par ailleurs le bon goût de permettre la définition de dynamiques environnementales basées sur des algorithmes hérités du domaine des automates cellulaires. Cependant leurs utilisations posent le problème de la granularité spatiale des perceptions/actions qu'un agent peut effectuer du fait de la régularité de leur discrétisation : quelles que soient les actions d'un agent, et on sait qu'elles peuvent être très hétérogènes, leur portée est toujours la même (modulo un certain facteur) : la cellule.

Approche continue. Au contraire, la seconde solution considère chaque agent comme le référentiel par rapport auquel la portée effective de chaque perception/action doit être calculée. Il s'agit ici d'augmenter la finesse de la modélisation du système en individualisant ce traitement en fonction du type de la perception/action et des caractéristiques de l'agent. On doit par exemple être capable d'implémenter le fait que les capteurs visuels d'un robot n'ont pas forcément une portée qui soit un multiple de celle de ses capteurs d'ondes radio. On doit aussi pouvoir prendre en compte l'état de fonctionnement dans lequel se trouvent les capteurs pour définir leur portée : on peut par exemple désirer l'altérer volontairement en la multipliant par un nombre réel. Cette implémentation de la contrainte de localité est bien sûr une solution plus complexe à mettre en œuvre que la précédente et son application s'adresse plus particulièrement à la modélisation de systèmes où l'environnement est non déterministe et/ou continu. La copie d'écran présentée par la figure 3.8 illustre cette approche. Dans cette simulation chaque agent possède un rayon de perception qui lui est propre, ici représenté par un cercle dont l'agent est le centre.

Le principal inconvénient de ce type d'implémentation réside dans le fait que le code source associé peut rapidement devenir impossible à développer et/ou à réutiliser si cette approche est mise en œuvre de manière ad hoc. A ce propos, l'étude réalisée dans sa thèse par Magnin sur les relations entre un agent et son environnement montre que de nombreux inconvénients, rencontrés aussi bien lors de l'analyse que de l'implémentation des systèmes multi-agents, sont en fait dus à un manque de différenciation entre la partie physique d'un agent et son système

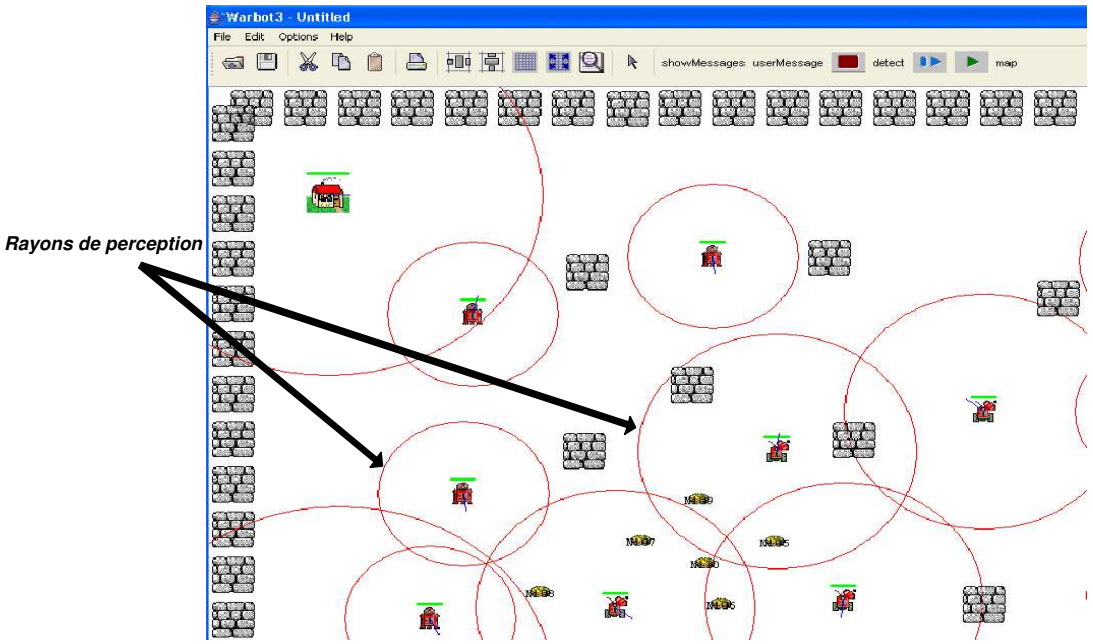

FIG. 3.8 – Approche continue pour la perception et l'action.

décisionnel [Magnin, 1996]. Magnin parle de *confusion des rôles* entre la composante physique et le *cerveau* d'un agent. Son travail montre les problèmes liés à une telle confusion ainsi que l'importance de faire une telle distinction d'un point de vue génie logiciel.

On retrouve aussi cette idée de séparation explicite entre différentes composantes d'un agent dans les travaux de Lhuillier [Lhuillier, 1998] et de Soulié [Soulié, 2001]. Il est en effet clairement beaucoup plus simple de modifier ou de substituer les différents éléments d'une simulation (agents et/ou environnement) si l'action, la perception, le comportement interne d'un agent et l'environnement sont modélisés par des entités distinctes. En effet, lorsqu'un agent utilise, pour percevoir ou agir, des méthodes définies dans la classe qui représente l'environnement, il n'est pas possible de remplacer le modèle de l'environnement par un autre, plus complexe, sans avoir à modifier l'ensemble des classes concernées, c'est-à-dire celles de l'agent et de l'environnement. Outre cet indéniable aspect pratique, cette distinction est aussi d'un intérêt conceptuel fondamental qui nous amène à discuter la deuxième contrainte que nous avons proposée : la contrainte d'intégrité.

Contrainte d'intégrité environnementale

L'énoncé de cette contrainte stipule qu'un agent ne doit pas modifier directement les variables d'état de l'environnement. En effet, ce n'est pas une entité en mesure de calculer la conséquence réelle de ces décisions sur l'évolution du monde. Comme le souligne Ferber, le résultat de l'action d'un agent est aussi fonction des autres actions et de l'évolution endogène de l'environnement [Ferber, 1999]. Ainsi, il est conceptuellement incorrect qu'un agent puisse modifier une variable modélisant l'état de l'environnement. Très peu de travaux intègrent cette contrainte et l'action d'un agent est généralement modélisée par son résultat pour des raisons de simplicité. Autrement dit, dans la majorité des cas, les agents modifient les variables environnementales de leur propre chef pour signifier leurs actions.

Pour compliquer les choses, les notions d'agent et d'environnement sont à ce point inséparables qu'une partie de l'entité conceptuelle désignée par le terme agent est contenue dans l'environnement. Pour illustrer notre propos, prenons l'exemple de la simulation d'un robot réel. Lorsque nous employons le terme agent pour le désigner, nous faisons référence au robot dans son ensemble : le programme informatique qui l'habite et ses moyens d'action physiques (bras articulé, système de déplacement, etc.). Dans la réalité, ce n'est pas parce que le système décisionnel du robot prend l'initiative d'une action que le résultat escompté va forcément se réaliser[33] : ses moyens physiques peuvent ne pas être opérationnels par exemple (mécanique cassée, manque d'énergie, etc.). Or ce qui est appelé agent dans la figure 3.1 est une boîte noire dans laquelle toutes les variables d'état sont par définition accessibles aux processus qui modélisent la dynamique de l'agent. Dans le cadre d'une simulation, les variables qui représentent la partie physique d'un robot ne doivent donc pas se trouver dans cette boîte noire, puisqu'elles ne sont pas sous le contrôle de l'agent. Par conséquent, ces variables font partie intégrante de l'environnement. C'est pourquoi, pour simuler un agent, il convient de faire une distinction nette entre les variables d'état qui sont utilisées par son architecture interne et les variables qui modélisent sa partie physique.

Ceci étant dit, il nous faut ici revenir quelques instants sur la représentation schématique d'un système multi-agents que nous avons donnée dans la section 3.2.1. Comme nous l'avons dit précédemment, du point de vue d'un agent, tout ce qui ne concerne pas son architecture interne fait de facto partie de l'environnement. Par ailleurs, nous venons de voir l'importance de l'environnement dans la réalisation d'une action. C'est pourquoi, lorsque l'on considère l'implémentation d'un système comportant plusieurs agents, il est conceptuellement tout à fait incorrect de représenter le fait que deux agents puissent interagir ensemble en concrétisant un lien direct entre leurs deux architectures internes, sans passage par l'environnement (figure 3.9). La figure 3.9 illustre cette contrainte.

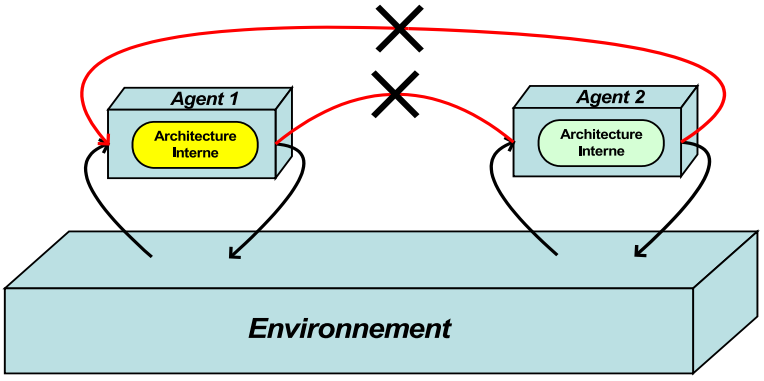

FIG. 3.9 – La contrainte d'intégrité environnementale implique qu'il ne doit pas exister de relation directe entre les architectures internes de deux agents autonomes.

La séparation explicite entre les différentes composantes d'un agent trouve ici une justification supplémentaire. En effet, si cette distinction n'est pas effective dans l'implémentation, il est difficile de garantir qu'un agent n'a pas un accès direct sur l'architecture interne d'un autre, comme c'est en fait souvent le cas. Nous mettrons en évidence ce problème dans le chapitre 7. Pour l'instant, on peut remarquer que concrétiser cette distinction dans l'implémentation

[33] On trouve dans [Brooks, 1992] une très intéressante analyse des problèmes qu'il faut prendre en compte lorsqu'on s'attaque à la simulation de robots réels : imperfection des valeurs fournies par les senseurs, incertitude sur le résultat des commandes appliquées aux effecteurs, etc.

n'est pas du tout trivial et peut rapidement devenir un casse-tête si l'on n'y prend pas garde. En effet, comme nous venons de le voir dans le cas d'un robot, la frontière entre un agent et son environnement est assez subtile. Elle dépend d'ailleurs aussi de la nature de l'agent considéré : les capteurs et les effecteurs sont bien sûrs très différents suivant les différentes catégories d'entités qui peuvent être rencontrées. De plus, du fait de la relation forte qui existe entre ces deux entités, le design de l'un influence le design de l'autre. Ce qui constitue à n'en pas douter le principal frein à l'élaboration de codes réutilisables. Utiliser le code représentant le comportement d'un agent sur un autre environnement que celui pour lequel il a tout d'abord été conçu nécessite encore aujourd'hui de lourdes modifications.

Pour éviter de faire la confusion des rôles dont parle Magnin, une très bonne solution consiste à considérer les moyens d'action et de perception d'un agent comme des modules individuels distingués de son système conatif. C'est par exemple ce que propose Soulié qui réifie dans son travail les notions de capteurs et d'effecteurs [Soulié, 2001]. Il fait ainsi une distinction claire entre les modules qui représentent le système conatif et ceux qui modélisent la partie physique de l'agent, notamment ses moyens de perception et d'action, qui eux font partie de l'environnement. Il parle ainsi de *l'instance de l'agent dans l'environnement*.

Finalement, outre l'implémentation de l'architecture interne des agents d'un côté et celle de l'environnement de l'autre, la véritable difficulté réside en fait dans la manière dont le lien et la frontière qui existent entre ces deux abstractions sont concrétisés. Autrement dit, le point le plus délicat se situe dans ce que Soulié a très justement appelé le *lien de dépendance bidirectionnelle* entre un agent et son environnement. Son travail montre clairement que l'étude de ce lien constitue un point clé pour l'élaboration de structures logicielles génériques.

Pour résumer, il est important de faire la distinction entre les deux composantes d'un agent dès le modèle conceptuel de manière à respecter la contrainte d'intégrité environnementale dans l'implémentation. Nous aurons l'occasion de revenir sur ce point important à de nombreuses reprises dans ce document. Il constitue en effet un des piliers de notre analyse et nous montrerons comment cette idée peut être généralisée. Nous verrons alors tous les bénéfices que l'on peut tirer d'une distinction claire entre les deux composantes d'un agent : le *corps* et l'*esprit*[34].

3.5 La gestion du temps

3.5.1 Nécessité de gérer l'évolution du temps

Rappelons ici que le principe de la simulation multi-agents est basé sur l'idée qu'il est possible de représenter de manière informatique un ensemble d'entités autonomes évoluant dans un environnement commun. Du fait de cette appartenance à un seul et même environnement, tous les agents doivent être soumis à une même loi temporelle de manière à respecter le principe de causalité [Fyanio *et al.* , 1998]. Les agents évoluent, agissent et interagissent "tous en même temps" suivant des conditions environnementales qui sont partagées par tous.

Implémenter de façon satisfaisante une telle dynamique nécessite de gérer l'évolution de l'ensemble des entités suivant une même échelle temporelle. De façon intuitive, utiliser une architecture multiprocesseurs (un agent par machine par exemple) semble être une solution au problème de la gestion du temps. Mais sans synchronisation les agents évoluent au rythme

[34] Nous employons ici le terme *esprit* en lieu et place de cerveau car il a l'avantage de n'avoir aucune connotation physique. Merci à Guillaume Defuant pour nous avoir indirectement soufflé cette remarque.

de la complexité de leur architecture interne, ce qui ne permet pas de contrôler la cohérence de la dynamique globale du système [Resnick, 1990, Lhuillier, 1998]. Ainsi il est indispensable de gérer l'exécution des agents de façon à respecter un principe de causalité dans le processus de simulation des interactions.

Comme nous allons le voir, toute la difficulté est de faire en sorte que l'implémentation de cette gestion soit le plus près possible de ce que l'on souhaite modéliser, c'est-à-dire un système composé d'entités autonomes évoluant de concert dans un environnement commun.

3.5.2 Le comportement d'un agent : un processus discret

Comme nous l'avons dit dans le chapitre précédent, les modèles utilisés pour représenter le comportement d'un agent sont par nature discrets. En effet, bien qu'un agent puisse être plongé dans un environnement possédant une dynamique modélisée de manière continue, son processus décisionnel est généralement considéré comme un processus événementiel faisant évoluer ses variables d'état de manière discrète.

Ainsi, les différents paramètres qui façonnent l'architecture interne d'un agent et modélisent son comportement (désirs, croyances, représentation du monde, etc.) changent de façon instantanée. Dans l'état actuel des connaissances scientifiques, la délibération d'un agent est effectivement difficilement modélisable sous une autre forme étant donné sa complexité intrinsèque dans le monde réel. Par ailleurs, la perception, la délibération et l'action, sont dans la très grande majorité des cas associées à un unique et même instant t pour des raisons de simplicité.

Parfois le cycle d'un agent n'est pas modélisé par un événement ponctuel. Le temps de la perception et le temps de l'action sont alors distingués. Sans pour autant modéliser le comportement d'un agent par un processus continu, certaines approches nécessitent en effet de donner une épaisseur temporelle au cycle de l'agent. Par exemple, dans le domaine de l'intelligence artificielle distribuée, l'évaluation des ressources computationnelles (temps machine, mémoire, réseau, etc.) consommées par un agent pour résoudre un problème constitue le principal objet d'étude [Uhrmacher & Schattenberg, 1998, Vincent *et al.* , 2001]. A ce titre, la durée de délibération d'un agent doit être modélisée afin de pouvoir la mesurer et la comparer. Par exemple, Cohen la considère comme une fonction du temps de calcul effectif [Cohen *et al.* , 1989].

En fait, on se trouve ici dans un cadre expérimental particulier où une partie de la structure du système réel est parfaitement connue : il s'agit des agents informatiques eux-mêmes. Par conséquent, mesurer concrètement ce que leur coûtent leurs réflexions en temps machine a ici un véritable sens du fait de l'équivalence parfaite qui existe entre le modèle du processus décisionnel et son implémentation : les algorithmes **sont** exactement le processus décisionnel.

Ce qui a ici un sens n'en a plus dès qu'on s'attaque à des modèles comportementaux qui modélisent des êtres vivants ou artificiels : le fonctionnement du cerveau animal n'a par exemple assurément rien à voir avec l'application des structures algorithmiques que nous connaissons aujourd'hui. C'est pourquoi, dans ce type de modèle, le cycle d'un agent est généralement considéré comme un processus instantané.

Les modèles multi-agents utilisent donc les deux grands principes d'implémentation qui permettent de manipuler des événements discrets : la discrétisation régulière du temps et le principe de simulation par événements. Dans le domaine des systèmes multi-agents, on parle respectivement de l'approche *synchrone* et de l'approche *asynchrone* pour désigner de manière informelle ces deux principes de simulation.

3.5.3 Approche synchrone : simulations à *pas de temps constant*

Première approche

La technique la plus simple permettant d'implémenter une gestion du temps discrète consiste à activer, pour un instant t, le cycle de tous les agents (et éventuellement les objets de la simulation) de façon séquentielle.

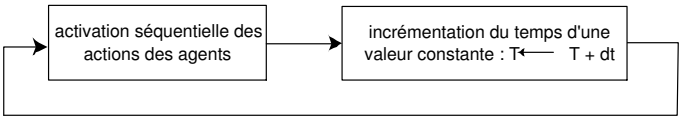

FIG. 3.10 – Principe de simulation à *pas de temps constant*.

Ce principe de fonctionnement est très utilisé du fait de sa simplicité d'implémentation puisqu'il suffit de programmer une fonction très simple qui active l'ensemble des agents. C'est par exemple le rôle joué par la fonction appelée *run-animas* dans la plate-forme LIVEWORLD développée par Travers [Travers, 1996] :

```
(defun run-animas ()
       (dolist (a *running-animas*)
       (send a :step)))
```

Le gros problème de ce type de fonctionnement tient dans le fait que l'ordre d'exécution des agents a un impact direct sur l'évolution du monde. En voici un exemple. La figure 3.11 décrit un modèle de type *proies/prédateurs*. On postule qu'une proie (un triangle) est capturée lorsqu'elle est entourée de quatre prédateurs (les ronds) suivant le voisinage de Von Neumann. Dans cet exemple, nous supposons par ailleurs que les agents ne peuvent se déplacer que dans les quatre directions cardinales. On voit sur cette figure que l'ordre d'exécution des agents (ici représenté par un numéro) peut changer l'issue d'une même situation. Dans cet exemple, la vie de la proie dépend de son indice d'activation. Ainsi à partir d'un même état du monde à un instant t, on peut obtenir avec **les mêmes comportements** des états du monde différents à $t + dt$ suivant l'ordre dans lequel sont activés les agents. Ce qui peut naturellement conduire à des biais de simulation importants.

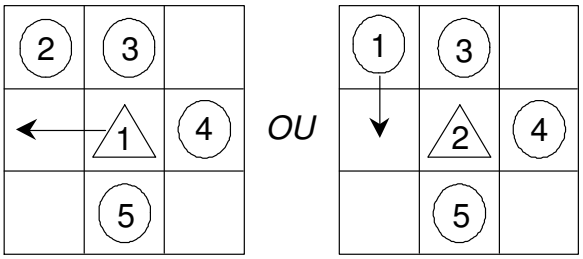

FIG. 3.11 – La survie de la proie dépend de sa place dans la liste d'exécution.

Pour remédier à ce problème, la première extension du modèle de simulation précédent consiste à brasser la liste d'exécution aléatoirement pour chaque pas de temps à fin d'éviter qu'un agent (le premier de la liste) ne soit systématiquement "avantagé ". C'est par exemple la méthode utilisée dans [Epstein & Axtell, 1996]. Ceci nous permet d'ouvrir ici une petite parenthèse pour souligner l'importance de la qualité des générateurs de nombres aléatoires dans

la simulation en général [L'Ecuyer, 1990]. A ce propos, [Hill, 2003] constitue un point d'entrée récent sur cette problématique et propose une technique qui permet d'optimiser l'utilisation des générateurs de nombres aléatoires grâce à un pré-calcul.

Si la technique précédente permet dans un sens de mettre, au moins sur le plan statistique, tous les agents sur un pied d'égalité, il subsiste certains cas qui restent problématiques. Notamment en ce qui concerne la cohérence des différentes perceptions que les agents ont du monde : le monde perçu par différents agents à un même instant t de la simulation n'est pas identique. En effet, un agent perçoit à t un monde qui a été modifié par les agents qui ont agi avant lui. En d'autres termes, la perception d'un agent peut potentiellement s'appuyer sur des variables qui devraient être estampillées à $t + dt$. Et ce, alors que les agents sont supposés agir de manière concurrente à cet instant. Ainsi le dernier agent peut avoir une vue du monde totalement différente du premier alors que l'état de celui-ci est censé toujours être donné pour l'instant t. L'action d'un agent, qui est en principe déduite de sa perception du monde, est en fait directement lié à sa position dans la liste d'activation et par conséquent aux actions effectuées auparavant par les autres agents. Prenons ici un exemple très simple. Considérons un environnement modélisé par une unique variable d'état binaire ayant 0 pour valeur initiale. Dans cet environnement, nous plaçons n agents possédant un comportement simple qui consiste à modifier la variable environnementale dans son contraire. Avec ce modèle très simple, on peut remarquer que suivant la technique d'implémentation précédente, chaque agent décide son action sur la base d'une perception du monde à l'instant t qui dépend de l'action de son prédécesseur et qui est différente d'un agent au suivant. Par ailleurs, on peut remarquer que l'état final de l'environnement pour un pas de temps dépend de la parité du nombre d'agents. Il est donc fortement souhaitable que tous les agents aient la même perception du monde pour un même instant t si l'on souhaite que leurs actions puissent être considérées comme parallèles [Campos, 2000]. Cela afin que le processus de délibération d'un agent ne soit pas biaisé par des comportements produits au même instant t.

Utilisation d'un état tampon et résolution de conflits

Travers a bien identifié le problème précédent et propose de simuler la concurrence en utilisant un état tampon pour les variables d'état environnementales. Inspirée par la dynamique des systèmes de type Moore (cf. section 2.5.2), l'idée est de reporter la validation des actions sur l'environnement à la fin d'un cycle de façon à ce que tous les agents perçoivent le même état du monde à un instant t. Pour cela les actions des agents se font sur des variables d'état tampons et le monde n'est pas directement modifié. La fonction *run-animas* est maintenant définie de la manière suivante. Dans une première phase, les modifications sont stockées dans des variables tampons (new-value) et sont ensuite validées lors d'une deuxième phase qui prend place une fois que tous les agents ont agi :

```
(defvar *phase* 0)
(defvar *phase2-boxes*)

(def-lw-method :set #/global (new-value)
        (case *phase*
                (1 (progn
                        ;; use lower-level mechanism to save new value in annotation
                        (set-slot-1 (getb-force self :new-value) new-value)
                        (pushnew self *phase2-boxes*)
                        new-value)) ; return the new value
                ((0 2) (set-slot-1 self new-value))))

(defun run-animas ()
        (let ((*phase* 1)
```

```
        (*phase2-boxes* nil))
(dolist (a *running-animas*)
        (send a :step))
(setf *phase* 2)
(dolist (f *phase2-boxes*)
        (send f :set (slot f :new-value)))))
```

L'utilisation de cette méthode entraîne cependant directement un nouveau problème. Un conflit apparaît lorsque deux agents ou plus spécifient de nouvelles valeurs différentes pour une même variable. Il est donc nécessaire de résoudre ce conflit pour avoir un état du monde cohérent. La figure 3.12 illustre ce principe de simulation :

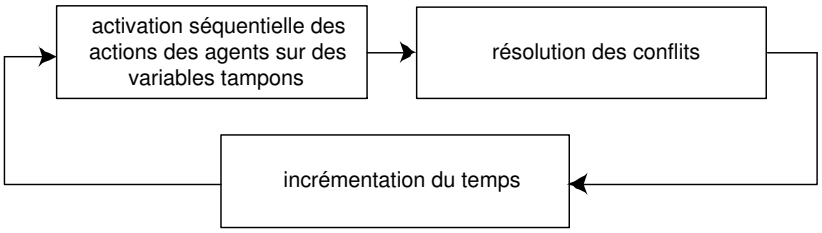

FIG. 3.12 – Simulation avec état tampon et résolution de conflits.

D'une manière générale, ce genre de méthode peut être assimilé aux techniques inspirées par une *approche à trois phases* comme celle utilisée dans [Dumont & Hill, 2001] par exemple. Cette approche consiste à classifier les différentes activités suivant leurs interdépendances. La première phase consiste dans la gestion du temps, la seconde exécute les activités dont la réalisation n'est pas conditionnée et la troisième traite les actions qui sont en conflit ou interdépendantes [Balci, 1988]. La méthode la plus utilisée pour résoudre un conflit consiste à choisir aléatoirement l'agent dont l'action sera finalement validée. Ainsi, bien que les agents aient maintenant la même perception du monde, on peut retomber sur des cas problématiques comme ceux que nous avons présentés précédemment.

Cette technique a cependant un avantage qui est indéniable : la détection des conflits étant simple, il est donc possible de les résoudre d'une façon **systématique** et **contrôlée**, et donc de les inclure explicitement dans le modèle de simulation[35]. A partir de là, il est alors possible d'élaborer des méthodes de résolution de conflits plus fines ayant un véritable sens pour le modèle. Une possibilité consiste par exemple à classer les différentes actions suivant un niveau de priorité qui détermine l'ordre dans lequel celles-ci doivent être exécutées [Campos, 2000]. A ce propos, Travers propose une technique de résolution des conflits assez complexe qui utilise une sorte de journal des actions (qui a fait l'action) lui permettant de déterminer parmi les valeurs proposées celles qui conviennent le mieux étant donné un raisonnement portant sur l'ensemble des priorités associées aux agents.

Hormis les problèmes directement liés à la complexité de résoudre certains conflits, ces techniques ont par ailleurs une autre limite qui se situe cette fois à un niveau conceptuel. Il est très intéressant de remarquer que la résolution d'un conflit implique que l'action initiale d'un agent peut être invalidée. En toute logique, certains agents proposent donc des valeurs qui correspondent à des actions qui ne sont pas valides, ce qui remet en cause leur processus décisionnel. La résolution d'un conflit est donc en quelque sorte effectuée à partir de données erronées : les agents calculent un état du monde qui ne prend pas en compte les autres actions qui sont effectuées au même instant. Or il est clair que le résultat d'une action n'est pas

[35] Travers montre par ailleurs que la présentation du conflit à l'utilisateur est aussi une possibilité intéressante.

uniquement lié à l'entité agissante [Ferber, 1999]. Nous reviendrons sur ce point un peu plus loin.

Le problème de la granularité temporelle des actions

De la même manière que la discrétisation de l'environnement peut poser un problème pour modéliser des actions dont les portées sont fortement différentes, la discrétisation régulière du temps soulève la question de la granularité temporelle des actions. La figure 3.13 montre un exemple de simulation où plusieurs agents souhaitent se déplacer dans un environnement discret. On voit sur cette figure que s'il est possible à une entité de se déplacer de plus d'une case dans un seul pas de temps, plusieurs agents peuvent par exemple avoir des trajectoires qui se croisent sans être à aucun instant de la simulation en collision, la vitesse des déplacements n'ayant pas de lien direct avec le temps. Et ce, quelque soit la technique utilisée (boucle simple ou résolution de conflits), les positions finales n'étant pas en conflit.

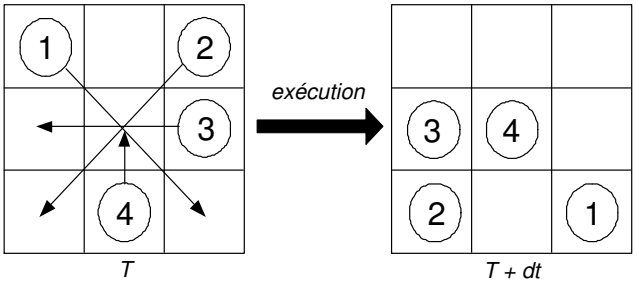

FIG. 3.13 – Le problème de la granularité des actions.

Ainsi, il est important qu'il existe une relation de cohérence entre la granularité temporelle des actions et l'unité du pas de temps lorsque le cadre expérimental l'implique. Ce qui est le cas dans les simulations qui modélisent un trafic automobile par exemple [Dresner & Stone, 2004]. Le cas précédent est bien sûr une caricature mais il est important de comprendre que, étant donnée l'hétérogénéité des modèles qui sont utilisés, de nombreux problèmes liés à cette question peuvent apparaître sans pour autant être aussi criants. On trouve un inventaire un peu plus exhaustif de ces problèmes dans [Magnin, 1996].

3.5.4 Simulations par événements

Une représentation du temps plus souple

Bien que les simulations à pas de temps constant soient très largement majoritaires, la gestion de l'ordonnancement des agents peut sembler contre nature dans la mesure où toutes les entités sont activées en même temps. En tant qu'observateur, il est en effet difficile de se représenter la réalité comme un ensemble de systèmes dont les entités seraient "mises à jour" de façon simultanée par une horloge globale [Huberman & Glance, 1993]. La nature nous montre au contraire des entités qui agissent à des instants non corrélés du fait de leur autonomie. On retrouve cette idée dans [Lawson & Park, 2000] où les auteurs expliquent qu'une activation asynchrone des agents paraît mieux à même de modéliser l'évolution d'une société artificielle. Ainsi, il est parfois insatisfaisant d'utiliser une discrétisation régulière du temps, notamment lorsque le modèle nécessite de prendre en compte des actions fortement hétérogènes du point

de vue de leur fréquence. Dans ce contexte, l'approche événementielle propose une plus grande souplesse dans la modélisation de l'écoulement du temps. On retrouve ainsi dans la simulation multi-agents les différentes techniques et implémentations utilisées pour gérer ce type de modèles (cf. section 2.6.3 page 35). On trouve dans [Guessoum, 1996] une analyse de ces différentes solutions dans le contexte des systèmes multi-agents.

La plate-forme DIMA[36] est un exemple de l'utilisation d'un modèle événementiel pour la simulation multi-agents [Guessoum, 2000]. Outre la gestion des événements, le principe de simulation associé à cette plate-forme repose aussi sur la notion d'*activité agent*, ce qui permet de donner une épaisseur temporelle aux tâches réalisées par les entités du système. Autre exemple, dans la plate-forme JAMES, Uhrmacher utilise le formalisme DEVS pour modéliser la dynamique du système : chaque agent est un composant atomique qui définit ainsi sa réaction aux événements extérieurs et la manière dont il génère des événements de façon proactive [Uhrmacher & Schattenberg, 1998]. Dans ce type de modèle, chaque agent évolue donc à son propre rythme et génère des événements dont la date de réalisation lui est personnelle. Ainsi, l'activation des agents ne se fait plus de façon synchrone mais asynchrone. De la même manière, l'évolution endogène du monde est modélisée par un générateur d'événements indépendant qui correspond à sa propre dynamique.

Dans certains cas, l'ordonnancement et la date des événements sont déterminés a priori et la simulation consiste à exécuter la *liste des événements*. Dans d'autres, il peut être nécessaire de déterminer les événements en cours de simulation de façon à prendre en compte les conséquences des événements précédents. Dans son simulateur SIEME, Magnin utilise par exemple un ensemble de règles environnementales qui lui servent à calculer la date des événements futurs. Le principe est de décrire les interactions possibles a priori dans le modèle et de déterminer, grâce à ces règles, l'ordre dans lequel elles interviennent au fur et à mesure de la simulation. Par exemple si on envisage un système d'objets en mouvement où les collisions sont les seuls événements considérés, il s'agit de déterminer le choc le plus proche dans le temps pour, à ce moment-là, calculer le résultat de l'interaction de collision et définir les nouvelles trajectoires des objets. Déterminer le futur événement de cette simulation consiste alors à calculer l'instant où aura lieu la prochaine collision étant données ces nouvelles trajectoires. La simulation évolue donc d'interaction en interaction. La figure 3.14 illustre ce principe :

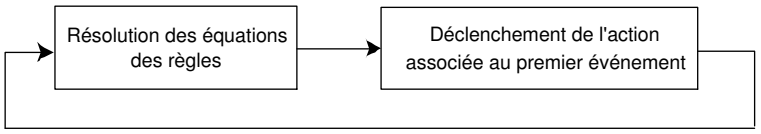

FIG. 3.14 – Principe de fonctionnement du simulateur SIEME [Magnin, 1996].

Encore des conflits

Bien que les agents agissent à des instants non corrélés, deux événements peuvent cependant intervenir au même instant de la simulation. Ce qui pose bien sûr le problème de la résolution d'un conflit dans les mêmes termes que précédemment. De plus, les simulations multi-agents utilisent fréquemment le principe événementiel de pair avec une discrétisation du temps régulière : dans ce cas les agents n'agissent pas tous à chaque pas de temps (par exemple tous les deux ou trois pas de temps) mais se retrouvent souvent en concurrence. Il

[36] www-poleia.lip6.fr/~guessoum/dima.html.

faut alors résoudre les conflits, le plus souvent en effectuant un choix aléatoire pour déterminer l'ordonnancement dans lequel les actions des agents sont validées.

3.5.5 Le problème de la simultanéité

La résolution de conflits semble ainsi être une question quasi incontournable, qu'elle soit résolue de façon "détournée" par la création d'une liste des actions (aléatoire ou par priorité) ou de manière frontale en traitant les différents conflits qui peuvent apparaître. C'est pourquoi, que le modèle utilise un modèle du temps discret ou un principe événementiel, la véritable question qui se pose est celle de la représentation de la simultanéité des actions. Il suffit de considérer l'hétérogénéité des actions qui peuvent être modélisées dans une simulation multi-agents pour mesurer à quel point cette tâche n'est pas triviale. L'indépendance qui peut exister entre deux actions n'est parfois qu'apparente.

Modéliser la simultanéité de deux événements n'est bien sûr pas l'apanage des seuls systèmes multi-agents. Le problème se pose dans les simulations classiques et ses conséquences sont du même ordre : c'est-à-dire la possibilité d'obtenir plusieurs états du système différents à $t + dt$ suivant l'ordonnancement du traitement des événements à l'instant t [Zeigler *et al.* , 2000]. Ce problème a donc une traduction dans les formalismes classiques. Les approches que nous avons présentées peuvent ainsi être rapprochées des deux principales solutions qui ont été proposées dans le cadre du formalisme DEVS. Dans ce formalisme, le problème se pose dans le cas des systèmes composés (où la sortie d'un composant peut constituer l'entrée d'un autre) où plusieurs fonctions de transition peuvent être activées en même temps du fait d'événements possédant la même date de réalisation. En effet, par définition un composant atomique privilégie la fonction de transition qui représente l'influence des événements externes (δ_{ext}). De fait, l'ordre dans lequel sont activés les composants a une importance : la transition interne d'un composant, δ_{int}, peut entraîner l'annulation de la transition interne d'un autre par l'émission d'un événement sur son port d'entrée.

Dans ce contexte, on parle des *collisions* entre fonctions de transition. Du fait de cette ambiguïté, la modélisation des systèmes composés **nécessite** d'apporter une solution formelle et explicite à ce problème de manière à définir une dynamique qui soit unique et contrôlée. Dans la forme classique de DEVS, la solution à ce problème est formalisée par la définition d'une fonction *Select* qui détermine, pour le système composé considéré, l'ordre dans lequel ses composants en collision seront séquentiellement activés. Autrement dit, les conflits sont ici résolus en amont par la constitution d'une liste ordonnée. Ce qui peut être rapproché des idées de brassage aléatoire des agents ou de la définition d'un ordre de priorité entre les actions/agents.

Cette première solution a cependant montré certaines limites qui ont poussé à la réalisation d'une extension du formalisme DEVS[37]. En effet, la sérialisation de l'exécution des composants ne reflète pas la situation à laquelle il faut faire face et elle ne permet pas d'exploiter concrètement et simplement le parallélisme d'un système dans sa modélisation. Chow et Zeigler ont donc proposé une extension du formalisme DEVS appelée *Parallel DEVS* [Chow & Zeigler, 1994]. Ces auteurs résument les objectifs et les raisons qui ont motivé cette extension de la manière suivante :

[37] il est intéressant de noter que cette extension est apparue plus de quinze ans après la définition de DEVS dans sa forme classique. Ce qui prouve que le problème de la simultanéité n'est pas immédiat à cerner.

> "***Collision Handling*** *: The behavior of a collision must be controllable. Predefining a collision behavior is a limitation on the modeling capability that is not a necessary price to pay for parallelism.*"

> "***Parallelism*** *: The formalism must not use serialization function that prohibits possible concurrences. The parallelism among the internal transitions and simultaneous events must be fully exploited.*"

On voit cette fois que la simultanéité est traitée en tant que telle et qu'elle doit faire partie de la modélisation. La solution apportée par *Parallel DEVS* consiste ainsi à traiter explicitement la simultanéité en permettant au modélisateur d'apporter une solution particulière à la gestion d'une collision. Pour cela, la définition de la dynamique d'un composant est augmentée d'une fonction supplémentaire, δ_{con}, *the confluent transition function*, qui est chargée de traiter explicitement le cas où les deux autres fonctions de transition classiques peuvent être déclenchées à la même date. Cette fonction permet ainsi au modélisateur de donner une véritable sémantique à la résolution d'une collision et elle lui donne l'occasion de prendre en compte la simultanéité en tant que telle. Il faut d'ailleurs noter que la fonction *Select* n'a ici plus aucune utilité et qu'elle est abandonnée dans *Parallel DEVS*. De plus, chaque composant peut recevoir non plus une seule entrée mais un ensemble d'événements simultanément, on parle d'un *bag of inputs* (idem pour les événements en sortie).

A priori, les solutions utilisées dans le cadre de la simulation multi-agents qui utilisent des résolutions de conflits peuvent être rapprochées de cette solution formelle : au lieu de sérialiser a priori les actions, on apporte une solution à leur concurrence. Il existe cependant une différence assez subtile entre ces deux approches. Dans *Parallel DEVS*, l'idée n'est pas fondamentalement de résoudre le problème où deux états du système seraient en ***conflit***. Au contraire, grâce à la fonction δ_{con}, le système possède une dynamique clairement définie qui n'engendre aucun conflit : la réponse au problème de la simultanéité est apportée avant de calculer l'état suivant du système. De plus, la notion de *bag of inputs/outputs* formalise le fait que des événements simultanés constituent un tout dont les éléments ne doivent pas être traités séquentiellement. Bien sûr, la définition de δ_{con} peut être fondée sur un principe similaire à la résolution de conflits, d'ailleurs il est tout à fait possible de retrouver le comportement de la fonction *Select*. Mais l'idée n'est pas là et le but avoué est bien de traiter la simultanéité de manière frontale.

A partir de là, on peut se demander pourquoi les modèles multi-agents n'ont pas intégré des points de vue et des objectifs similaires à ceux proposés par *Parallel DEVS*, ne serait-ce qu'en regard de sa dénomination. En effet, les modèles multi-agents en restent aujourd'hui à la sérialisation ou à la résolution de conflits. En fait, les modélisations multi-agents utilisent toutes fondamentalement la même représentation de l'action : par modification, directe ou indirecte, des variables de l'environnement : Σ définissant l'ensemble des états possibles du monde, l'action d'un agent est concrétisée par la modification discrète de l'environnement d'un état $\sigma \in \Sigma$ à un autre état σ'. En d'autres termes, l'action d'un agent est souvent représentée par un événement du type *modifier la variable d'état A avec la valeur x*. Comme le souligne Ferber, ces représentations classiques de l'action se prêtent mal en l'état à une implémentation simple et efficace de la concurrence. Elles confondent dans l'action ce qui est produit par les agents avec ce qui se produit effectivement :

> "*Elles mélangent le geste et le résultat du geste.*"

Ainsi, l'action d'un agent est effectivement le plus souvent représentée par le résultat qu'elle est supposée engendrer sur l'environnement. De fait, on se retrouve alors dans des situations où la simultanéité de certaines actions engendre des situations qui ne peuvent être que conflictuelles. Magnin en convient lui aussi :

> "*Si dans le modèle Sieme les règles gèrent les interactions entre les entités du système, aucune prise en compte générique des actions n'est proposée. En fait, actuellement une action se limite à l'exécution du code ayant un accès direct aux attributs de l'entité qui agit. Or le résultat d'une action n'est pas uniquement lié à l'entité agissante.*"

Conflits ou séquence d'actions/interactions, la simultanéité n'est jamais véritablement prise en compte à la base du modèle de simulation utilisé. Et peu importe qu'on simule avec une technique à base de pas de temps ou par événements, le problème vient bien de l'utilisation d'une représentation classique de l'action qui est inadéquate pour la modélisation de systèmes multi-agents. Ce qui engendre par conséquent aussi une représentation de l'interaction insatisfaisante. En effet, au-delà du problème de la modélisation de l'écoulement du temps, la véritable question qui se pose est celle de la représentation des actions simultanées, donc de l'***interaction***. Nous considérons que cette problématique, qui est dans un premier temps liée au problème de la modélisation de l'action dans les systèmes multi-agents, est tout à fait fondamentale et nous lui consacrerons les chapitres 6 et 7. Nous verrons alors comment l'idée de la fonction δ_{con} peut être appliquée dans le cadre des systèmes multi-agents grâce au principe Influence/Réaction. Nous montrerons ensuite ce que cela nous fait gagner du point de vue de la modélisation de l'interaction.

3.6 Synthèse : les quatre aspects d'un modèle de simulation multi-agents

Sur la base de ce que nous avons présenté dans la deuxième partie de ce chapitre, et à l'instar de la démarche proposée par l'approche Voyelles, nous pensons qu'il est intéressant de considérer la problématique de la modélisation et de la simulation des systèmes multi-agents suivant quatre aspects fondamentaux : l'architecture interne des agents, le modèle environnemental, la gestion du temps et la gestion des interactions. Nous distinguons ainsi quatre modules dans l'architecture d'un modèle multi-agents (figure 3.15) :

- le module ***comportement*** où la question concerne la modélisation des processus de délibération des agents (leur esprit).
- le module ***environnement*** où le problème consiste à définir les différents objets physiques du monde (l'environnement situé et le corps des agents) ainsi que la dynamique endogène de l'environnement.
- le module ***ordonnancement*** qui concerne la modélisation de l'écoulement du temps et la définition de l'ordonnancement utilisé.
- le module ***interaction*** qui concerne plus particulièrement la modélisation du résultat des actions et des interactions qu'elles entraînent à un instant t.

La modélisation et l'implémentation de chacun de ces modules et de leurs relations sont autant de points délicats qui soulèvent les différentes problématiques que nous avons présentées. Bien que généralement les modèles multi-agents reposent implicitement sur ce genre de décomposition, nous verrons que le module *interaction* n'est jamais véritablement clairement défini

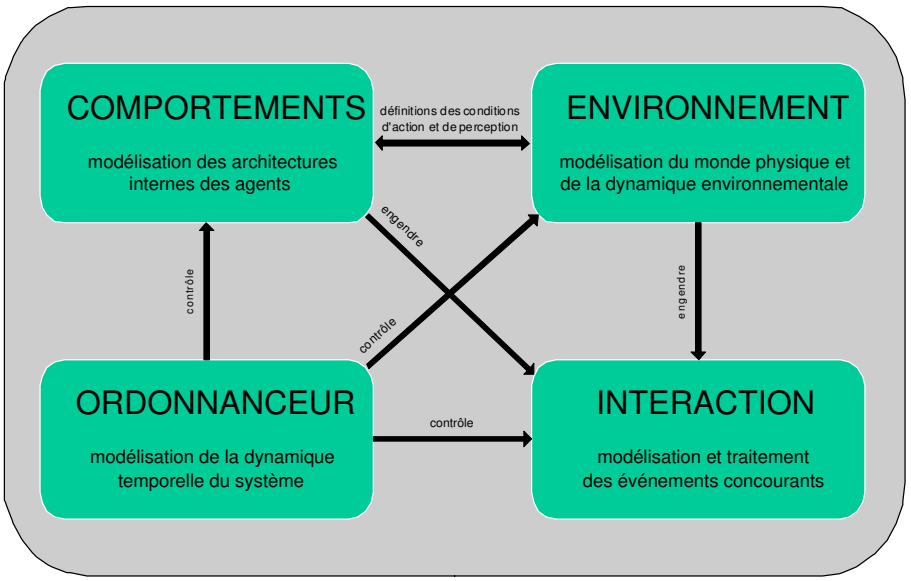

FIG. 3.15 – Les quatre aspects d'un modèle de simulation multi-agents.

en ce sens qu'il n'est jamais identifié en tant que tel et qu'il se fond le plus souvent dans la description des autres modules. C'est pourquoi, toujours à l'image de la philosophie proposée par l'approche VOYELLES, nous pensons qu'il est important de considérer explicitement chacun de ces différents aspects lors de la conception et de l'implémentation d'un modèle de simulation multi-agents. Nous ferons souvent référence à cette décomposition dans la suite et nous la mettrons concrètement en œuvre dans le chapitre 9.

Par ailleurs, le lecteur aura remarqué que nous ne faisons ici pas mention de la dimension organisationnelle d'un système multi-agents. Nous considérons qu'il s'agit d'une problématique d'un niveau supérieur qui concerne la modélisation du système multi-agents en lui-même[38]. En fait, la décomposition proposée dans VOYELLES ne doit pas être mise en correspondance directe avec celle que nous venons de faire. Un système multi-agents reste un composé AEIO. Ici nous nous sommes spécifiquement intéressé à définir les aspects fondamentaux qui sont liés à l'élaboration d'un **modèle de simulation** multi-agents. Toujours dans l'esprit de l'approche VOYELLES, il s'agit de proposer une décomposition générique qui pose un cadre conceptuel qui pourra être utilisé quelle que soit la méthodologie qui sera employée pour modéliser chacun de ces différents aspects[39].

3.7 Résumé du chapitre

Dans ce chapitre nous avons posé le contexte dans lequel s'inscrit cette thèse. Dans un premier temps nous avons présenté les grandes lignes du paradigme agent. Ensuite nous avons

[38] Une structure organisationnelle peut tout à fait être utilisée pour mettre en place les différents modules, c'est d'ailleurs ce que nous ferons pour déployer les outils de conception que nous proposerons dans le chapitre 5. Cependant, ce niveau n'intervient pas directement dans la définition de la dynamique du système proprement dite.

[39] C'est pourquoi ces différents aspects n'intègrent d'ailleurs volontairement pas de questions méthodologiques d'ordre supérieur comme celles qui sont par exemple liées à l'identification des différents acteurs qui participent à la conception de l'expérience dans son ensemble (thématicien, modélisateur, informaticien) [Vanbergue, 2003, Meurisse, 2004].

vu quelles étaient les motivations sous-tendues par la modélisation multi-agents et nous avons donné des exemples de quelques domaines où son principe est appliqué. Dans les sections 3.4 et 3.5 nous avons attiré l'attention du lecteur sur quelques-unes des questions qui se posent lors de la modélisation et de l'implémentation d'un système multi-agents et nous avons voulu souligner le fait qu'il ne s'agit aujourd'hui pas d'une tâche triviale. En ce qui concerne les agents et l'environnement, nous avons vu l'importance pratique et conceptuelle qu'il y a à faire une distinction explicite entre la partie physique d'un agent et son processus décisionnel, entre le *corps* et l'*esprit*. Lorsque nous avons considéré les problématiques liées à la gestion du temps, nous avons vu que la simultanéité des actions, propriété inhérente des systèmes multi-agents, constitue toujours actuellement un point de modélisation extrêmement difficile. De cet embarras découle aussi la difficulté que nous avons à représenter certaines interactions de manière satisfaisante dans les systèmes multi-agents. Nous approfondirons ces deux questions dans les chapitres 6 et 7 car nous les considérons en grande partie responsables des problèmes liés à la problématique que nous allons maintenant aborder.

Chapitre 4

Divergence implémentatoire : le problème de la réplication

DANS ce chapitre nous allons présenter quelques-unes des questions qui ont initiées et motivées nos directions de recherche. Comme nous le verrons, ces questions s'inscrivent dans le cadre d'une problématique qui se rapporte à la vérification et à la validation des simulations multi-agents. Ici, nous nous intéresserons plus particulièrement aux problèmes soulevés par la difficulté que nous avons à vérifier la *relation de simulation* (cf. section 2.8.3 page 42) dans ce type de simulation. Nous verrons notamment que les modèles multi-agents souffrent assez souvent d'un manque de spécifications qui rend difficile leurs réplications. Ce qui remet en cause la validité et l'intérêt des expériences. Dans un premier temps, nous isolerons l'une des conséquences les plus importantes de cet état de fait sous la forme de ce que nous appellerons le *phénomène de divergence implémentatoire*. Ensuite, nous essaierons de déterminer quelques-unes des raisons qui sont à l'origine de ce problème et nous présenterons les directions que nous avons prises dans le but d'expliciter, d'analyser et de minimiser ces phénomènes.

4.1 Aspects épistémologiques de la simulation multi-agents

4.1.1 Un cadre expérimental particulier

Etant donné son objet, la modélisation d'entités autonomes et de leurs interactions, la simulation multi-agents met en jeu des systèmes dont les dynamiques sont souvent très complexes et mal connues. Les plus grandes difficultés concernent sans aucun doute les modèles où le but est de modéliser le comportement d'entités humaines ou animales et où l'on ne peut se raccrocher à des modèles solides comme les lois de la physique. Il est clair que nous sommes aujourd'hui très loin de pouvoir modéliser le comportement d'un être animé avec fidélité et les modèles multi-agents que nous manipulons sont une représentation très simplifiée, voire simpliste, de la réalité. Par ailleurs, certaines expériences sont conduites sans être inspirées par une réalité à laquelle on pourrait confronter les résultats de la simulation, comme c'est fréquemment le cas dans le domaine de la vie artificielle par exemple. La modélisation multi-agents est donc un exercice difficile pour lequel il n'existe pas de méthodologie préétablie et où les approches formelles restent difficiles à appliquer [Klügl *et al.* , 2002]. Par conséquent, les modèles multi-agents sont souvent élaborés et présentés de manière informelle et le problème de leur validation reste une question difficile [David *et al.* , 2002].

Dans de nombreux cas, l'un des premiers objectifs n'est donc pas de coller fidèlement à la réalité que nous connaissons. En fait, l'exercice intellectuel que nous faisons lorsque nous modélisons la dynamique d'un système comme le produit d'un ensemble d'entités autonomes constitue déjà un objet d'étude en soi. Le modèle et sa simulation deviennent alors des outils qui nous permettent de partager des connaissances, des théories, des hypothèses et des points de vue sur le monde. D'une manière générale, lorsque la complexité d'un modèle nous échappe, la simulation joue ainsi principalement le rôle d'une source d'information qui permet d'alimenter les discussions [Kieken, 2003]. Ainsi, les raisons qui motivent une expérience de simulation multi-agents peuvent être très différentes selon le contexte. Par exemple, Resnick nous explique que sa motivation première n'est pas de comprendre un phénomène réel mais d'étudier la manière dont nous comprenons et imaginons des modèles basés sur une vision décentralisée du monde [Resnick, 1994] :

> "*In recent years, there has been considerable research into analytic techniques for describing and "solving" decentralized problems, and making accurate predictions about decentralized systems. But that is not my primary interest. Rather, I am interested in developing heuristics and qualitative tools to help people think about decentralized systems in new ways. My hope is that these conceptual tools will help people to move beyond the centralized mindset*"

Pour autant, grâce à l'aspect novateur du paradigme, la simulation multi-agents a su se rendre indispensable et son intérêt pratique est indéniable. L'approche multi-agents connaît aujourd'hui en effet un franc succès et elle est appliquée dans de très nombreux domaines avec parfois des enjeux importants. Suivant le contexte, si le but n'est toujours pas de reproduire fidèlement la réalité des interactions, on tente cependant de se rapprocher d'une certaine correspondance avec le monde réel. Bien sûr la difficulté que nous avons à modéliser ces systèmes complexes est encore d'actualité mais leur application dans des domaines comme la gestion de ressources renouvelables est la preuve d'une ambition certaine. Il ne s'agit plus seulement d'étudier les interactions et leurs propriétés mais bien de tirer des conclusions à partir des résultats fournis par la simulation. On parle d'ailleurs souvent de la simulation comme d'un *outil d'aide à la décision*. La complexité des processus mis en jeu dans une simulation multi-agents ne doit donc pas faire oublier qu'elle constitue un processus scientifique qu'il convient de soumettre autant que possible à des méthodes de validation et de vérification.

4.1.2 Ce que nous dit la *relation de simulation*

La vérification et la validation (V&V) d'une simulation sont des questions très difficiles qui comportent de très nombreux aspects et autant de méthodologies[1] [Balci, 1998, Sargent, 2001]. De plus, il est important de comprendre qu'il ne s'agit pas fondamentalement d'apporter une réponse qui soit strictement positive ou négative quant à la pertinence de l'expérience. Il s'agit plutôt d'élaborer des principes méthodologiques qui permettent de guider et de valider les différentes étapes du processus, de la définition du système à l'interprétation des résultats. Sargent en illustre les principales problématiques à l'aide du schéma de la figure 4.1.

Parmi les principes généraux donnés par la théorie, nous allons ici focaliser notre attention sur celui qui nous dit que l'expérimentation doit vérifier la *relation de simulation*. Comme nous l'avons vu dans le chapitre 2, cette relation soulève la question de la neutralité du simulateur vis-à-vis du modèle. Il s'agit de garantir autant que possible que le simulateur

[1] Dans [Balci, 1998], Balci dénombre pas moins de 77 techniques de vérification et de validation qui s'appliquent à différents moments du processus de la M&S.

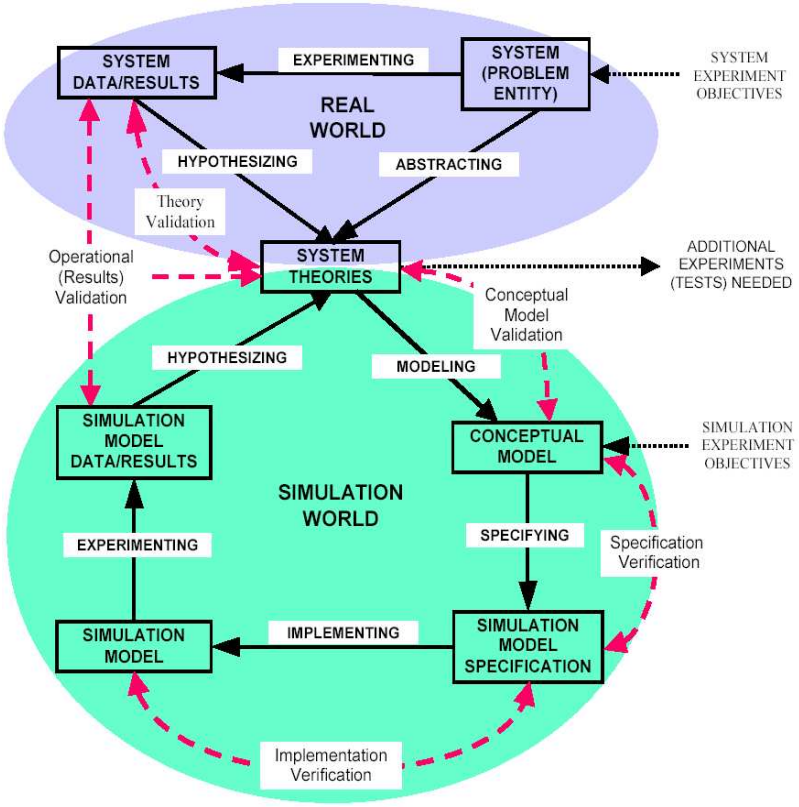

FIG. 4.1 – Vérification et de la validation d'un processus de simulation [Sargent, 2001].

exécute le modèle tel qu'il a été formulé et qu'il n'introduit pas de biais dans le processus de simulation dus à sa propre implémentation. Idéalement, un même modèle devrait en effet toujours donner les mêmes résultats quelle que soit l'implémentation. Si la programmation du simulateur influence la dynamique du modèle, le comportement du système observé au moment de l'exécution n'est pas le reflet du modèle tel qu'il a été conçu et les résultats ne doivent pas être interprétés avec confiance. Le problème n'est pas de savoir si la simulation est représentative de la réalité mais bien de faire en sorte que son application corresponde à la réalité que le concepteur a à l'esprit en élaborant son modèle. D'ailleurs, ce n'est pas parce qu'on étudie un système qui n'a pas d'existence physique que celui-ci n'a pas une réalité, en tant que modèle, qu'il ne faut pas trahir au moment de l'exécution : même si on ne souhaite pas réaliser une simulation du monde réel, il est cependant fondamental que les résultats ne soient pas affectés par le fonctionnement du simulateur.

Comme on peut le voir sur la figure 4.1, Sargent distingue deux aspects dans l'étude de la relation de simulation :

- *la vérification des spécifications* qui consiste à s'assurer que les spécifications du modèle de simulation sont satisfaisantes étant donné un système informatique cible. Elles doivent notamment permettre une implémentation non ambiguë de celui-ci.
- *la vérification de l'implémentation* où il s'agit de s'assurer que les spécifications du modèle ont été correctement concrétisées dans l'implémentation et que le simulateur ne comprend pas d'erreur de programmation et/ou de biais liés à l'environnement d'exécution (si les grands nombres ne sont pas stockés correctement par exemple).

Dans le domaine de la simulation multi-agents, ces deux questions sont en fait excessivement problématiques. Tout d'abord, comme nous l'avons dit dans la section précédente, les modèles multi-agents sont souvent présentés de manière informelle et la vérification des spécifications est assez souvent délaissée. Dans le même temps, nous avons vu dans le chapitre précédent que l'implémentation d'un simulateur multi-agents est beaucoup plus complexe qu'il peut paraître a priori et que de multiples facteurs sont susceptibles d'engendrer des biais de simulation difficilement identifiables. Bien que le domaine de la simulation multi-agents constitue ainsi un terreau si propice aux problèmes liés à ces questions, il nous faut cependant constater que la relation de simulation est une problématique qui est aujourd'hui loin d'avoir la place qu'elle mérite. Au contraire, les résultats d'une simulation sont souvent présentés sans aucune référence à cet aspect de l'expérience et, comme le souligne très justement Axelrod dans [Axelrod, 1997], communiquer les résultats d'une simulation sans donner des détails précis sur le modèle et son implémentation n'est pas d'une grande utilité pour les personnes qui souhaitent les exploiter.

Dans ce qui suit, nous allons tout d'abord plus précisément aborder la question de la vérification des spécifications dans les simulations multi-agents. Pour bien réaliser le manque de spécifications dont souffrent les modèles multi-agents, nous allons voir que celui-ci a une conséquence directe qui est observable et que nous appellerons le phénomène de la *divergence implémentatoire*. Ensuite, à défaut d'apporter une solution immédiate à ce problème, nous essayerons d'en étudier les causes afin de mieux comprendre son origine dans le contexte des systèmes multi-agents.

4.2 Le phénomène de divergence implémentatoire

4.2.1 Description

Faisons ici un petit rappel. Comme pour toute simulation informatique, le processus expérimental lié à l'élaboration d'une simulation multi-agents passe grossièrement par trois phases :

1. l'élaboration du modèle : durant cette phase, il s'agit de spécifier un modèle élaboré à partir du système source considéré.
2. exécution du modèle : les spécifications du modèle sont ici implémentées dans des structures informatiques concrètes qui sont exécutées de manière à obtenir le comportement du modèle.
3. les résultats obtenus sont alors analysés. Ceci peut amener à une révision du modèle (phase 1), en cas de non validation, ou à l'interprétation et à l'exploitation des résultats.

Lorsque les spécifications d'un modèle multi-agents sont insuffisamment précises du point de vue de leur implémentation, la deuxième phase de ce processus peut par conséquent engendrer un problème qui est lié à cette imprécision : ce même modèle peut être implémenté de plusieurs manières différentes et donner des résultats contradictoires suivant l'interprétation que fait le programmeur du modèle. Ainsi, la relation de simulation n'est pas vérifiée. Ce phénomène, la divergence implémentatoire, remet donc en cause les interprétations des résultats et la validité de l'expérience. La figure 4.2 illustre cette problématique.

Il est important de distinguer ce phénomène, qui concerne une faiblesse au niveau de la spécification de l'implémentation, de la divergence d'un modèle du fait de caractéristiques qui lui sont propres. Par exemple, un système chaotique diverge fortement à chaque fois qu'il

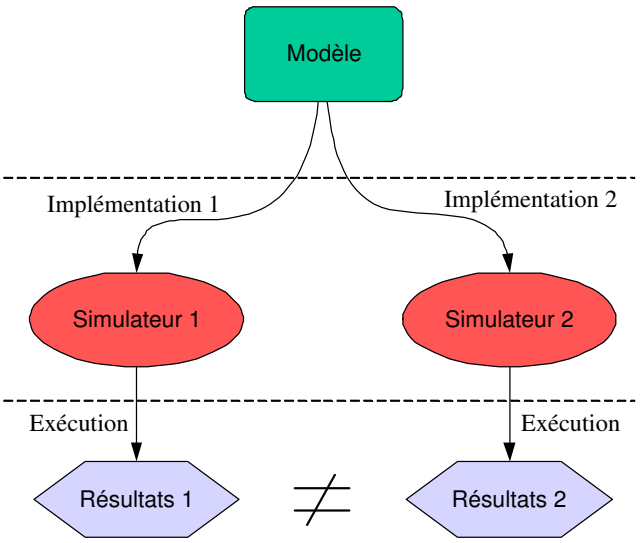

FIG. 4.2 – Le phénomène de divergence implémentatoire.

est exécuté avec de nouveaux paramètres initiaux, même si les changements sont minimes[2]. Ici, nous ne faisons pas référence à cela. Notre objectif est d'attirer l'attention du lecteur sur le fait que le manque de spécifications d'un modèle rend l'implémentation du simulateur sujet à interprétation et donc non neutre vis-à-vis des résultats obtenus du fait des biais potentiellement introduits. Ce qui peut avoir des répercussions importantes sur les résultats obtenus suivant les différentes implémentations qui sont possibles pour le simulateur.

A cela s'ajoute encore une fois le fait que l'implémentation d'une simulation multi-agents est une tâche excessivement complexe qui comporte de très nombreux paramètres dans chacun des modules que nous avons définis. C'est pourquoi, ce ne sont pas seulement deux ou trois implémentations qui peuvent être réalisées à partir d'une spécification incomplète, mais beaucoup plus. Et ce, avec des résultats à chaque fois différents des précédents du fait de la complexité des systèmes considérés. Cette potentialité à implémenter de nombreuses versions d'un modèle de simulation multi-agents, et les divergences qui en découlent, ont été étudiées dans des travaux comme [Huberman & Glance, 1993, Magnin, 1996, Axtell, 2000a, Lawson & Park, 2000, Michel, 2000, Meurisse & Vanbergue, 2001]. Comme nous allons maintenant le voir, il est donc naturel que l'une des conséquences directes du phénomène de divergence implémentatoire repose sur la difficulté de reproduire les expériences de simulation qui ont été menées antérieurement.

4.2.2 Illustrations du phénomène

Difficultés de répliquer les expériences

Plus généralement, la divergence implémentatoire s'illustre donc dans la difficulté que nous avons à aligner et à répliquer les modèles multi-agents qui sont proposés dans la littérature. Par exemple dans [Rouchier, 2003], Rouchier n'a pas été en mesure de reproduire les résultats du modèle proposé par [Duffy, 2001]. Et ce, même après s'être assurée auprès de l'auteur du modèle lui-même que les choix d'implémentation de la nouvelle version correspondaient à la

[2]Par exemple, la modification de la graine (*seed*) utilisée pour initialiser le générateur de nombres pseudo aléatoires est parfois suffisante pour modifier le comportement d'un système.

logique première du modèle. Ce qui l'a amenée à critiquer la validité du modèle initial mais surtout, et c'est le plus important, la manière dont celui-ci était exposé dans l'article référence. Autrement dit, les spécifications du modèle étaient trop pauvres pour pouvoir correctement répliquer l'expérience. Axelrod a fait à ce propos une remarque pertinente : la publication d'articles n'est certainement pas la meilleure méthode pour partager les résultats d'une simulation[3] [Axelrod, 1997]. Axelrod identifie principalement trois raisons à cela. Premièrement, les modèles considérés sont le plus souvent très complexes et le détail de leur dynamique n'est généralement pas donné de façon exhaustive pour des raisons de clarté et/ou de place. Deuxièmement, l'analyse des résultats est souvent exprimée sous une forme narrative qui ne fait pas apparaître tous les détails techniques qui permettraient de juger de sa validité. Enfin, de nombreux articles utilisent des raccourcis terminologiques susceptibles de tromper une audience interdisciplinaire.

L'expérience de Edmonds et Hales (2003)

Pour illustrer ces différentes idées, on peut s'attarder quelques instants sur la très intéressante étude qui a été menée par Edmonds et Hales dans [Edmonds & Hales, 2003]. Dans cette expérience, le modèle de [Riolo *et al.* , 2001], tiré de la revue *Nature*, a été réimplémenté indépendamment par les deux auteurs. Sans se concerter lors de la phase d'implémentation, ceux-ci ont obtenu des résultats assez proches l'un de l'autre mais significativement différents de l'expérience originale. Tirée de [Edmonds & Hales, 2003], la figure 4.3 schématise le processus expérimental qui correspond à la démarche qui a été suivie par ces deux auteurs :

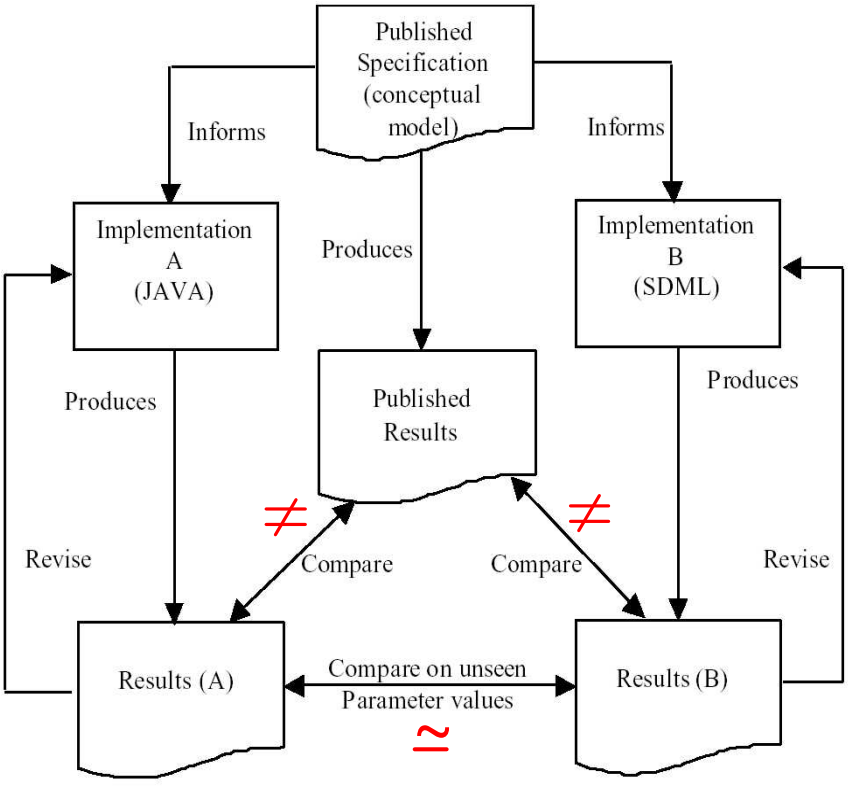

FIG. 4.3 – Une expérience de réplication par [Edmonds & Hales, 2003].

[3]Le texte original se place dans le contexte de la simulation des systèmes sociaux mais nous pensons que le point de vue d'Axelrod est tout à fait généralisable aux autres domaines abordés par la simulation multi-agents.

A cette divergence, les auteurs voient trois raisons possibles :

- l'implémentation utilisée pour obtenir les résultats publiés ne correspond pas au modèle décrit dans l'article original (la relation de simulation n'a pas été vérifiée).
- certains aspects du modèle n'ont pas été clairement énoncés dans la publication.
- les deux nouvelles versions du modèle ont été, indépendamment, incorrectement implémentées. De la même façon qui plus est : les deux nouvelles implémentations ont apparemment fait la même erreur.

Comme le souligne Edmonds et Hales, les deux derniers points sont bien sûr fortement liés : le deuxième entraîne le troisième. Pour trouver la véritable origine de la divergence des résultats, les auteurs ont alors implémenté plusieurs algorithmes différents pour un point où le modèle restait flou quant à la méthode utilisée pour comparer le score de deux agents[4]. Ainsi, après avoir introduit ce qu'ils considéraient comme un biais de simulation, ils finirent par obtenir des résultats équivalents à ceux qui avaient été publiés. Ensuite, les différentes implémentations obtenues leur ont permis de tester la robustesse des différents résultats et de mettre en évidence le caractère biaisé des conclusions qui avaient été publiées. Ce qui les a aussi amenés à reformuler le modèle original pour que celui-ci soit plus explicite. Voici quelques-unes des conclusions que Edmonds et Hales tirent de cette expérience :

- il devrait exister une norme quant à la publication de résultats obtenus par simulation. La description du modèle doit être suffisante pour permettre sa réplication par d'autres personnes (on retrouve ici le point de vue d'Axelrod).
- dans le cas où le modèle ne peut pas être étudié par une analyse formelle, la simulation doit être répliquée par différentes personnes (si possible avec différentes plates-formes). Cela peut en effet permettre de mettre en évidence les points où la description du modèle est ambiguë. Sans cela, les résultats ne peuvent pas être interprétés avec confiance.

Cette analyse montre clairement que le problème principal est bien celui de la vérification des spécifications d'implémentation. Elle démontre par ailleurs l'intérêt qu'il y a à effectuer une ou plusieurs réplications indépendantes du contexte originel. D'une part, cela permet de mettre au jour les faiblesses des spécifications proposées. D'autre part, cela permet d'avoir de nouveaux (souvent les premiers) points de comparaison auxquels les résultats de l'expérience originale peuvent être confrontés. Ce qui est souvent plus révélateur que lorsque le modèle est exploré par une seule équipe. Ainsi, lorsque cela est possible du point de vue des ressources humaines, la réplication indépendante est incontestablement un plus. D'ailleurs, elle fait partie des méthodes de validation préconisées dans les contextes classiques [Balci, 1998], bien que très peu utilisée dans les faits [Arthur & Nance, 2000]. Nous montrerons dans ce chapitre dans quelle mesure cette idée a guidé le développement des outils de conception de simulateur multi-agents que nous présenterons.

4.2.3 Pas de solution évidente

Les travaux que nous venons de citer dans les sections précédentes sont autant d'exemples qui montrent que la divergence implémentatoire est un problème fondamental qui soulève la question de la validité des simulations multi-agents. C'est pourquoi, diminuer ce phénomène autant que possible est un véritable défi et constitue un enjeu capital pour la communauté multi-agents : cela doit faciliter le partage des résultats et augmenter la crédibilité de ce processus expérimental.

[4]Dans cette simulation, les agents possèdent des variables quantitatives, des *tags*, modélisant des caractéristiques sociales qui sont comparées pour déterminer la similarité potentielle de deux agents.

Rappelons ici encore une fois la principale cause qui engendre le phénomène de divergence implémentatoire. Les spécifications d'implémentation qui décrivent la dynamique du système sont souvent insuffisantes, que cela soit dans le modèle original et/ou dans la publication correspondante. Son implémentation est alors ambiguë et la relation de simulation ne peut pas être vérifiée.

Partant de là, la solution à cette problématique semble toute trouvée : il suffit a priori d'utiliser des spécifications d'implémentation suffisamment puissantes pour que toutes les ambiguïtés de programmation du modèle puissent être levées, même à partir d'un modèle décrit par le langage naturel. Si de plus, un tel formalisme a le bon goût d'être applicable à l'ensemble des modèles multi-agents qui peuvent exister, si tant est que cela soit possible, alors le tour est joué. Un des objectifs de ce document est d'apporter une pierre à cet édifice. Cependant, l'accomplissement de ce but ne saurait être novateur si l'on s'arrête sur une analyse aussi simpliste de la situation. Plaider l'utilisation de formalismes rigoureux revient évidemment à enfoncer une porte ouverte dans la mesure où la majorité des modélisateurs sont bien sûr conscients de leurs bienfaits. De plus, même s'ils constituent des approches particulières et difficiles à appréhender, il existe bien sûr quelques travaux basés sur des formalismes qui permettent une implémentation non ambiguë, notamment les travaux qui tournent autour de l'utilisation du formalisme DEVS par exemple ([Uhrmacher, 2001b, Barros, 1997, Duboz, 2004]). Et bien que la majorité des modèles multi-agents proposés n'utilisent pas des spécifications satisfaisantes, les travaux que nous avons présentés dans la section précédente prouvent qu'il existe une prise de conscience de ces phénomènes de divergence et que le problème est connu.

En fait, comme nous l'ont encore rappelé Edmonds et Hales, les modèles multi-agents actuels se prêtent difficilement à une analyse formelle. C'est pourquoi il nous semble ici encore plus intéressant de chercher à savoir pourquoi les spécifications de ces modèles sont si difficiles à exprimer et si souvent imprécises. Attribuer cette lacune à un simple manque de rigueur ne serait bien sûr ni constructif, ni réaliste. Autrement dit, nous allons essayer d'identifier quelques-unes des raisons pour lesquelles les implémentations des simulations multi-agents sont si difficiles à spécifier. Nous allons donc nous intéresser aux causes de la cause du phénomène.

4.3 Aux origines du problème

4.3.1 La tentation du programmeur

Loin d'être un cas marginal, il arrive que l'implémentation constitue le seul modèle de l'expérimentation. Autrement dit, il n'est pas rare que le modèle ne soit pas explicité, sous une autre forme que le langage naturel, en dehors du programme informatique utilisé pour réaliser la simulation. Quel programmeur de simulations multi-agents n'a jamais implémenté un simulateur sans élaborer au préalable un modèle papier de la dynamique du système ? Ce qui s'explique en partie par le manque de formalismes adaptés aux systèmes multi-agents trouve aussi ses raisons dans le fait qu'il arrive parfois qu'on ne sache pas exactement ce que l'on veut simuler. En conséquence, le programme informatique résultant est souvent la seule manifestation du modèle que le designer ait à l'esprit.

Prenons un exemple. Lorsque la démarche consiste à inférer un système par exemple, l'objectif est d'obtenir un modèle ayant un comportement macroscopique à partir d'interactions microscopiques qu'on ne connaît pas encore et qu'on cherche à découvrir. Dans ce cas, un premier modèle informel est utilisé pour réaliser une première implémentation. La suite logique d'une telle expérimentation consiste bien sûr à réviser le modèle en fonction des résultats

obtenus par la simulation. Cependant, il n'est pas rare que la révision de ce modèle soit directement concrétisée par la modification de l'implémentation. A cela plusieurs raisons, d'une part le caractère informel du modèle laisse une grande liberté au programmeur et d'autre part la complexité du programme obtenu pousse rapidement son concepteur à considérer que certaines parties de son code peuvent être buggées, ce qui l'amène à réviser son implémentation sans avoir l'impression de toucher au modèle.

Il est bien sûr important de programmer un simulateur en étant conscient qu'il ne faut pas faire de faute de programmation, cependant il y a ici un énorme piège : un programmeur est capable d'ingéniosité. Par conséquent, tant que le comportement du système obtenu ne correspond pas aux attentes ou qu'il lui paraît incohérent, le programmeur est fortement tenté de modifier l'implémentation jusqu'à l'obtention de résultats qui le satisfassent. Autrement dit, celui-ci va passer son temps à modifier le simulateur, en changeant les mécanismes liés aux différents modules (scheduling, comportements des agents, gestion des actions et interactions, représentation de l'environnement, etc.), pour que ce dernier réponde à ses attentes. Etant donné la multitude des paramètres qui peuvent engendrer des biais de simulation, il est ici important de comprendre que le simulateur est, à ce stade de l'expérimentation, un système pour lequel il n'existe pas encore de modèle papier équivalent.

A partir de là, il faut se demander dans quelle mesure le simulateur obtenu correspond encore à l'idée du modèle originel : en est-il encore le reflet ? Si l'on prend le contre-pied de cette question et que l'on considère maintenant que l'implémentation constitue elle-même le modèle, une autre question fondamentale doit être posée : peut-on, à partir du code informatique, établir des spécifications qui permettent de le comprendre, de l'exposer et de le répliquer ? Sans la possibilité de réplication, nous avons vu que l'interprétation des résultats n'a pas beaucoup d'intérêt pour ses utilisateurs potentiels. De plus, cet exercice de traduction, du fonctionnement du simulateur vers un modèle papier, est par ailleurs une étape indispensable si l'on souhaite communiquer les résultats de l'expérimentation. Cette démarche pose cependant le problème de sa faisabilité. En effet, les changements apportés au code source qui modifient la dynamique du système sont souvent assez subtils, ce qui ne facilite pas leur traçabilité surtout lorsque les modifications ne sont pas ajoutées de façon atomique. Par conséquent, leurs implications dans le résultat global en deviennent difficilement identifiables [David *et al.* , 2002]. La première question de ce paragraphe doit alors être posée dans le sens inverse : le modèle élaboré à partir du code informatique représente-t-il effectivement le fonctionnement du simulateur ? On peut légitimement en douter pour toutes les raisons que nous avons déjà exposées. Nous pouvons résumer notre point de vue de la manière suivante : simuler ne veut pas dire programmer.

4.3.2 Le problème de l'interdisciplinarité

Nous avons vu que la simulation multi-agents est aujourd'hui utilisée dans de nombreux domaines de recherche. Ceci n'est bien sûr pas sans poser quelques problèmes qui contribuent au phénomène de divergence implémentatoire. Au-delà des questions liées aux variations sémantiques des différentes terminologies employées, l'interdisciplinarité pose encore une fois le problème de la traduction des modèles envisagés dans des structures informatiques qui leur correspondent [Vanbergue, 2003, Meurisse, 2004]. Autrement dit, nous nous intéressons ici aux problèmes posés par le passage d'un modèle multi-agents élaboré par l'expert d'un domaine particulier, et donc non informaticien, à son implémentation sur ordinateur par un expert en informatique. Comme nous allons le voir, ce processus est bien sûr affecté par les différents points que nous avons déjà exposés auparavant. En fait, il les catalyse.

Comme nous l'avons dit dans le chapitre précédent, l'approche multi-agents constitue un outil de modélisation dont les définitions des principes fondamentaux (autonomie, proactivité, etc.) restent encore aujourd'hui assez floues quant à la manière dont ils doivent être implémentés. En partie liée à la volonté de ne pas restreindre le champ d'application des systèmes multi-agents en contraignant leurs implémentations (d'où l'utilisation systématique de la définition faible d'agent, cf. section 3.1.1 page 45), cette absence de correspondance entre le paradigme et des structures informatiques concrètes ne facilite en rien la modélisation et la simulation de ces systèmes. Comme le souligne très justement l'analyse proposée dans [Drogoul *et al.* , 2002], la difficulté que nous avons à spécifier ce qu'est réellement un agent d'un point de vue computationnel se retrouve à tous les niveaux de la conception d'une simulation multi-agents. Notamment, les auteurs expliquent que l'expert d'un domaine particulier ne peut avoir qu'une idée approximative de ce qu'il est véritablement possible de modéliser ou non grâce à ce paradigme. En effet, dans le cas général, celui-ci ne possède pas la connaissance qui lui permettrait d'appréhender la manière dont son savoir peut être concrètement traduit en langage informatique.

Par conséquent, lors de l'élaboration de son modèle, l'expert se concentre principalement sur la spécification de deux aspects de celui-ci : la définition du comportement des entités individuelles et la nature de l'environnement dans lequel celles-ci vont évoluer. Cette approche a bien sûr l'inconvénient majeur d'évacuer en partie les problèmes qui sont liés à l'implémentation. La tentation dont l'expert en informatique chargé d'implémenter le modèle fait déjà l'objet est alors largement amplifiée : malgré les imprécisions du modèle, il lui faut cependant l'implémenter d'une manière ou d'une autre. En conséquence, la dynamique du modèle est ici aussi fortement modifiée par l'interprétation que celui-ci en fait lorsqu'il l'implémente. Il est alors, encore une fois, impossible de garantir que le programme informatique obtenu est bien l'expression de ce que l'expert du domaine a en fait à l'esprit.

On ne peut pas ici ne pas mettre en cause l'expert en informatique. En effet, il est de sa responsabilité d'informer l'expert du domaine qui lui a commandé l'expérimentation des limitations et des problèmes qui sont liés à l'utilisation du paradigme agent. Ceci est d'autant plus important que cela doit permettre d'éclaircir les points où le modèle proposé pose des problèmes quant à son implémentation. A ce propos, on peut dire que la simulation multi-agents est en quelque sorte victime de son succès : de part son aspect novateur, elle a séduit des personnes d'horizons divers qui ne sont pas toujours au fait des problèmes que nous avons soulevés. Par ailleurs, la montée en puissance régulière des processeurs et l'ergonomie croissante des plates-formes ont facilité l'accès à cette technologie en réduisant les investissements humain et financier nécessaires à de telles expérimentations.

4.3.3 L'embarras du choix

Nous avons vu que la manière dont sont publiés les résultats d'une simulation est un facteur qui ne facilite pas leur réplication et leur réutilisation dans un autre contexte que celui de l'expérience originelle. A propos de réutilisation, le très grand nombre de plates-formes existantes rend bien sûr encore un peu plus difficile le partage des travaux de recherche. Il faut alors se demander pourquoi il existe autant de plates-formes.

Lorsqu'on en vient à attaquer l'étape d'implémentation du modèle, deux solutions sont envisageables : implémenter le modèle sur une plate-forme préexistante ou développer son propre simulateur. Vu le nombre important de plates-formes aujourd'hui disponibles, on pourrait s'imaginer qu'il n'y a que l'embarras du choix et qu'il existe forcément une plate-forme qui

répond aux besoins de l'expérience. En fait, il s'agit bien d'un embarras. A vrai dire, la majorité des projets de simulation multi-agents choisissent la deuxième solution et développent leur propre plate-forme. Ce qui peut sembler paradoxal vu les efforts supplémentaires qui doivent être déployés pour aboutir à un résultat. En fait, il existe principalement deux raisons à cela.

La première est liée au manque de souplesse des plates-formes de simulation. En effet, suivant la plate-forme cible, le modèle est plus ou moins contraint par les modèles environnementaux et les architectures agent qui sont supportés par celle-ci. Autrement dit, il est nécessaire de formater le modèle de manière à ce qu'il puisse être implémenté en utilisant le "langage" du simulateur. Dans certains cas cela peut être impossible, dans d'autres le modèle s'en trouvera considérablement affaibli. Etant donnée l'hétérogénéité des domaines abordés, il n'est alors pas surprenant qu'il soit parfois nécessaire d'élaborer une plate-forme spécialement conçue pour répondre aux besoins particuliers de l'expérience considérée.

La deuxième raison vient de ce que les techniques d'implémentation utilisées pour les différents modules qui composent une plate-forme de simulation sont souvent encapsulés et/ou très mal documentés. Finalement, étant donnée la complexité des programmes utilisés pour coder un simulateur multi-agents, chaque plate-forme constitue une instance singulière qui apporte ses propres réponses aux problèmes posés par la modélisation et l'implémentation des modèles multi-agents. Ce qui a pour conséquence de rendre le fonctionnement interne d'un simulateur quelque peu ésotérique pour toute personne n'ayant pas participé à son élaboration. En tant qu'utilisateur, il est ainsi souvent très difficile, voire impossible, de maîtriser l'ensemble des rouages du processus de simulation et donc de contrôler que le simulateur n'engendre pas des biais dus à sa propre implémentation. C'est pourquoi, il est sans aucun doute beaucoup plus rassurant de développer son propre simulateur car on peut alors en connaître tous les aspects. L'opération de développement peut dans un premier temps paraître contraignante mais c'est bien la clé qui permet de véritablement contrôler son expérimentation.

Il faut donc considérer que, dans la majorité des cas, un projet de simulation nécessite l'implémentation d'une plate-forme qui lui convienne. Bien sûr, une telle plate-forme peut dans une certaine mesure être utilisée pour d'autres problèmes liés au domaine concerné mais il faut garder à l'esprit qu'il est très difficile d'obtenir une plate-forme "clé en main" ayant un fort pouvoir de généricité. Les développeurs de la plate-forme Swarm l'ont très bien compris. En effet, cette plate-forme ne propose pas un système de simulation "tout en un" où seul le comportement des agents reste à coder. Au contraire elle constitue un ensemble de bibliothèques et de facilités logicielles destinées à fournir les moyens de développer sa propre application. Le très grand nombre d'applications élaborées sur le principe de Swarm (Repast, Echo, etc.) démontre l'utilité d'une telle démarche. D'ailleurs, à propos de Swarm, il est plus juste d'utiliser les termes *outils logiciels de conception de simulation* plutôt que plate-forme. Les outils de simulation que nous présenterons dans le chapitre suivant ont été conçus dans le même esprit.

4.3.4 Le manque de formalisme adéquat

Dans le chapitre précédent, nous avons vu que la question de la modélisation de l'interaction soulève de nombreuses difficultés. Nous avons notamment abordé ce sujet lorsque nous avons présenté les problèmes liés à la représentation des actions simultanées. En fait, la difficulté que nous avons à modéliser de telles situations est sans aucun doute en partie responsable du caractère informel de l'approche multi-agents. En effet, alors que ce paradigme de modélisation se fonde sur la définition d'un ensemble d'entités autonomes agissant de concert

sur un environnement commun, très peu d'approches proposent des solutions qui prennent en compte la simultanéité des actions sous une autre forme que la résolution de conflits ou la mise en ordre séquentielle des actions. Et comme nous l'avons vu, ce point précis est extrêmement délicat et peut contribuer à rendre le simulateur générateur de biais et les solutions couramment envisagées ne sont pas forcément satisfaisantes du point de vue de la modélisation du système. En fait, la manière dont ce problème est finalement modélisé fait souvent partie des points flous du modèle. Points pour lesquels l'implémentation finale est donc largement laissée à l'appréciation du programmeur en charge du simulateur.

D'une manière plus générale, le manque de formalisme généralement associé aux modélisations multi-agents se reflète dans l'absence de correspondance entre les modèles qui sont utilisés pour définir un modèle de simulation multi-agents et des structures informatiques concrètes.

4.4 Choix de recherche

4.4.1 Résumé de la problématique

Résumons la situation. La divergence implémentatoire est un phénomène dont il nous faut essayer de diminuer les possibilités d'apparition. Dans le même temps, nous avons vu que le problème de la vérification des spécifications d'implémentation dans les simulations multi-agents n'a pas de solution systématique dans la mesure où les modèles considérés sont difficiles à formaliser et à maîtriser. En effet, nous avons vu que le codage des différents modules qui composent un simulateur multi-agents recèle de nombreux paramètres susceptibles d'influer sur les résultats obtenus. Par ailleurs, la complexité de la dynamique obtenue ne facilite pas l'identification de ces différents paramètres, même pour le concepteur du simulateur. C'est pourquoi il est difficile de faire le recensement des biais de simulation qui peuvent potentiellement apparaître du fait de l'implémentation. De plus, certains de ces biais sont si subtils qu'ils ne sont généralement pas repérés, même lorsque l'analyse du fonctionnement du simulateur n'est pas superficielle. Autre point délicat, dans certains cas il n'existe pas de système référence auquel on puisse confronter les résultats obtenus par simulation. Ce qui facilite encore moins l'évaluation des problèmes liés à l'implémentation. Finalement, nous pensons qu'une grande partie des difficultés liées à la modélisation et à la simulation des systèmes multi-agents se résume à un manque de correspondance entre les spécifications des modèles et les structures informatiques qui permettent de les exécuter. Dans la suite de ce document, les solutions, ou plutôt les approches, que nous proposons s'articulent autour de deux axes différents mais cependant complémentaires.

4.4.2 Première approche : sur les structures informatiques

La première direction que nous prendrons concerne une approche méthodologique de la manière dont il nous faut conduire nos expériences de simulations multi-agents, notamment lorsqu'une approche formelle du modèle fait défaut. L'approche proposée dans ce cadre repose sur la proposition d'outils génériques permettant de concevoir des simulateurs dont le fonctionnement soit explicite et flexible. Notre motivation est double :

– rendre explicites les structures informatiques utilisées dans un simulateur. L'idée est que si les spécifications des modèles doivent correspondre à des structures informatiques, il

est nécessaire de les rendre explicites. Ce qui permettra aussi de faciliter l'élaboration de réplications indépendantes.
– permettre l'exploration d'un modèle en facilitant les possibilités de variations sur son implémentation. Il s'agit ici de faire un effort sur la réutilisabilité des outils qui sont conçus

Réplications indépendantes

Si l'on part du constat que la dynamique de tout modèle multi-agents peut potentiellement être biaisée par son implémentation, ce qui est vrai dans la majorité des cas, et que de surcroît la plupart des modélisations sont proposés de façon informelle, une réplication indépendante de l'expérience originale est sans doute l'un des meilleurs moyens de contrecarrer le phénomène de la divergence implémentatoire. Dans le vocabulaire de la simulation classique, on qualifie ce type d'approche de *vérification et validation indépendante* (*independent verification and validation* IV&V) [Arthur & Nance, 1996, Sargent, 2001]. Sargent distingue deux façons de procéder suivant que la réplication est effectuée en parallèle de l'expérience originale ou a posteriori. A ce propos, il est intéressant de noter que la première est jugée beaucoup plus rentable que la seconde. Sargent explique que l'une des conclusions de l'étude menée par [Wood, 1986] sur de nombreux projets est que, pour un coût prohibitif, le seul bénéfice d'une réplication a posteriori est de permettre une évaluation qualitative des moyens de vérification et de validation qui ont déjà été mis en œuvre dans l'expérience originale. Finalement, qu'elles soient appliquées a priori ou a posteriori, les méthodes de IV&V n'ont jamais été vraiment très populaires et leur intérêt n'a que rarement été mis en avant.

Arthur et Nance pensent que c'est un tort et que la plus-value de cette approche est largement sous-estimée dans le domaine de la simulation [Arthur & Nance, 2000]. Leur idée est qu'il s'agit d'une méthode qui doit être mise en parallèle avec ce qui se fait aujourd'hui dans l'ingénierie logicielle classique où ce type de méthodes est largement intégré dans le processus de développement. L'importance des bêta testeurs dans le cycle de vie d'un logiciel complexe en est la parfaite l'illustration.

Dans le cadre de notre problématique, l'expérience de Edmonds et Hales nous a clairement montré l'utilité de la réplication[5] et nous avons vu que cette approche comporte de nombreux avantages. Dans un premier temps elle est évidemment le moyen de rendre explicite un phénomène de divergence implémentatoire lorsque les différentes réplications ne donnent pas les mêmes résultats. Mais surtout, elle peut potentiellement révéler les différents points où le modèle souffre d'un manque de spécification engendrant une ambiguïté sur son implémentation. Par ailleurs, en fournissant une nouvelle batterie de résultats, elle donne aussi un point de comparaison supplémentaire (parfois le premier) qui est le bienvenu.

Pour que ce type de méthodes puisse être généralisé, il est important que la communauté élabore de plus en plus d'outils de conception logicielle qui facilite le développement de simulateurs multi-agents quel que soit le contexte de leur utilisation (cf. section 4.3.3). Autrement dit, l'un des principaux défis est aujourd'hui celui de la réutilisabilité des logiciels que nous concevons. Les outils de conception de simulateurs que nous présenterons dans le chapitre suivant ont été élaborés autour de cette idée.

[5] Il est très intéressant de noter que les auteurs n'ont pas seulement réalisé une réplication a posteriori, mais qu'ils ont aussi élaboré un nouveau modèle indépendamment l'un de l'autre. Démarche qui leur a été d'une grande utilité pour mettre au jour les insuffisances des spécifications du modèle original.

Variations sur l'implémentation

Réaliser une réplication dont l'implémentation est effectuée par une tierce personne n'est cependant pas toujours possible pour des raisons évidentes de disponibilité en moyens humains. Dans le cas où une seconde implémentation réalisée indépendamment par un autre programmeur n'est pas possible, répliquer une seconde fois l'expérience suivant de nouvelles orientations peut sembler inutile. Il y a effectivement de fortes chances pour que la nouvelle implémentation ne soit fondamentalement pas différente de la première étant donné qu'on ne change pas de programmeur. De plus, il s'agit là d'une tâche très contraignante dans la mesure où le développement d'un seul simulateur demande déjà de nombreuses heures de travail. Réimplémenter un nouveau simulateur peut alors paraître bien trop coûteux pour un résultat dont on fait l'hypothèse qu'il sera similaire. Ce qui peut bien sûr être un mauvais calcul.

Faut-il pour autant se limiter à une seule version de l'implémentation ? Nous pensons bien sûr que non. S'il n'est effectivement pas très productif de redévelopper soi-même entièrement un simulateur, il est par contre tout à fait pertinent d'implémenter des variations sur les différents modules qui le constituent. Même si, du point de vue de la validation, cela n'a pas la même valeur qu'une réplication indépendante, modifier les points clés du fonctionnement interne du simulateur présente les mêmes intérêts : d'une part cela permet d'identifier une potentielle influence de celui-ci sur les résultats et d'autre part cela donne des points de comparaison concrets. Certains des travaux que nous avons précédemment cités, notamment [Axtell, 2000a, Lawson & Park, 2000, Edmonds & Hales, 2003, Meurisse & Vanbergue, 2001, Michel, 2000], sont basés sur cette idée : à partir du constat du phénomène de divergence implémentatoire, ils montrent la nécessité d'explorer plusieurs implémentations différentes.

Finalement, de la même manière qu'il est souhaitable d'effectuer une analyse de sensibilité du modèle à ses différents paramètres, on voit ici toute la nécessité de concevoir des outils de simulation permettant d'explorer les différentes façons dont un système multi-agents peut être implémenté afin de mesurer leurs impacts sur les résultats produits par le simulateur. Dans le cadre de l'analyse de sensibilité, le logiciel *SimExplorer* propose par exemple une approche qui permet d'explorer les différents paramètres d'un modèle indépendamment de sa nature grâce à l'identification de fonctionnalités logicielles génériques [Amblard *et al.* , 2003]. Nous verrons dans le chapitre suivant que les outils de conception de simulateur que nous décrirons ont été élaborés dans le même esprit en ce qui concerne l'implémentation du modèle lui-même. Par ailleurs, ces outils nous ont permis de conduire nos recherches suivant la deuxième direction que nous souhaitons aborder dans ce document.

4.4.3 Deuxième approche : sur les principes de modélisation multi-agents

Notre deuxième angle d'attaque s'inscrira dans une approche plus formelle du problème qui nous amènera à considérer de nouveaux moyens de construire un modèle basé sur le paradigme agent. Dans le chapitre 6, nous essaierons de montrer pourquoi il nous semble aujourd'hui important de revoir une partie de nos principes de modélisation.

Si la question de la représentation de la simultanéité se trouve comme nous le pensons au cœur du paradigme multi-agents, il nous faut alors nous demander pourquoi ce problème a aussi peu d'écho dans la communauté, proportionnellement à l'importance que nous lui donnons. Pourquoi ne traitons-nous pas la simultanéité en tant que telle dans nos modèles ? En fait, il se trouve que cette question ne se pose pas de façon intuitive et explicite pendant l'élaboration du modèle, notamment lorsque celle-ci se focalise sur la définition des comporte-

ments. Ceci précisément parce qu'il s'agit aujourd'hui d'un problème qui n'apparaît vraiment en tant que tel qu'au moment de l'implémentation : le traitement du comportement des agents est forcément réalisé de façon séquentielle. Même sur une machine multiprocesseur à mémoire partagée, la modification de cette dernière est strictement séquentielle, réaliser deux affectations simultanément sur une même variable n'a d'ailleurs aucun sens. C'est donc seulement au moment de l'implémentation que nous essayons de trouver des solutions aux problèmes de la concurrence des actions et que nous appliquons des méthodes de résolution de conflits par exemple. Mais, le problème n'ayant pas été pris en compte dans la définition même du modèle, les solutions que nous lui apportons ne sont donc pas forcément satisfaisantes (cf. section 3.5 page 64).

D'une manière plus générale, nous pensons qu'il est important de faire remonter les problèmes, qui sont liés à l'outil informatique, parmi les préoccupations qui concernent l'étape de modélisation d'un système. Si l'on veut pouvoir définir des spécifications d'implémentation qui soient sans ambiguïté, il nous faut en effet définir des modèles qui prennent explicitement en compte les contraintes qui sont liées à la nature de l'outil que nous utilisons. Par ailleurs, une telle démarche doit permettre d'élaborer des modèles qui soient toujours plus cohérents avec l'idée que nous nous faisons de l'approche multi-agents, en forçant notre réflexion sur notre façon de modéliser de tels systèmes. Nous aborderons ces questions dans les chapitres 6 et 7.

4.5 Résumé du chapitre

Dans ce chapitre nous avons présenté le phénomène de la divergence implémentatoire. Notre but a été attiré l'attention du lecteur sur le fait que ce problème soulève la question de la vérification de la relation de simulation qui est un point fondamental de la théorie de la M&S. A travers quelques exemples, nous avons vu que le manque de spécification des implémentations utilisées pour la simulation multi-agents constituent la principale cause de ce phénomène.

Cependant, nous avons vu qu'une simulation multi-agents est un processus complexe où il est très difficile d'intégrer formellement l'ensemble des paramètres susceptibles de modifier la dynamique d'un modèle. De par la complexité des modèles considérés, il est en effet très difficile de garantir que l'implémentation d'un modèle correspond exactement à ce qui est désiré et le manque de spécification est un problème pour lequel il est difficile d'apporter une solution définitive et universelle étant donné le grand nombre de facteurs qui interviennent dans ce processus expérimental. C'est pourquoi, la première approche du problème que nous avons présentée consiste à considérer que ce phénomène est toujours potentiellement présent et qu'il est ainsi nécessaire d'explorer plusieurs techniques d'implémentation. Le chapitre suivant est dédié à la présentation d'outils génériques de conception de simulateur qui permettent une telle approche.

Nous avons aussi introduit la deuxième approche que nous aurons dans ce document et qui reposera sur une réflexion portant sur les principes que nous utilisons pour modéliser un système multi-agents. Celle-ci concernera notamment la question de la modélisation de l'action et de l'interaction. Comme nous l'avons dit, nous considérons en effet que la difficulté que nous avons à spécifier un modèle multi-agents trouve aussi son origine dans les limitations des principes de modélisation que nous utilisons pour les définir. Les chapitres 6 et 7 sont consacrés à ces questions.

Nous ferons dans la dernière partie de ce document une synthèse des différents points que nous aurons abordés et nous proposerons un principe de modélisation qui garantit une implémentation non ambiguë grâce à une mise en correspondance directe entre le modèle et son implémentation. Le schéma de la figure 4.4 illustre notre démarche.

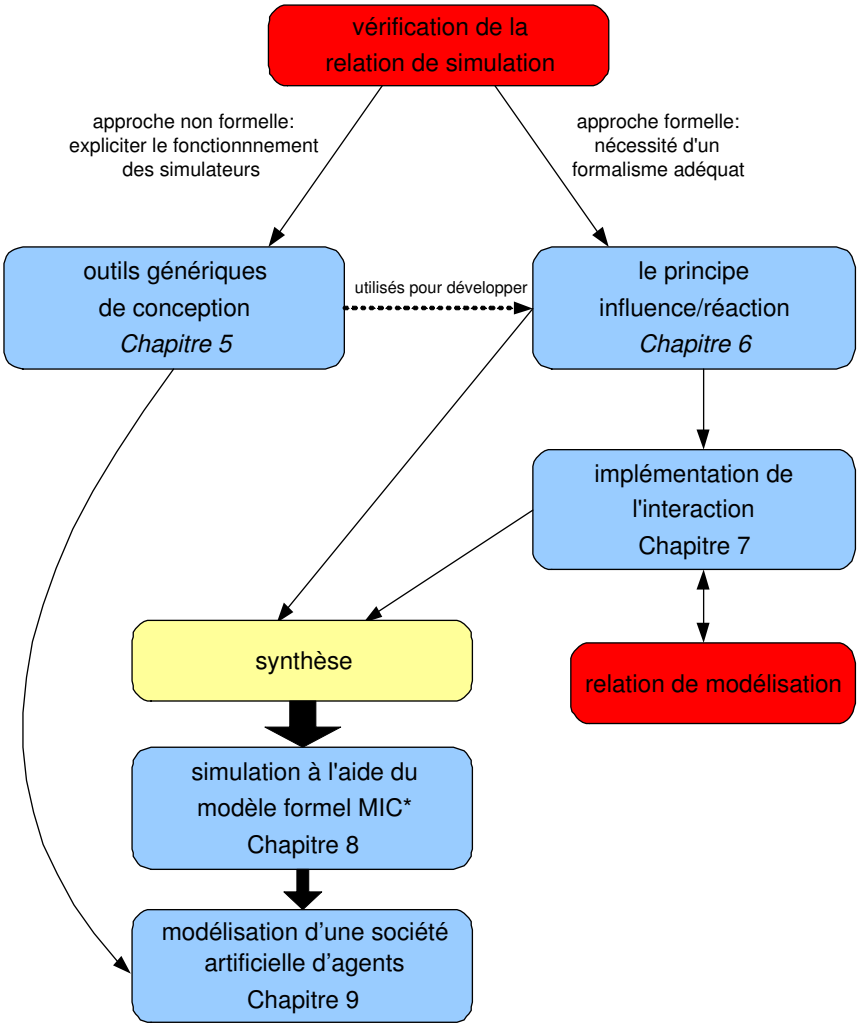

FIG. 4.4 – Plan des chapitres suivants.

Chapitre 5

Outils génériques de conception de simulateurs multi-agents

POUR véritablement prendre la mesure des problèmes qui se posent lors de l'implémentation d'un simulateur multi-agents, il nous fallait bien sûr nous impliquer dans la réalisation d'outils permettant de réaliser ce type de logiciel. Dans ce chapitre nous allons détailler les outils de conception de simulateur de la plate-forme MADKIT et nous présenterons quelques-unes des applications qui en sont issues.

5.1 Intérêts et objectifs

Etant donné le contexte que nous avons présenté dans le chapitre précédent, il est très important de fournir aux développeurs des outils logiciels leur permettant d'implémenter des variantes de leurs simulateurs le plus aisément possible. Elaborer ce type d'outils constitue aujourd'hui un véritable défi pour les informaticiens dans la mesure où, d'un point de vue génie logiciel, il nous faut considérer la réutilisabilité des outils que nous concevons. En quelque sorte, il est important de penser ces outils de conception en tant que "langages de programmation de modèles".

De plus, bien que cette problématique se situe à un niveau très technique, l'intérêt d'une telle démarche vis-à-vis des utilisateurs non informaticiens n'en est pas moindre : plus l'implémentation d'un simulateur sera rendue explicite et intelligible moins les modèles comporteront de spécifications incomplètes ou ambiguës.

Il nous faut donc concevoir des bibliothèques d'outils de conception suffisamment souples pour permettre l'élaboration de simulateurs dont l'implémentation des différents modules soit facilement interchangeable. Autrement dit, alors que dans le cas général les plates-formes de simulation encapsulent le fonctionnement interne du simulateur, de tels outils doivent le rendre explicite et permettre sa modification a posteriori.

Dans la majorité des cas, seul le module qui implémente le comportement des agents est facilement accessible et modifiable : il est en général possible de remplacer le comportement d'un agent par un autre sans trop de difficultés. Ce qui est un minimum étant donné que le comportement des agents constitue bien sûr le principal objet d'étude. Par contre les autres modules sont le plus souvent inaccessibles et difficilement remplaçables. Il est alors impossible de modifier le fonctionnement interne du simulateur sans remettre en cause une grande par-

tie du code. Le problème est donc bien d'élaborer des outils de conception dont le pouvoir de généricité concerne aussi les autres modules de la simulation : le module environnemental, le module interactionnel et le module d'ordonnancement. Dans ce chapitre, nous nous intéresserons plus particulièrement aux problèmes liés à l'implémentation du module d'ordonnancement. Nous donnons cependant ici quelques indications quant à la manière dont nous traiterons les autres modules.

Module environnemental

Parmi les travaux qui se sont penchés sur ces problèmes de génie logiciel, la plupart d'entre eux abordent la question de l'indépendance du code représentant l'agent par rapport à celui de l'environnement. Comme nous l'avons vu au chapitre précédent (cf. 3.4), il s'agit effectivement d'un point délicat très difficile à appréhender du fait de la complexité du lien qui existe entre un agent et son environnement : la difficulté que nous avons à délimiter conceptuellement la frontière qui sépare ces deux entités se retrouve au niveau de l'implémentation. Ce qui rend difficile la modification de l'architecture de l'environnement indépendamment de celle des agents.

Dans ce contexte, les travaux que nous avons précédemment cités comme [Soulié, 2001, Lhuillier, 1998] proposent des approches de conception qui sont sans aucun doute des exemples à suivre. En effet, la réification systématique du lien qui existe entre un agent et son environnement sous la forme de deux types d'objets distincts, capteurs et effecteurs, permet effectivement d'envisager la modification de l'environnement sans invalider le code des agents grâce à une distinction claire entre la partie cognitive et la partie physique de l'agent. Ici, nous n'aborderons pas directement ces questions mais les outils que nous proposons nous laisseront une entière latitude sur la manière dont ce module peut être implémenté. Dans le chapitre 8, nous verrons comment ce point de vue peut être généralisé par une approche formelle de la modélisation de l'environnement et des agents.

Module des interactions

Comme nous l'avons vu dans le chapitre précédent, le problème de la modélisation des actions et des interactions est en général considéré comme une sous problématique liée à l'implémentation des deux modules précédents. En effet, le problème se pose à la fois pour l'implémentation du module environnemental , pour lequel il faut définir la manière dont les agents agissent sur l'environnement, et pour l'implémentation du module d'ordonnancement, à travers le problème de la modélisation du déroulement des actions et des interactions. Ainsi, cette problématique ne fait pas l'objet d'une analyse particulière et le module des interactions n'existe pas en tant que tel, c'est-à-dire comme un composant autonome. Ce module est en quelque sorte "éparpillé" dans les autres parties du simulateur.

Les outils logiciels de simulation que nous allons maintenant présenter ne prennent pas directement en compte la question de l'interaction. En fait, nous verrons dans le chapitre 7 qu'il s'agit d'une question qui concerne l'élaboration du modèle, et non le fonctionnement interne du simulateur au sens d'une plate-forme d'exécution. C'est pourquoi ces outils ne font aucune supposition sur la manière dont est implémentée cette question. Nous verrons cependant pourquoi il est important d'associer cette problématique à un module particulier dissocié des autres et nous proposerons une méthodologie de modélisation qui permet de le faire dans le chapitre 8.

Module d'ordonnancement

Paradoxalement, alors que le module d'ordonnancement joue clairement un rôle fondamental dans le processus de simulation, c'est aussi le moins étudié et les outils réellement dédiés à la conception et à l'implémentation de différentes techniques d'exécution ne sont pas légion. Parmi les rares approches existantes, on peut noter que la plate-forme Ascape, dont les modèles de simulation sont basés sur la définition d'un ensemble de règles comportementales, permet à l'utilisateur de modifier, à l'aide d'une interface graphique, la manière dont ces règles sont appliquées sur les agents lors de l'exécution : par type de règle (une règle sur une ensemble d'agents) ou par agent (un ensemble de règles sur un seul agent) [Parker, 2001]. Meurisse propose aussi un outil de simulation permettant de modifier l'ordre dans lequel les différents comportements des agents seront exécutés[1] [Meurisse & Vanbergue, 2001]. A l'aide d'une interface graphique et sans modification du code original, cet outil permet de tester plusieurs séquences d'activation différentes grâce à la manipulation de composants représentant des comportements atomiques (des méthodes d'objets en fait). Cet outil est cependant destiné à des utilisateurs non informaticien et son utilisation nécessite la définition de comportements qui respectent une syntaxe de programmation particulière qui ne couvre volontairement pas toutes les techniques d'exécution possibles.

Swarm (et aussi maintenant RePast) fait aussi ici figure d'exception. Il est en effet possible de modifier assez facilement la manière dont le système est exécuté en modifiant directement les listes d'activités qui sont définies dans un swarm (cf. section 3.3.3). Ce qui est normal étant donné que cette question fait partie intégrante d'un processus de modélisation réalisé dans le contexte de la plate-forme Swarm. Cependant, aussi souple soit-elle, la "machine virtuelle" de Swarm impose elle aussi de définir des dynamiques qui reposent en fait sur une unique politique d'exécution dont le principe ne peut être modifié : la liste d'activités. Autrement dit, le noyau d'exécution de Swarm contient déjà la définition de ce qu'est la dynamique des systèmes qui s'exécuteront sur lui. Dans le cadre de notre étude, nous nous intéresserons à la conception d'outils permettant de définir des politiques d'exécution sur lesquels aucune contrainte n'est imposée. Cependant, ce qu'il faut retenir de ces différents exemples, c'est qu'ils ont le mérite de constituer des approches qui placent la problématique de l'ordonnancement à sa vraie place car ils la rendent explicite et manipulable. Cette question doit en effet faire partie intégrante de la modélisation d'un système. Les outils de conception que nous allons présenter dans la section suivante conservent cette idée mais n'imposent pas de contrainte sur la nature de l'implémentation de la politique d'ordonnancement choisie.

5.2 La plate-forme MadKit

Dans cette section, nous allons présenter les grandes lignes du modèle organisationnel *Aalaadin*, aujourd'hui beaucoup plus connu sous le nom du modèle Agent/Groupe/Rôle AGR [Ferber & Gutknecht, 1998, Gutknecht & Ferber, 1998] et de la plate-forme MadKit [Gutknecht *et al.* , 2001, Gutknecht, 2001] qui est basée sur celui-ci.

5.2.1 Le modèle AGR : Agent/Groupe/Rôle

Inspiré par les travaux de Gasser et la plate-forme MACE, le modèle *Aalaadin* se fonde lui aussi sur l'utilisation de concepts organisationnels. L'idée fondamentale de cette approche

[1] Ce qui permet d'ailleurs à l'auteur de mettre en évidence le phénomène de divergence implémentatoire

est de considérer qu'une structure organisationnelle hiérarchique permet de faciliter l'analyse, la conception et l'exécution de systèmes multi-agents hétérogènes.

Le modèle *Aalaadin* est basé sur trois concepts clés : l'*agent*, le *groupe*, le *rôle*. On parle ainsi du modèle AGR[2] (figure 5.1).

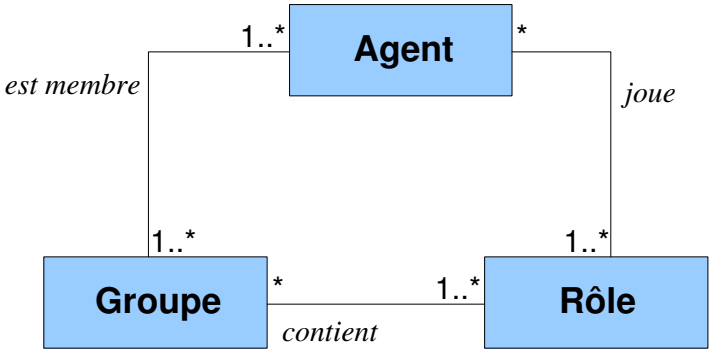

FIG. 5.1 – Le modèle Agent Groupe Rôle.

Agent

Quasiment aucune contrainte n'est posée sur l'architecture interne ou sur le modèle de l'agent. L'*agent* est simplement décrit comme une entité autonome communicante qui joue des *rôles* au sein de différents *groupes*. Cette très faible sémantique est volontaire. Aucune supposition n'est faite sur le processus décisionnel de l'agent et celui-ci ne repose sur aucun formalisme en particulier. Le but est de laisser toute liberté au concepteur pour choisir l'architecture qui sera la mieux appropriée à ses besoins.

Groupe

Le groupe est la notion primitive de regroupement d'agents. Dans une première approche, le groupe est donc vu comme un moyen de localiser un ensemble d'agents. Plus précisément, associé à un ensemble de *rôles*, il définit la structuration organisationnelle d'un système multi-agents usuel. Chaque agent peut être membre d'un ou de plusieurs groupes et les différents ensembles peuvent donc se recouper librement. Par ailleurs, un groupe peut être créé dynamiquement par un agent.

Rôle

Dans le modèle *Aalaadin*, le rôle est considéré comme la représentation abstraite d'une fonction, d'un service ou d'une identification d'un agent au sein d'un groupe particulier. Chaque agent peut avoir plusieurs rôles. Un même rôle peut être tenu par plusieurs agents. Les rôles sont locaux aux groupes. La tenue d'un rôle dans un groupe préexistant doit être demandée par l'agent et n'est pas forcément accordée. La maîtrise de l'hétérogénéité des situations d'in-

[2]Il faut noter qu'il existe des extensions intéressantes du modèle AGR, notamment [Amiguet *et al.* , 2003, Parunak & Odell, 2002, Ferber *et al.* , 2005].

teraction est rendue possible par le fait qu'un agent peut avoir plusieurs rôles distincts au sein de plusieurs groupes, et que les interactions sont toujours locales à un groupe.

5.2.2 Principes d'implémentation utilisés dans MadKit

La plate-forme MadKit [3] est l'incarnation de l'utilisation du modèle AGR. Dans MadKit, ce dernier est utilisé à la fois pour fournir une structure organisationnelle aux systèmes multi-agents qui sont exécutés et pour son propre fonctionnement interne. L'implémentation de MadKit intègre par ailleurs trois principes de conception supplémentaires : une *architecture à micro-noyau*, l'*agentification systématique des services* et l'utilisation d'un *modèle graphique componentiel*.

Architecture à micro-noyau

Seules les opérations de base de la plate-forme sont implémentées dans le noyau[4] : gestion de la structure organisationnelle, cycle de vie des agents, routage des messages et mécanismes internes pour la simulation.

Agentification systématique des services

Hormis l'utilisation de la structure organisationnelle et l'envoi de messages, l'ensemble des services offerts par la plate-forme est individuellement incarné par un agent ou un ensemble d'agents : surveillance du fonctionnement et de l'état du système, connexions avec un réseau de noyaux, gestion des messages non locaux, outils de simulation et d'une manière générale toutes les applications et extensions de la plate-forme. Outre l'utilisation du modèle AGR au niveau méta du fonctionnement de la plate-forme, le principal intérêt de cette approche est de permettre le remplacement d'un agent de manière transparente pour le reste du système multi-agents : celui-ci n'a en effet aucune répercussion sur le routage des messages si sa position dans l'organisation reste inchangée. Par ailleurs, cette approche permet de faire bénéficier tous les systèmes multi-agents qui sont exécutés sur la plate-forme de l'ensemble des fonctionnalités qui ont été implémentées par ailleurs : un agent, quelle que soit son architecture interne, peut potentiellement communiquer avec n'importe quel autre agent modulo sa place dans l'organisation.

Utilisation d'un modèle graphique componentiel

Chaque agent est responsable de sa propre interface graphique[5]. Ce qui permet de voir un agent comme une simple fenêtre ou de lui associer une interface correspondant à une application complète selon les besoins. De plus, un agent peut aussi fonctionner sans interface graphique comme c'est le cas pour la majorité des agents système. La plate-forme elle-même supporte plusieurs modes de fonctionnement graphique (au-dessus du noyau) : en mode texte, sous la forme d'une application graphique traditionnelle (menus, exploration des archives, etc.) et en mode application web sous la forme d'une Applet Java.

[3] www.madkit.org.

[4] L'ensemble du noyau fait aujourd'hui 75 kilos de bytecode. Un peu plus lourd que dans les premières versions de la plate-forme, le noyau actuel conserve cependant l'état d'esprit originel.

[5] Une fenêtre par défaut est cependant disponible lorsque les agents ne proposent pas leur propre interface.

5.2.3 Historique des versions

Initiée par Gutknecht et Ferber, la première version de la plate-forme MADKIT voit le jour à la fin de l'année 1997[6]. Nous avons personnellement rejoint le projet lorsque la plate-forme était dans sa version 1.3. A partir de là, on peut grossièrement résumer les différentes étapes qui ont amené la plate-forme dans sa version actuelle (4) de la façon suivante. La 1.4 a intégré la première version des outils de simulation que nous présenterons dans la section suivante. Les versions 2.x ont consisté dans une mise à plat des différentes nouveautés apportées par les versions précédentes et dans l'amélioration des mécanismes d'interfaçage graphique.

La version 3 correspond au portage du code source aux spécifications 1.2 du langage Java. Ceci a été l'occasion d'une refonte du micro-noyau qui a entraîné de nombreuses modifications sur certains points clés de la plate-forme, notamment au niveau de la gestion de la structure organisationnelle et de la synchronisation entre plusieurs noyaux qui est maintenant gérée par un agent système spécifique, le *SiteAgent*. Un nouveau niveau organisationnel, la *communauté*, a aussi été introduit à cette occasion. Une communauté permet de distinguer des parties de l'organisation (un ensemble de groupes) qui correspondent à une application en particulier, ce qui permet d'une part de clarifier la structure organisationnelle et d'autre part de limiter les échanges réseau dans un contexte distribué[7].

Ces différents changements sont cependant restés transparents pour l'utilisateur lambda, au contraire des modifications effectuées lors du passage à la version 4. Utilisé dans un premier temps en interne par notre équipe, la qualité des outils logiciels proposés par le projet *Apache Ant*[8] pour la gestion et la compilation de code source nous a poussés à repenser la modularisation du code source de manière à ce que l'indépendance des différents modules soit effective à la fois dans la gestion des sources, pour le développement et l'exécution.

A l'instar de nombreux logiciels libres actuels, la version 4 de la plate-forme fonctionne donc aujourd'hui sur un modèle d'application basée sur l'utilisation de plugins dont la mise à jour se fait individuellement. Autrement dit, la plate-forme est aujourd'hui distribuée dans une version minimale en taille et dont on peut récupérer les différentes parties via Internet selon les besoins. Ce changement était par ailleurs nécessaire étant donné le nombre croissant et l'hétérogénéité des applications développées par plusieurs dizaines de développeurs. Cette évolution a coïncidé avec une importante mise à jour de l'interface graphique principale et la mise en libre accès de l'ensemble du code source via le site Sourceforge[9].

5.3 Simulation dans MADKIT

5.3.1 Ordonner et observer

Comme nous l'avons dit dans le chapitre précédent, il est très difficile de proposer des outils qui soient génériques et qui puissent être utilisés quel que soit le contexte de l'expérimentation. On ne peut en effet guère faire de suppositions sur la manière dont le concepteur envisage de modéliser les agents, l'environnement, les interactions et la gestion du temps. Cependant, à l'instar de ce qui est fait dans SWARM, nous pensons que les programmes utilisés pour coder une

[6] www.madkit.org/madkit/doc/notes.html propose un historique des différentes versions de MADKIT.

[7] Les différents noyaux MADKIT peuvent par exemple se connecter uniquement sur une communauté particulière. Seuls les groupes qui appartiennent à cette communauté seront alors synchronisés.

[8] ant.apache.org/.

[9] sourceforge.net/projects/madkit/.

simulation multi-agents intègrent au minimum les problèmes suivants : gérer le déroulement de la simulation à l'aide d'une politique d'exécution et observer le fonctionnement du système en permettant de sonder les données du modèle (pour l'affichage et l'analyse des résultats).

Les outils de conception de la plate-forme MadKit sont donc implémentés autour de deux principales notions : l'ordonnancement et l'observation. Dans MadKit, les mécanismes qui permettent de traiter ces problèmes sont respectivement concrétisés sous la forme de deux agents intégrés au noyau : l'agent *Scheduler* et l'agent *Watcher*. L'utilisation de ces deux agents étant tout à fait similaire, nous ne présenterons ici en détail que l'agent Scheduler.

5.3.2 Organiser pour régner

A l'instar des outils que nous avons présentés précédemment, les outils de conception implémentés dans la plate-forme MadKit n'encapsulent pas le fonctionnement interne du simulateur et visent à le rendre explicite, notamment en ce qui concerne le problème de l'ordonnancement. Cependant, au contraire des approches précédentes, le but de ces outils est de fournir, aux programmeurs expérimentés qui souhaitent développer leur propre simulateur, les briques de base qui leur permettront de le faire sans pour autant leur imposer une technique de simulation particulière. Notamment, nous allons voir que ces outils n'imposent pas de faire un choix unilatéral en ce qui concerne la méthode d'ordonnancement choisie.

Il faut en effet remarquer que le fonctionnement d'une simulation est en général toujours basé sur une unique politique de gestion des agents : tous les agents sont soumis à un seul et même principe de synchronisation. A partir de là, la tâche peut s'avérer excessivement complexe dès l'instant où on s'attaque à un modèle comportant des agents très différents. En effet la granularité des actions (mouvement, vitesse, etc.) et la sémantique des interactions (collisions, coordination, etc.) peuvent être très différentes. De plus, il n'est pas rare que les processus qui accompagnent la simulation (affichages, analyse statistique, etc.) soient eux aussi gérés sur le même principe. Dans un système événementiel, l'affichage correspond à un événement qu'il faut gérer de la même façon que les autres. Ainsi le fait que tout le système soit architecturé autour d'une unique politique d'exécution contribue à rendre son analyse difficile et limite ses possibilités de modification et d'extension à d'autres agents.

Inspirée des principes de décomposition et de hiérarchisation proposés par Swarm, l'idée est simple et d'ailleurs elle n'est pas nouvelle dans le principe. Il s'agit de diviser le problème de l'ordonnancement global du simulateur en autant de sous problèmes que nécessaire. En effet il suffit de remarquer que – quelle que soit la méthode – résoudre les problèmes de synchronisation de tous les agents de la même manière n'a plus vraiment de sens lorsque l'on est amené à considérer plusieurs types d'agents, chacun de ces types pouvant nécessiter l'emploi de techniques de synchronisation et d'ordonnancement différentes.

C'est pourquoi, il est plus simple de se concentrer sur un groupe d'agents dont la synchronisation est considérée comme cruciale pour le processus de simulation. Ainsi, en élaborant des groupes formés d'agents dont la synchronisation des actions est nécessaire, on décompose le problème global : chaque groupe peut être traité indépendamment de façon à identifier un protocole d'ordonnancement adapté à la situation. De plus, ces groupes définissent une organisation multi-agents dont la nature reflète de façon explicite la logique du simulateur. Il s'agit donc ici de tirer parti de la structure organisationnelle utilisée dans la plate-forme MadKit pour définir des politiques d'exécution explicites.

5.3.3 L'agent Scheduler de MADKIT

Faisons ici un petit rappel sur les principes de conception de la plate-forme. MADKIT n'impose aucune architecture agent en particulier et elle est basée sur l'utilisation d'une structure organisationnelle. De plus, MADKIT intègre deux principes de conception supplémentaires : une architecture à micro-noyau et l'agentification systématique des services. Notre idée est d'avoir transposé ces principes de conception pour la simulation.

Nous avons donc implémenté un agent intégré au micro-noyau qui permet la conception de techniques hétérogènes d'ordonnancement : l'agent *Scheduler*. Sa fonction consiste à manipuler **des** politiques d'exécution sur lesquelles aucune contrainte n'est posée. Pour cela, cet agent est associé à un objet outil générique appelé *Activateur*.

Dans sa forme la plus simple un activateur est simplement le moyen pour le Scheduler d'identifier un ensemble d'agents étant donné un groupe et un rôle. Par exemple, un agent Scheduler peut créer un activateur sur le rôle *agent* dans le groupe *simulation*, ou encore *afficheur* dans le groupe *représentation graphique*. Plus précisément, une fois un groupe d'agents identifié, le fonctionnement d'un activateur consiste à déclencher un comportement spécifique sur ce groupe d'agents. Parmi les activateurs proposés par défaut par la plate-forme, l'un des plus simples consiste simplement à exécuter une méthode particulière sur l'ensemble des agents du groupe et du rôle concernés.

Du point de vue de la généricité et de la réutilisabilité, le principal avantage de ces outils tient dans l'utilisation conjointe de la structure organisationnelle et de la réflexivité du langage Java qui permet de s'affranchir de la classe exacte des agents (au sens objet). Il est ainsi possible de remplacer un agent par un autre de façon transparente pour l'ensemble du système, quelle que soit la nature du nouvel agent. Il suffit simplement que ce nouvel agent, modulo son positionnement dans l'organisation, implémente la (ou les) méthode sur laquelle l'activateur est défini.

Par ailleurs, la prise en compte des agents qui sont contrôlés par un activateur est dynamique, c'est-à-dire qu'un agent peut être lancé en cours d'exécution et être immédiatement intégré par le système grâce à la détection de son arrivée par la structure organisationnelle et donc par l'activateur concerné : pour un agent, le simple fait de jouer un rôle précis engendre ainsi sa participation au processus de simulation. En outre, rappelons ici encore une fois que les agents qui participent au processus de simulation sont aussi des agents MADKIT à part entière et qu'ils bénéficient de l'ensemble des fonctionnalités de la plate-forme (routage des messages, interface graphique individuelle si désirée, etc.).

Pour concevoir un simulateur particulier, le principe est de spécialiser, si nécessaire, de nouvelles sous-classes d'activateur de façon à définir une technique de scheduling particulière qui pourra ensuite être appliquée ponctuellement par le Scheduler, à différents groupes d'agents.

Dans MADKIT, les principales sous-classes d'activateur disponibles concernent la simulation à pas de temps constant et la mise en place d'un principe événementiel simple. Voici une partie de leur code :

`DiscreteTimeActivator` :

```
synchronized public SimEvent execute(double GVT){
        for (Iterator i = getAgentsIterator();i.hasNext();)
                executeBehaviorOf(i.next());
        return new SimEvent(this, intervalStep);
}
```

`DiscreteEventActivator` :

```
synchronized public SimEvent execute(double GVT){
        //get next event generator
        EventGenerator theAgent = (EventGenerator) ((EventObject) eventList.remove(0)).
            getSource();
        //execute the agent
        executeBehaviorOf(theAgent);
        // next event for this behavior
        double nextEventDeltaTime = theAgent.nextEventDeltaTimeFor(method);
        // internal list management
        eventList.add(new SimEvent(theAgent,GVT+nextEventDeltaTime));
        Collections.sort(eventList);
        // return this event
        return new SimEvent(this, nextEventDeltaTime+GVT);
}
```

Ces deux activateurs ne sont que des exemples de la manière dont ils peuvent être écrits. Ils peuvent être plus simples ou plus complexes suivant les besoins. L'avantage de cette décomposition en activateurs indépendants tient dans le fait qu'un même agent Scheduler peut créer autant d'activateurs que nécessaire. Sa tâche se résume à ordonner l'exécution de ses activateurs pour définir le processus de simulation dans son ensemble. L'agent scheduler est par ailleurs le garant de la cohérence du temps simulé global (Global Virtual Time GVT) dans la mesure où l'ordonnancement des différents activateurs est sous sa responsabilité. La figure 5.2 décrit par exemple un simulateur comportant un agent Scheduler S qui utilise deux types d'activateur (deux politiques d'exécution différentes) sur trois groupes d'agents. De plus cette figure montre qu'il est possible pour un agent d'appartenir à plusieurs groupes simultanément.

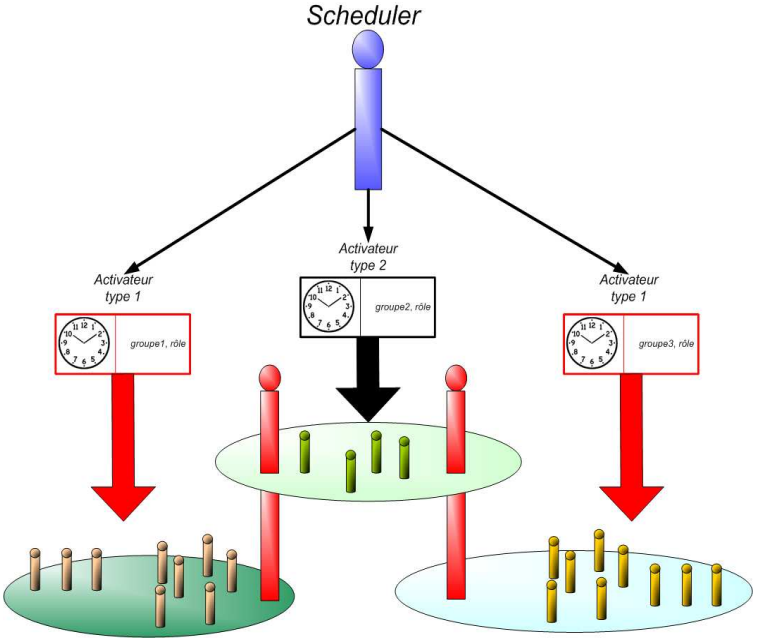

Fig. 5.2 – Un simulateur organisationnel : 2 types d'activateurs sur 3 groupes.

Le principal avantage de la décomposition du scheduling global réside dans la possibilité de modifier ou de remplacer un activateur sans avoir à toucher la structure globale : on a découplé les problèmes de gestion de la simulation (Scheduler) de celui de la gestion de la synchronisation des agents (Activateur), le Scheduler n'étant pas responsable de la qualité d'un activateur. Un agent Scheduler peut de plus, en tant que simple agent, faire l'objet d'un contrôle supérieur. Par ailleurs, de la même manière qu'un activateur est indépendant de

l'architecture des agents qu'il contrôle, la réutilisation des activateurs est soumis à très peu de contraintes et un agent scheduler particulier peut potentiellement utiliser des activateurs définis par d'autres personnes. Le cœur d'un agent scheduler peut-être le suivant (il s'agit ici du scheduler utilisé dans la plate-forme TURTLEKIT que nous décrirons rapidement un peu plus loin) :

```
/** agent activation function in MadKit*/
public void activate()
{
        requestRole(group, "scheduler");
        // a particular activator
        turtleDoIt = new TurtleActivator(group);
        addActivator(turtleDoIt);
        // basic activators
        oberserversDoIt = new TurboMethodActivator("watch",group,"observer");
        addActivator(oberserversDoIt);
        viewersDoIt = new TurboMethodActivator("display",group,"viewer");
        addActivator(viewersDoIt );
        diffusion= new TurboMethodActivator("diffusion",group,"world");
        addActivator(diffusion);
...
        oberserversDoIt.execute();
}

public void executeTurtles(){   turtleDoIt.execute();   }

public void scheduleWorld()
{
        executeTurtles();
        executeDiffusion();
        executeEvaporation();
        executeObservers();
        executeDisplay();
        incrementeIteration();
}
```

5.3.4 Méthodologie associée

La méthodologie de conception associée à ces outils définit trois étapes :

1. exprimer la structure organisationnelle du simulateur sous forme de groupes et de rôles.
2. élaborer les types d'activateurs nécessaires au simulateur, c'est-à-dire des techniques de gestion d'exécution locales.
3. définir le fonctionnement global du simulateur en ordonnant l'exécution des activateurs.

L'énoncé de la troisième étape peut laisser croire que nous nous trouvons face à la même problématique que celle liée à l'ordonnancement des agents. Comme nous l'avons dit, il s'agit bien de deux niveaux d'analyse distincts. En effet un activateur, lié à un groupe, définit une technique de scheduling particulière (à pas de temps constant, par événements, etc.) pour des agents dont la synchronisation est considérée comme nécessaire. Au contraire, l'ordre d'exécution des activateurs décrit simplement la logique du simulateur, c'est-à-dire l'ordre dans lequel les différents groupes clés (agents simulés, affichage, observation, etc.) interviennent dans le processus de simulation.

La figure 5.3 illustre cet aspect de notre approche. Elle décrit un exemple où quatre activateurs différents A1, A2, A3 et A4 sont utilisés. Chaque activateur est lié à un ensemble d'agents identifié par un groupe et un rôle. Les politiques de scheduling utilisées par les dif-

férents activateurs sont indépendantes. A1 peut par exemple utiliser une méthode à pas de temps constant alors que A2 définit un principe événementiel.

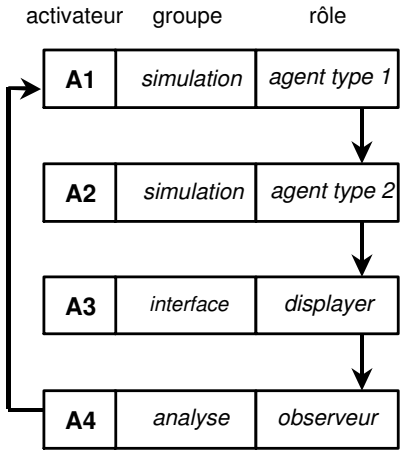

Fig. 5.3 – Principe d'ordonnancement des activateurs au sein d'un scheduler.

Comme nous l'avons mentionné précédemment, il est évident que l'ordre d'exécution des activateurs peut influer sur les résultats. Ainsi si on inverse A1 et A2, on peut obtenir des résultats différents étant donné que ces activateurs manipulent des agents (agent type1 et agent type2) qui peuvent modifier le monde.

Néanmoins il ne s'agit pas ici d'un problème de synchronisation : la décomposition en deux activateurs implique qu'on suppose que les différents groupes d'agents ne doivent pas intervenir en même temps (ou du moins qu'ils ne sont pas en interférence). Par contre, il est tout à fait envisageable de considérer, sur la base des résultats ou pour explorer plus profondément le modèle, que la synchronisation de deux groupes soit finalement nécessaire. Il est alors très intéressant de pouvoir changer localement le fonctionnement du simulateur. On peut par exemple imaginer la fusion de A1 et A2 en un seul activateur plus adéquat, A3 et A4 restant valides.

L'interchangeabilité des activateurs, obtenue grâce à la vue organisationnelle, trouve alors tout son sens : elle permet de constituer rapidement plusieurs simulations basées sur différentes techniques d'ordonnancement sans avoir à recoder les agents de la simulation, ni sa gestion globale. Il est ainsi possible de réaliser des variations de l'implémentation du simulateur très simplement et de manière dynamique, sans avoir à retoucher le code source.

Nous soutenons qu'une telle méthodologie permet de construire une politique d'exécution, complexe au niveau global, qui reste compréhensible, modifiable et donc analysable grâce à la décomposition du problème. La possibilité de pouvoir modifier **localement** le fonctionnement du simulateur permet la réalisation de simulations toujours plus complexes sans pour autant restreindre les possibilités d'exploration du modèle.

5.3.5 Exemples d'applications

Les outils et la méthodologie que nous venons de présenter ont été utilisés au sein et en dehors de notre équipe pour de nombreuses expériences de simulation dans des domaines divers. Nous présentons ici quelques-unes de ces réalisations et nous en mentionnerons d'autres plus tard dans ce document (notamment l'application *Warbot* dans le chapitre 6).

Robotique mobile collective

Les travaux de Simonin ont été parmi les premiers à essuyer les plâtres grandeur nature, c'est-à-dire dans le cadre d'un projet de recherche [Simonin, 2001] (figure 5.4). Finalement, après quelques améliorations, les outils de conception ont permis de réaliser une plate-forme de simulation multi-robots dont les possibilités ne sont pas figées [Simonin *et al.* , 2002]. Cette plate-forme de simulation a notamment permis d'étudier divers systèmes liés aux problématiques de la RMC : la résolution de conflits entre robots autonomes dans des environnements très contraints [Simonin & Ferber, 2001], le traitement des problèmes de fourragement par de nombreux robots sur de vastes environnements [Simonin & Ferber, 2000], la simulation de leurs communications locales, ainsi qu'une représentation dynamique 3D de leurs influences, la mise au point d'une architecture de navigation réactive pour robots mobiles autonomes [Lucidarm *et al.* , 2002]) et l'étude d'un module d'apprentissage introduit à ce modèle [Chapelle *et al.* , 2002].

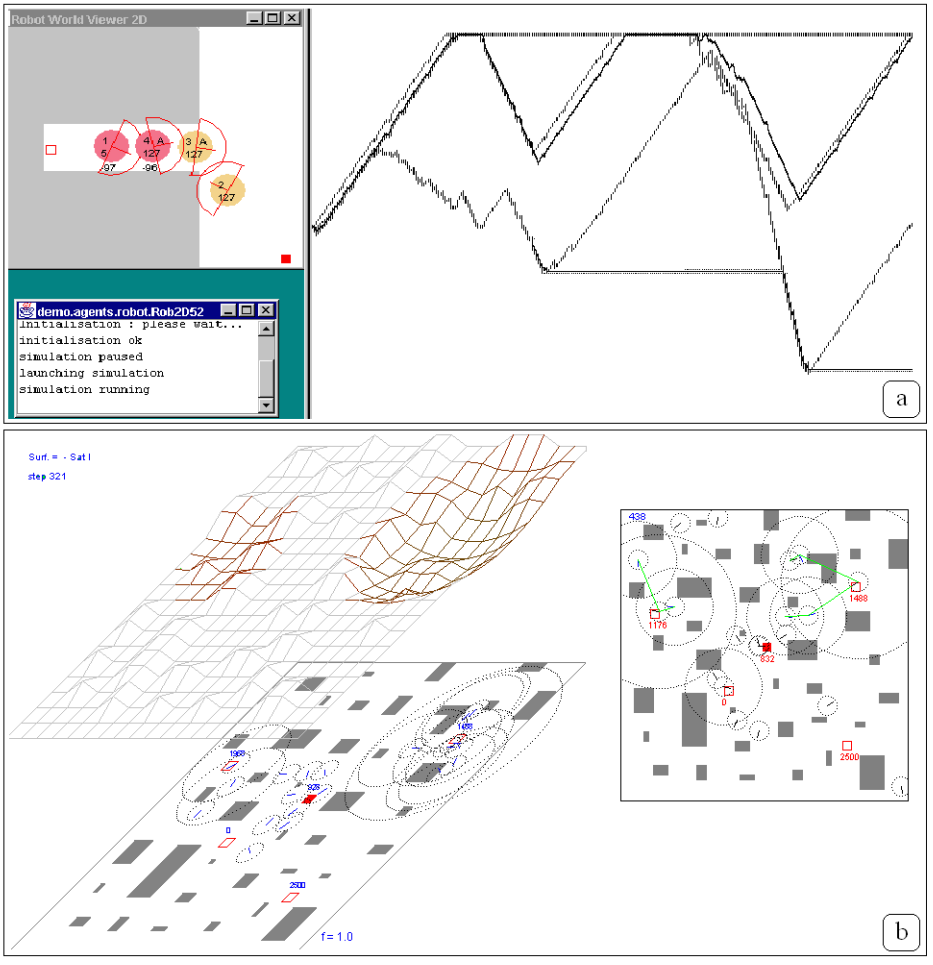

FIG. 5.4 – Simulations multi-robots dans MADKIT.

Simulation distribuée interactive

Comme nous l'avons dit, l'un des atouts majeurs de ces outils tient dans le fait qu'ils sont implémentés sous la forme d'agents et qu'ils sont ainsi capables de communiquer par message. Cette fonctionnalité a notamment facilité la conception d'une application de simu-

lation distribuée interactive [Michel & Bommel, 2002]. Inspirée du monde des jeux vidéo, le principe de cette application est de permettre à plusieurs utilisateurs connectés en réseau d'interagir sur une même simulation[10]. Dans ce cadre, la possibilité de pouvoir communiquer entre agents sans se soucier de leur localisation a été d'un intérêt remarquable. En effet, MadKit gère la distribution de manière transparente pour les agents en assurant la cohérence de la structure organisationnelle qui est partagée par l'ensemble des sites connectés. Le routage des messages est ainsi effectué par le noyau et les agents n'ont pas se préoccuper de savoir si leurs interlocuteurs sont exécutés sur un autre noyau ou non. Autrement dit, grâce à la structure organisationnelle, l'agentification des composants a pour principal intérêt de permettre à ces derniers de communiquer, notamment par broadcast sur un groupe, avec tous les agents présents dans l'organisation sans se soucier de leur localisation physique. En supprimant ainsi les contraintes techniques liées à la distribution, MadKit permet de se focaliser sur les problèmes d'algorithmique distribuée et de profiter au mieux des principaux atouts de la plate-forme (hétérogénéité et modèle organisationnel).

Hormis les agents simulés, cette application utilise 5 types d'agent. L'agent *Scheduler* est chargé de l'ordonnancement de l'ensemble des agents de l'application. Les agents de type *Controller* contrôlent le déplacement des agents. Les agents *Epiphyt* espionnent le système et envoient les informations recueillies aux agents de type *Projectionist* qui sont eux chargés de l'affichage. Enfin, les agents de type *InputListener* écoutent les interactions utilisateurs et les transmettent aux agents concernés (de type *Controller*).

Cette organisation est répliquée sur l'ensemble des sites où se déroulent les simulations. Les communications entre les agents se font quant à elles suivant la place qu'ils tiennent dans l'organisation. La figure 5.5 illustre le mécanisme qui a ainsi été mis en place. Par ailleurs, ce principe et les outils de simulation MadKit associés sont utilisés au cœur d'un jeu vidéo multi-joueurs en ligne, *Astronoïd*[11], développé par la société *Aleph 0*.

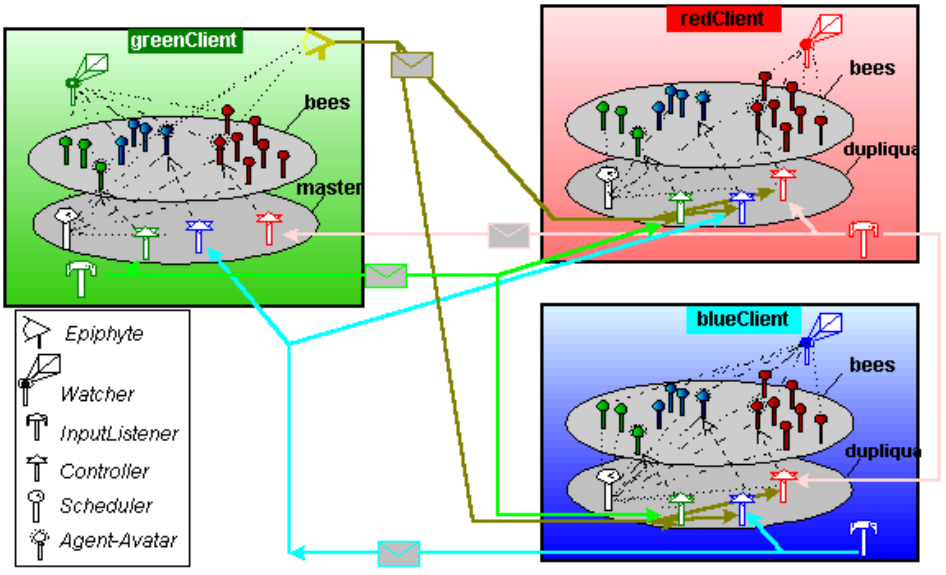

Fig. 5.5 – Principe de l'application de simulation interactive distribuée.

[10]La simulation consiste dans un ensemble d'agents se déplaçant dans un environnement continu. Outre la possibilité pour les différents utilisateurs de paramétrer la simulation, ils peuvent aussi contrôler le déplacement des agents à l'aide de la souris ou en leur imposant une politique de mouvement prédéfinie. Il est par ailleurs possible de filtrer la perception que chaque utilisateur a de la simulation en fonction de son rôle.

[11]Ce jeu vidéo est basé sur le principe d'une course galactique entre vaisseaux spatiaux www.astronoid.net.

La plate-forme TURTLEKIT

Le TurtleKit est une plate-forme de simulation pour agents réactifs qui clone une partie des principes de simulation des environnements STARLOGO/NETLOGO [Michel, 2002]. Elle a été pour nous le moyen de mettre à l'épreuve l'ensemble des fonctionnalités des outils de simulation puisqu'elle constitue une plate-forme de simulation complète : définition d'une architecture agent, de l'environnement, du principe d'exécution et de visualisation. Il est ainsi possible de programmer des simulations très rapidement en définissant des comportements simples.

Cependant, au contraire des environnements d'exécution qui nous ont inspirés, le propos de TURTLEKIT est de permettre aux utilisateurs expérimentés qui le souhaitent de modifier et/ou d'étendre l'ensemble des parties de la plate-forme le plus simplement possible. Ceci grâce à l'utilisation systématique des outils que nous avons présentés et à la mise à disposition de l'ensemble du code source, celui-ci ayant été programmé de manière à rendre explicites les différents points du modèle de simulation. Les agents de la simulation (interface graphique, scheduler, outil d'analyse, etc.) peuvent ainsi être librement redéfinis/étendus selon les besoins de l'utilisateur.

De nombreuses démonstrations de la plate-forme sont disponibles en ligne (sous la forme d'Applets exécutables) à l'adresse suivante : www.lirmm.fr/~fmichel/turtlekit. La figure 5.6 montre une copie d'écran de certaines de ces applications. TURTLEKIT a par ailleurs été utilisé par Beurier pour ses travaux de recherche qui concernent les problématiques d'émergence multi-niveaux dans les systèmes complexes [Beurier *et al.* , 2002]. Une version de cette application peut aussi être consultée en ligne à l'adresse suivante : www.lirmm.fr/~beurier/mle.

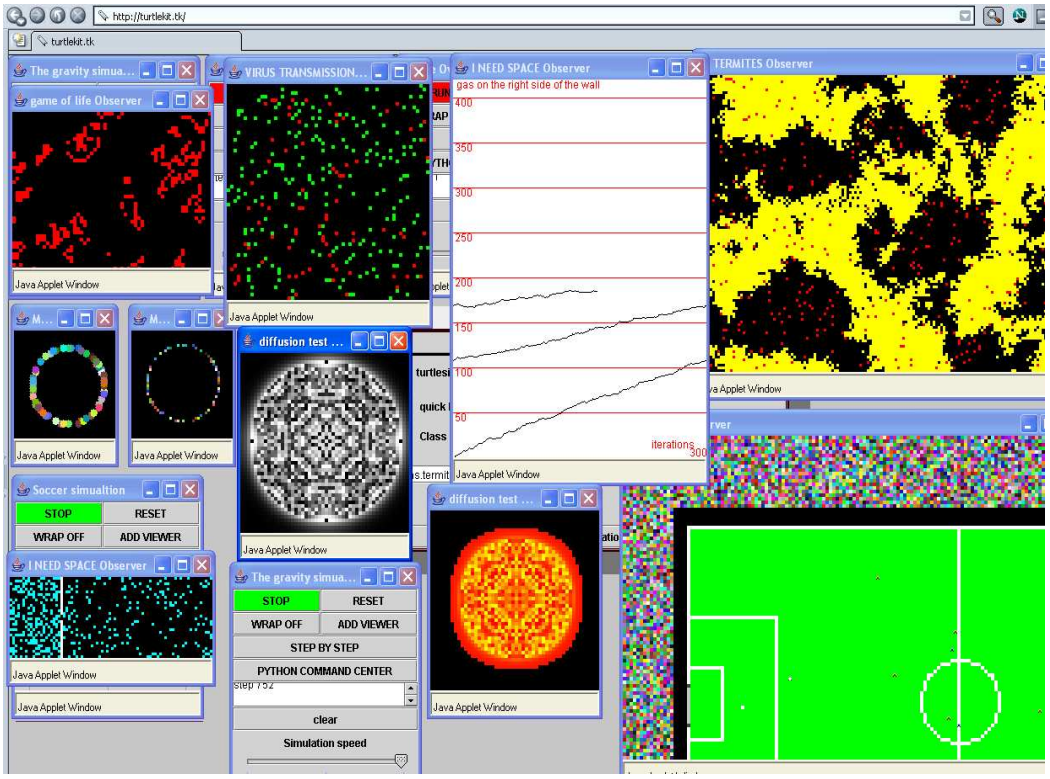

FIG. 5.6 – Exemples de simulations réalisées avec le TURTLEKIT.

Ceux que nous ne connaissons pas

Ne serait-ce que pour avoir répondu à de nombreux courriers électroniques qui concernaient soit directement le noyau de simulation, soit les applications construites au-dessus de lui, comme le TurtleKit, nous savons que de nombreuses expériences ont utilisé ces outils. Par ailleurs, la diversité des domaines abordés nous ont amené à penser que leur niveau de généricité constituait un juste milieu entre une programmation à partir de zéro et une plate-forme clé en main.

5.4 Pattern organisationnel pour la simulation

5.4.1 Apparition d'une organisation récurrente

Tout au long de notre travail de thèse, nous avons utilisé la méthodologie et les outils que nous avons présentés pour de nombreuses expériences, à la fois au travers d'implémentations test et dans le cadre de projets plus importants. L'expérience que nous avons accumulée sur ces différents cas nous a amené à constater que la plupart des simulateurs développés selon ces principes font apparaître de manière récurrente une structure organisationnelle plus ou moins identique, autrement dit un pattern d'organisation.

Nous avons donc décidé de le généraliser le plus possible afin d'obtenir une organisation d'agents qui puisse être réutilisée telle quelle. Le but est ainsi de fournir aux développeurs un peu plus que quelques classes génériques en proposant une organisation prête à l'emploi préalablement peuplée par un ensemble d'agents. Pour autant, ces agents restent toujours des coquilles vides que l'on peut étendre/remplacer selon les besoins que nécessite une simulation particulière. Cependant, certains groupes étant prédéfinis, on peut ainsi proposer un mécanisme de simulation minimal basé sur cette organisation et donc faciliter le développement et la réutilisation.

Par ailleurs, la description d'une organisation générique permet d'obtenir un niveau d'explicitation supplémentaire sur la manière dont les mécanismes du simulateur sont mis en œuvre. La structure que nous avons finalement exhibée consiste en fait dans une organisation très simple que nous allons maintenant décrire.

5.4.2 Simulateur : une machinerie et un modèle

Nous distinguons fondamentalement deux groupes de base : le premier concerne les agents qui sont chargés du fonctionnement du simulateur tandis que le deuxième regroupe plus particulièrement les agents qui participent à la dynamique du modèle : respectivement le groupe *engine* et le groupe *model*.

Le groupe *engine*

Dans le groupe *engine*, on retrouve les rôles suivants :

– le rôle *scheduler* qui identifie naturellement l'agent qui est chargé de contrôler l'exécution des agents à la fois dans le groupe *engine* et dans le groupe *model*. Comme nous l'avons dit, sa fonction consiste à gérer l'exécution globale du simulateur. Pour définir

une politique d'exécution particulière, celui-ci utilise des activateurs qui sont différents suivant les groupes d'agents considérés.
- le rôle *observer* qui désigne de manière générique les agents qui espionnent le modèle, que ce soit à des fins d'affichage, de stockage ou d'interfaçage. D'une manière générale, ce sont des agents qui n'interviennent pas dans la dynamique du modèle. Dans le mécanisme minimal que nous avons élaboré, ces agents sont par exemple exécutés à chaque fois qu'un changement d'état du système intervient.
- le rôle *environment* qui peut être joué par un ou plusieurs agents. Dans ce groupe, ce rôle est chargé de récupérer, d'intégrer et d'initialiser les agents qui participent au processus de simulation. Pour cela, l'environnement peut être un agent de type *Watcher* qui utilise une sonde (*Probe*) sur les groupes et les rôles qui sont utilisés par les agents simulés de manière à les "récupérer" automatiquement.
- le rôle *launcher* fait référence à un agent outil qui est chargé de créer l'organisation et de lancer des agents qui constituent le "boot strap" de la simulation, les agents *scheduler* et *model* par exemple. Etant donné sa fonction, cet agent peut être aussi utilisé pour lancer une batterie de simulation grâce à un script par exemple, éventuellement sur plusieurs machines différentes.
- enfin, il nous a semblé intéressant d'introduire le rôle *model* qui désigne pour nous le ou les agents qui sont dépositaires des caractéristiques du modèle et des paramètres initiaux. Ainsi, dans notre modèle minimal de simulation cet agent est chargé, d'une part de lancer les différents agents simulés qui participent à la simulation courante, et d'autre part de la définition de la dynamique globale que l'agent scheduler sera chargé d'exécuter : c'est par exemple dans cet agent que l'on peut créer les activateurs qui seront utilisés par l'agent scheduler. L'idée qui se cache derrière cet agent est en fait de centraliser et d'encapsuler l'ensemble des paramètres utilisés pour la simulation courante, qu'il s'agisse de paramètres initiaux ou de la nature des agents qui seront lancés. Ce qui permet ainsi de manipuler de nombreuses caractéristiques du modèle en modifiant uniquement cet agent. Par ailleurs, cet agent peut définir une interface graphique permettant de modifier dynamiquement ces paramètres. A ce propos, il est intéressant de voir que tous les agents peuvent avoir une interface graphique personnelle.

Le groupe *model*

Dans le groupe *model*, nous distinguons simplement les deux rôles suivants :

- le rôle *simAgent* qui identifie les agents du modèle proprement dit. Les agents qui possèdent ce rôle seront automatiquement absorbés par le processus de simulation. Autrement dit, ils seront contrôlés par le *scheduler*, intégrés par l'agent *environment* et observés par les agents qui jouent le rôle d'*observer*.
- on retrouve naturellement dans ce groupe le rôle *environment*. D'une part il fait évidemment partie du modèle du système, et d'autre part il est lui aussi contrôlé et observé selon les mêmes modalités que les autres agents. Cependant, son rôle distingué permet de lui appliquer une politique d'exécution particulière. Ce qui facilite par exemple le contrôle de sa dynamique endogène ou le déclenchement de calculs particuliers comme la résolution des conflits par exemple.

La figure 5.7 donne une illustration de l'architecture obtenue par l'application de ce pattern organisationnel.

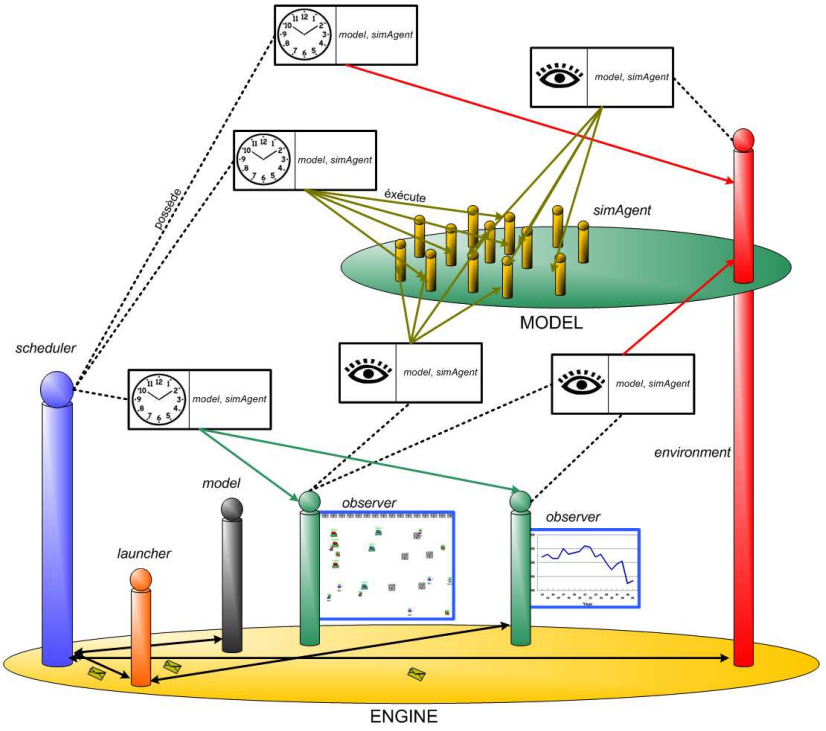

FIG. 5.7 – Pattern organisationnel pour la simulation dans MADKIT.

5.4.3 Extensions

Ce pattern peut être utilisé tel quel, cependant il peut bien sûr être étendu suivant les besoins de la simulation. Une des premières extensions possibles consiste à attribuer différents rôles aux agents de la simulation de manière à pouvoir élaborer des politiques d'exécution plus complexes et des observations plus ciblées. On pourra par exemple affecter le rôle *fourmi* à certains agents, tandis que d'autres auront le rôle de *larves*, le tout dans un groupe *fourmilière*.

Par ailleurs, il est possible de concevoir des simulations qui utilisent ce pattern de manière récursive afin d'obtenir un modèle multi-échelles par exemple. La figure 5.8 illustre cette idée.

5.4.4 Principe de simulation associé

Sur la base de ce pattern organisationnel, nous avons implémenté un noyau d'exécution minimal qui contient des instances génériques des différents agents, activateurs et sondes qui participent au processus de simulation ainsi défini, ceci de manière à fournir un mécanisme de simulation qui puisse être réutilisé sans difficulté. Le mécanisme de simulation est basé sur un principe événementiel où l'exécution d'un activateur particulier est considérée comme un événement atomique qui sera déclenché par le scheduler principal à la date fixée. Par exemple, un activateur chargé d'exécuter tout un ensemble d'agents à un instant t (DiscreteTimeActivator), c'est-à-dire suivant un principe de simulation à pas de temps constant, sera identifié dans le système comme un événement individuel et ponctuel qui se répète de façon régulière dans le temps. Le principal avantage de ce fonctionnement est de permettre à un utilisateur de mélanger différentes politiques d'exécution. Pour le scheduler les différents activateurs définissent tous des événements de même nature. Autrement dit, il est par exemple possible de faire évoluer les agents suivant un principe événementiel grâce à un premier activateur tan-

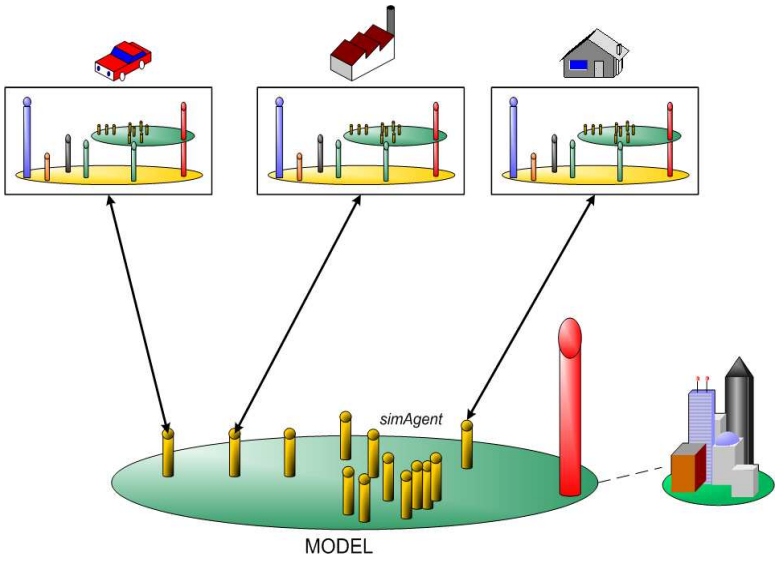

FIG. 5.8 – Utilisation récursive du pattern pour la simulation multi-échelles.

dis que l'évolution endogène de l'environnement sera modélisée par un processus définissant une discrétisation du temps régulière appliquée par un autre activateur. La partie critique du scheduler qui correspond à l'application de ce principe est la suivante :

```
/** returns true if there is no more event to execute*/
public boolean executeNextEvent()
{
        // next event in queue
        SimEvent event = (SimEvent) eventList.remove(0);
        if(event != null)
        {
                // get the source of the event
                GenActivator activator = (GenActivator) event.getSource();
                // update GVT
                setGVT(event.getDate());
                // execute next event
                SimEvent se = activator.execute(GVT);
                // add the external event produced
                if(se != null)
                        addEvent(event);
                //state has changed : do a display
                observers.execute(GVT);
                return false;
        }
        return true;
}
```

On voit ici que tous les activateurs sont traités de la même façon quelle que soit leur nature. C'est pourquoi il est donc possible de définir plusieurs types d'activateur différents et de les tester les uns après les autres, ou en même temps de façon à analyser les réactions du modèle de simulation très simplement. Le code suivant définit par exemple deux activateurs différents sur un même couple groupe/rôle qui peuvent être utilisés indépendamment ou simultanément.

```
activator1 = new DiscreteTimeActivator(community, GenModel.MODEL_GROUP, GenModel.
    SIMAGENT_ROLE, "doIt",1,1);
activator2 = new DiscreteEventActivator(community, GenModel.MODEL_GROUP, GenModel.
    SIMAGENT_ROLE, "doIt",1);
```

5.5 Résumé du chapitre

Dans ce chapitre nous avons présenté les outils logiciels qui nous ont servi de brique pour concevoir l'ensemble des applications de simulation que nous avons implémentées pendant nos recherches. Dans un premier temps ils nous ont permis de nous rendre compte par nous-mêmes des problèmes et des questions qui se posent lorsqu'on souhaite implémenter un simulateur multi-agents.

Dans un deuxième temps, il nous faut rappeler le contexte dans lequel ces outils ont été élaborés. Dans le chapitre précédent nous avons insisté sur l'importance de concevoir des outils de conception suffisamment génériques pour qu'ils puissent être réutilisés quel que soit le domaine d'application. De notre point de vue, chaque expérience de simulation est en effet singulière et nécessite un simulateur particulier qui soit adapté aux besoins. La réutilisabilité était donc un de nos objectifs majeurs. Sur ce point, la variété des simulations qui ont été implémentées à l'aide de ces outils nous a donné confiance dans notre approche. Rappelons ici qu'il s'agit de considérer que tout simulateur multi-agents doit faire face aux problèmes d'ingénierie logicielle liés à la gestion de l'exécution des agents et à l'observation du système.

Notre deuxième objectif était de faire en sorte que le fonctionnement des simulateurs élaborés à l'aide de ces outils soit facilement modifiable et explicite. A l'instar des outils d'analyse de sensibilité, l'intérêt est de permettre l'exploration des différentes implémentations qui sont possibles pour un modèle. Grâce à cette exploration, il s'agit d'une part d'identifier de possibles faiblesses dans les spécifications d'implémentations du modèle, et d'autre part de mieux comprendre l'implication des choix de programmation sur les résultats obtenus.

Finalement, ces outils ont aussi constitué pour nous le moyen d'implémenter les différentes perspectives de recherche que nous allons présenter dans la suite de ce document. Comme nous l'avons dit dans le chapitre précédent, nous pensons que le futur de la simulation multi-agents passe aussi par une réflexion approfondie sur la manière dont nous caractérisons et modélisons ces systèmes. Dans le chapitre suivant, nous allons précisément nous attaquer au problème de la modélisation de l'action dans les systèmes multi-agents. Nous allons essayer de convaincre le lecteur de l'intérêt de modifier notre point de vue sur la manière dont le résultat de la délibération d'un agent, c'est-à-dire son action sur l'environnement, doit être modélisé.

Chapitre 6

Le principe Influence/Réaction pour la simulation

Dans le chapitre 3 (section 3.5.5 page 71), nous avons vu que la simultanéité est une notion difficile à implémenter de manière satisfaisante alors que dans le même temps elle est inhérente au paradigme agent. Dans ce chapitre nous allons présenter le principe Influence/Réaction comme une solution pertinente aux problèmes posés par la représentation de la simultanéité des actions dans les systèmes multi-agents. Dans un premier temps, nous allons tout d'abord essayer de mettre en évidence le fait que la représentation classique de l'action est inadaptée à une prise en compte simple et explicite de la simultanéité. Nous verrons ensuite pourquoi le principe Influence/Réaction permet de contourner cette difficulté grâce à la modélisation de l'action sous la forme d'une influence. Nous proposerons ensuite une adaptation du modèle formel proposé par Ferber et Müller [Ferber & Müller, 1996] dans le cadre de la simulation et nous verrons comment il est possible de l'implémenter. Finalement, nous présenterons le projet *Warbot* qui concrétise quelques-unes des idées que nous allons présenter. Nous finirons par essayer de tirer les conclusions importantes de cette expérience.

6.1 Modélisation classique de l'action

6.1.1 Le problème de la porte

En guise de prélude à la présentation du principe Influence/Réaction, nous allons tout d'abord revenir un instant sur la manière dont les actions d'un agent sont généralement modélisées.

Considérons ici un système très simple composé de deux agents, numérotés 1 et 2, évoluant dans un environnement défini par deux zones, notées A et B, séparées par une porte battante. Etant donné ce système, nous allons nous intéresser à un instant particulier de la simulation où les deux agents se retrouvent exactement au même instant de chaque côté de la porte. De plus, chaque agent désire changer de pièce à cet instant précis. La figure 6.1 illustre cette situation.

Outre l'état interne des agents, si l'on considère uniquement cette portion de l'environnement, l'état du système est donné par la position des agents et par l'état de la porte, qui peut être ouverte vers A, ouverte vers B ou fermée. Le comportement d'un agent consiste dans trois actions possibles : pousser la porte si elle est fermée **ou** la franchir si la porte est ouverte vers

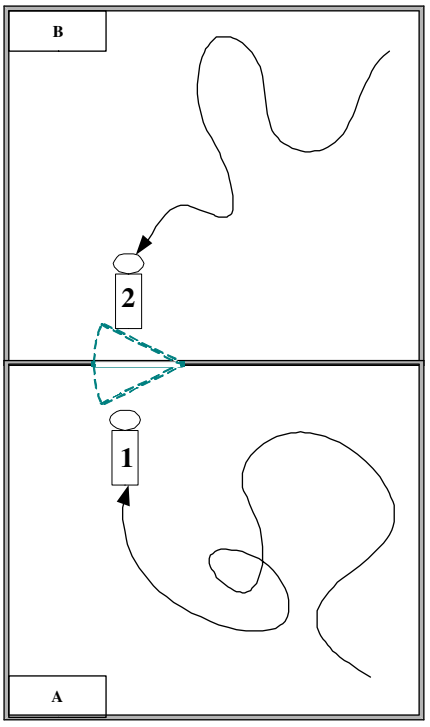

FIG. 6.1 – Le problème de la porte.

la pièce de destination (i.e. pour aller dans la pièce A la porte doit être ouverte du côté de A) **ou** enfin ne rien faire dans le cas où la porte est ouverte du mauvais côté. La perception q'un agent a de l'environnement concerne uniquement l'état courant de la porte et sa propre position.

Pour formaliser le système à cet instant de la simulation, nous allons utiliser le formalisme de [Genesereth & Nilsson, 1987] (cf. section 3.1.4 page 50). Tout d'abord, l'état global du système à un instant t de la simulation peut être décrit de la façon suivante :

- l'état de la porte est tel que :
  $\sigma_{door}(t) \in \Sigma_{door} = \{door(closed), door(A), door(B)\}$.
- la position des deux agents :
  $\sigma_n(t) \in \Sigma_n = \{A, B\}$.
- l'état global du système $\sigma_s(t) \in \Sigma_s$ est une combinaison telle que :
  $\sigma_s(t) = \sigma_{door}(t) \times \sigma_1(t) \times \sigma_2(t)$

Le comportement des agents $Behaviour_n$, ici considérés comme tropiques, correspond à l'application successive de la fonction de perception et de la fonction réflexe telles que :

- la perception d'un agent n à l'instant t est une information partielle sur l'environnement qui lui permet de faire sa décision. Ici cela équivaut à connaître sa propre position et l'état de la porte. La fonction $Percept_n : \Sigma_s \mapsto P_n$ est donc ici telle que :
  $Percept_n(\sigma_s(t)) = p_n(t) = \sigma_{door}(t) \times \sigma_n(t)$
- l'action A_n d'un agent n à un instant t découle de sa perception et appartient à l'ensemble $A = \{Push, Move, None\}$. Son résultat est défini par l'application de la fonction $Reflexe_a : P_a \mapsto A$ définie ici telle que :
  $A_n(t) = Reflexe_n(p_n(t)) = Push$ **ou** $Move$ **ou** $None$.

6.1.2 monEnvironnement.transformeToi(selonMonDesir())

Nous arrivons maintenant à la question de l'implémentation de l'action et de son résultat. Héritage de la conception de l'action dans le domaine de l'intelligence artificielle classique, l'action d'un agent est généralement modélisée comme la *transformation d'un état global* [Ferber, 1999]. Le résultat de l'action d'un agent est ainsi directement concrétisé par la transformation de l'environnement sous-tendue par cette action. Autrement dit, la **modification directe des variables d'état** de l'environnement **est** le moyen par lequel est signifié le **résultat de la délibération** d'un agent.

Dans notre cas par exemple, l'action d'un agent sera directement implémentée par une fonction $Transfo : A_n \mapsto \Sigma_s$: si un agent est activé et fait l'action de pousser la porte, l'état de la porte est immédiatement modifié de $door(closed)$ à $door(A)$ ou $door(B)$ suivant les cas. Concrètement, dans un code programmé à l'aide d'un langage objet, un agent exécute une méthode de l'environnement qui change ses variables internes.

6.1.3 Dynamique obtenue

Voyons maintenant comment se comporte la dynamique issue d'une telle implémentation dans le cas où les deux agents agissent à un même instant. Considérons un état initial $\sigma_s(0)$ où la porte est fermée et où les deux agents se trouvent chacun dans une zone. Soit plus formellement l'état initial suivant :

$$\sigma_s(0) = \sigma_{door}(0) \times \sigma_1(0) \times \sigma_2(0) = door(closed) \times A \times B \tag{6.1}$$

Traitement séquentiel

Essayons tout d'abord d'appliquer une politique à pas de temps constant simple où les agents sont activés et agissent séquentiellement. Si l'agent 1 est exécuté en premier, sa perception de l'environnement est $p_1(0) = \sigma_{door}(0) \times \sigma_1(0) = door(closed) \times A$ et son action, $Push$, change l'état de la porte en $door(B)$. Vient maintenant le tour de l'agent 2 dont la perception, et donc l'action, est effectué sur les nouvelles valeurs engendrées par l'action de 1. Autrement dit, on obtient $p_2(0) = door(B) \times B$ et son action sera donc $None$.

Comme on le voit, le fait d'avoir été exécuté en premier constitue pour l'agent 1 un avantage certain quant à l'accomplissement de son but : au prochain tour, la porte sera ouverte dans son sens. Par ailleurs, quel que soit son numéro d'activation, il pourra la franchir car si l'agent 2 agit en premier il ne pourra rien faire. Comme nous l'avons dit dans la section 3.5.3, pour éviter que l'agent 1 ne soit systématiquement avantagé dans son exploration, il est important d'exécuter les agents de façon aléatoire à chaque pas.

Cependant, nous avons vu qu'il subsiste avec cette technique un problème d'importance qui concerne la cohérence temporelle de la dynamique obtenue. Ici, ce problème est illustré par le fait qu'une des fonctions du modèle n'est pas respectée par cette implémentation. En effet, on a $Percept_n(\sigma_s(0)) = p_n(0) = \sigma_{door}(0) \times \sigma_n(0)$ alors que l'état de la porte perçu par le deuxième agent n'est pas égal à $\sigma_{door}(0)$. Autrement dit, la perception de l'agent numéro 2 à l'instant 0, $P_2(0)$, se fonde sur un état du monde qui n'est pas $\sigma_s(0)$ et qui ne fait pas partie de l'historique des états du système. Sa perception se fait donc sur un état du monde qui n'existe pas.

Traitement par résolution des conflits

Nous avons vu que pour éviter cela, le principe de simulation utilisant des variables tampons était une solution qui permet d'assurer la cohérence temporelle entre les différentes perceptions que les agents ont du monde à un instant de la simulation. Dans notre cas, il s'agit de faire en sorte que les agents ne modifient pas directement l'état de la porte de manière à ce que, dans le code, les agents utilisent la même valeur pour $\sigma_{door}(t)$. Une solution consiste donc ici à utiliser une variable $doorBuffer_n \in \{door(closed), door(A), door(B)\}$ pour stocker le résultat calculé par un agent.

A partir du même état initial que précédemment, on obtient maintenant un conflit qu'il nous faut résoudre puisque l'état final de la porte calculé par les deux agents est différent : $doorBuffer_1 = door(B) \neq door(A) = doorBuffer_2$. Comme nous l'avons dit, les méthodes basées sur une résolution de conflits posent de nombreux problèmes à la fois techniques et conceptuels. Ici, le fait de choisir entre l'un des deux résultats proposés par le calcul des agents revient à annihiler l'action de l'un d'entre eux. En effet, son comportement et son action sur l'environnement n'ont finalement eu aucune répercussion. Ce qui n'est pas très satisfaisant du point de vue de la représentation de la simultanéité.

6.1.4 Difficultés de modéliser la simultanéité

Si l'on présente le cas à quelqu'un qui n'est pas un informaticien, cette personne répondra certainement que la porte ne s'ouvre pas sous l'effet de l'action des deux agents. En effet c'est ce qui semble le plus logique et le modèle, tel que nous l'avons présenté, s'avère finalement assez flou en ce qui concerne ce problème en particulier. En fait, cela montre que pour véritablement traiter la simultanéité, il est nécessaire d'y faire explicitement allusion dans le modèle lui-même. Sans quoi on reste inévitablement dans le séquentiel dans le traitement des situations au moment de l'implémentation. Ici il nous faut donc ajouter le point de description suivant : *dans le cas où la porte est poussée par deux agents, celle-ci reste fermée.* La question est maintenant de savoir comment cette nouvelle information peut être implémentée.

Dans le cas d'une résolution de conflits, on peut remarquer qu'il est absolument impossible pour un agent de détecter à partir de sa perception que la porte restera fermée car il ne peut pas savoir ce que va faire son homologue. Du coup, on ne peut faire autrement que de raisonner sur la base d'un conflit pour détecter que deux agents ont effectué des actions simultanées qui interfèrent entre elles. Ici, déduire du conflit $doorBuffer_1 = door(B) \neq door(A) = doorBuffer_2$ que la porte reste fermée est assez simple. En effet, la combinaison de ces deux états incompatibles équivaut à savoir que deux agents ont poussé la même porte chacun de leur côté.

Ce qui est ici très simple le sera beaucoup moins à partir du moment où le système sera plus complexe. En effet, la seule chose qui nous dit que la porte reste fermée, c'est notre raisonnement d'être humain. Pour des systèmes plus complexes il sera beaucoup plus difficile de déduire a posteriori de plusieurs états incompatibles entre eux que des actions simultanées ont été effectuées et, surtout, il sera impossible d'y apporter une solution car il s'agit d'un calcul trop contraint où les problèmes d'incohérence entre les situations seront parfois inextricables.

Comme nous l'avons dit, les agents proposent des états du monde qui ne prennent pas en compte les actions des autres agents. Leur raisonnement est donc largement incomplet et aboutit nécessairement à proposer des états du monde qui seront contradictoires avec ceux qui seront proposés par les autres. Mais ce problème est tout à fait naturel. Un agent, dont la

perception est subjective, ne peut pas (ne doit pas) posséder toutes les clés qui lui permettent de réaliser un calcul exhaustif de la situation. De plus, nous avons insisté sur le fait qu'un agent n'est pas une entité capable de calculer les conséquences de ces actes. Il est alors tout à fait normal d'éprouver des difficultés à calculer le nouvel état du monde à partir de résultats qui ne prennent pas en compte l'ensemble des informations pertinentes, c'est-à-dire l'ensemble des actions.

Ce n'est donc pas aux agents de calculer ou de proposer les nouvelles valeurs des variables d'état de l'environnement. Cependant, si on revient quelques instants sur la manière dont l'action est généralement implémentée dans les systèmes multi-agents, c'est-à-dire par transformation d'un état global, on voit bien qu'il n'est pas possible de faire autrement avec cette modélisation de l'action. Telle quelle, elle équivaut en effet à faire correspondre la délibération d'un agent avec la modification des variables d'état de l'environnement. Ce que nous avons ici matérialisé par la fonction $Transfo$. Une telle fonction associe directement la délibération d'un agent et son résultat sur l'environnement, sans intermédiaire. Et c'est bien là qu'est le problème. Encore une fois, un agent ne possède pas les informations qui lui permettent de calculer les conséquences de ses actes sur l'environnement car il ne connaît ni les actions des autres agents ni la dynamique environnementale.

C'est pourquoi, si une telle fonction permet de représenter de façon satisfaisante l'action d'un agent lorsque celui-ci est seul dans l'environnement, elle s'avère finalement difficile à utiliser lorsque plusieurs agents agissent simultanément dans un environnement commun. Comme le souligne Ferber, cette représentation de l'action, ainsi que ses dérivées, ne permet de traiter la simultanéité qu'au prix de programmes complexes qui s'apparentent plus à des artifices de programmation ponctuels qu'à une véritable modélisation de la simultanéité [Ferber, 1999] :

> "*Dans le cadre des systèmes multi-agents, le problème est patent : alors que nous supposons en permanence que les actions des différents agents sont effectuées en parallèle, nous ne disposons pas de formalisme adéquat pour définir facilement des actions simultanées et donc représenter des actions collectives.*"

Dans la section 3.5.5 nous avons vu que le formalisme Parralel DEVS propose une approche novatrice de la modélisation de la simultanéité. Au lieu de considérer que la simultanéité de deux événements constitue un conflit, la fonction δ_{con} de ce formalisme permet au modélisateur de donner une véritable sémantique à ce type de situation. Il s'agit de considérer que la combinaison de deux événements simultanés constitue en fait un événement d'un type particulier que le système doit être capable de traiter en tant que tel. Pour ce faire, les événements produits à un même instant t de la simulation ne sont pas traités de façon séquentielle mais comme un tout (*bag of inputs/outputs*).

Dans le cas des systèmes multi-agents, cela revient à ne pas traiter les actions des agents séparément. Par conséquent il nous faut abandonner la représentation/programmation de l'action par transformation d'état car elle ne permet pas de conserver cette information, A_n étant immédiatement convertie par la fonction $Transfo$.

Il faut donc travailler en utilisant une étape intermédiaire de manière à collecter l'ensemble des informations nécessaires au calcul de l'état suivant, c'est-à-dire l'ensemble des actions produites à l'instant t : $\cup_{i=1}^{n} A_i(t)$. Il s'agit donc dans un premier temps de construire cette information et de la stocker pour pouvoir dans un deuxième temps décider du résultat engendré par cette situation. Le principe Influence/Réaction propose précisément de généraliser et de formaliser ce mécanisme à deux phases.

6.2 Une théorie de l'action adaptée aux agents

6.2.1 Changement de vocabulaire : *action* devient *influence*

Proposée, du point de vue multi-agents, par Ferber [Ferber, 1999][1] puis développée par Ferber et Müller dans [Ferber & Müller, 1996] (article que nous abrégerons par F-M dans la suite de ce texte), cette modélisation de l'action s'attaque directement au problème de la représentation de la simultanéité des actions dans les systèmes multi-agents.

Elle est fondée sur des principes d'influences et de réactions aux influences. Dans ce modèle, un agent est considéré comme une entité qui produit des **influences** sur son environnement et non des actions au sens que nous avons vu précédemment, c'est-à-dire par transformation directe des variables d'état de l'environnement. La différence est fondamentale. Les influences produites par un agent ne modifient pas directement l'environnement mais représentent plutôt le désir d'un agent de le voir modifié d'une certaine façon. Elles sont pour lui le moyen d'essayer de changer le cours des choses.

L'idée sous-tendue par cette approche est que le résultat effectif, sur l'environnement, de cette tentative de modification ne peut être calculé sans connaître l'ensemble des influences produites au même instant. En effet, comme nous l'avons déjà noté, le résultat de l'action d'un agent dépend des autres actions qui sont produites simultanément. Notamment, du fait des autres actions, le but d'une action peut être contrarié ou aboutir à un résultat imprévu. Un accident de la route en est le parfait exemple : en général, les deux conducteurs n'ont pas l'intention de créer un accident mais ils prennent, chacun de leur côté, des décisions qui aboutissent à cet accident.

Ainsi, en ne calculant pas directement le résultat d'une prise de décision et en la modélisant sous la forme d'une influence, le but de cette démarche est de bien distinguer les gestes produits par les agents, **les influences**, de ce qui se passe effectivement compte tenu des autres gestes, c'est-à-dire **la réaction** de l'environnement à l'ensemble de ces influences. Pour calculer cette réaction, les influences sont considérées en fonction de ce que F-M appelle les *lois de l'univers*. C'est dans ces lois qu'est modélisé le traitement de la simultanéité. La figure 6.2 illustre ce principe.

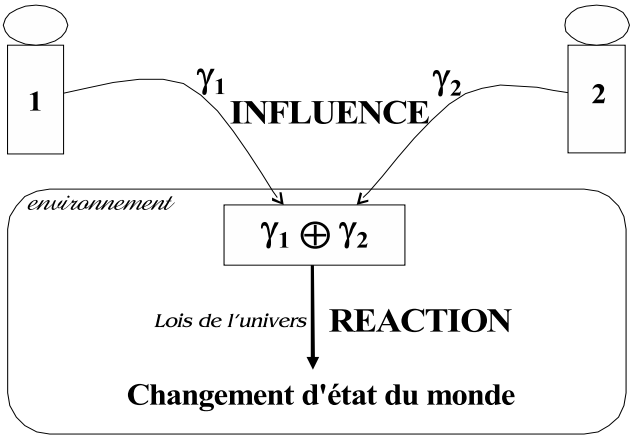

FIG. 6.2 – Le principe Influence/Réaction.

[1] La version française de cette référence date de 1995.

6.2.2 Changement de point de vue : simultanéité n'est pas conflit

Il est très important de bien comprendre que cette approche n'a rien à voir avec les techniques de résolution de conflits, même lorsque celles-ci intègrent explicitement la simultanéité à travers un raisonnement poussé comme celui que propose Travers par exemple (cf. section 3.5.3 page 66). L'idée est plutôt que les influences produites par les agents ne sont jamais contradictoires. Elles sont toujours valides. Il convient simplement de trouver la résultante de leur combinaison.

Par exemple, dans une logique de résolution de conflits telle que nous l'avons présentée pour le problème de la porte, le calcul de l'état final du système consiste à choisir entre deux états incompatibles : la porte est ouverte du côté de l'agent 1 ou de l'agent 2. Au contraire, si l'on considère que ces actions sont uniquement des influences, il devient naturel de pouvoir envisager une troisième solution pour le nouvel état du monde qui est que la porte ne s'ouvre pas sous l'effet de ces deux actions.

Ainsi, l'idée majeure du principe Influence/Réaction est de se concentrer sur la modélisation des interactions qui découlent du collectif, au lieu de considérer les entités de manière individuelle comme dans les approches classiques. Autrement dit, ce principe s'attaque véritablement à décrire l'aspect *multi* des systèmes multi-agents. C'est pourquoi nous considérons que son application constitue non seulement un élément capital et incontournable dans le domaine des simulations multi-agents mais aussi, plus fondamentalement, que ce principe est intimement lié au paradigme multi-agents et qu'il doit être à la base des approches de conception qui seront utilisées pour les plates-formes multi-agents du futur. Nous aurons l'occasion de justifier plus avant cette position dans le prochain chapitre. Nous verrons notamment que ce principe ne fait pas que résoudre le problème de la simultanéité mais qu'il constitue un moyen de concevoir des systèmes multi-agents qui respectent et intègrent clairement les contraintes intrinsèquement imposées par le paradigme.

6.2.3 Un principe, des modèles

Le lecteur aura remarqué que nous utilisons le terme *principe* et non *modèle*. Cette dénomination est volontairement utilisée pour bien distinguer l'approche Influence/Réaction de son application. Encore une fois, le principe est de considérer que les agents n'ont pas un contrôle direct sur le résultat que provoquent leurs actions et que l'ensemble des influences produites à un même instant doit être connu pour pouvoir calculer le nouvel état du monde en étudiant la combinaison de celles-ci.

De ce principe général, plusieurs modèles peuvent être élaborés pour permettre sa mise en œuvre. On peut notamment citer les travaux de Dàvila et Tucci, qui ont élaboré une plate-forme de simulation, GALATEA, basée sur un langage de programmation inspiré par le modèle de F-M [Dàvila & Tucci, 2000, Dàvila & Uzcágegui, 2000]. Les travaux de Weyns et Holvoet proposent aussi un modèle générique de systèmes multi-agents situés qui prend explicitement en compte les actions simultanées [Weyns & Holvoet, 2003b, Weyns & Holvoet, 2003a].

Chacun de ces travaux vise à améliorer/compléter le modèle de F-M en y apportant des modifications qui ont pour objet de lever certaines ambiguïtés et de permettre l'implémentation directe des modèles obtenus. Par exemple, outre la définition de quelques équations supplémentaires, Dàvila et Tucci ont judicieusement rajouté une variable temporelle dans le modèle de F-M, comme nous l'avions d'ailleurs aussi proposé dans [Michel, 2001].

Cependant, les modèles proposés par ces travaux ont tendance à complexifier l'application du principe Influence/Réaction car ils sont très influencés par l'implémentation sous-jacente envisagée. Ce qui a pour effet de les rendre difficilement généralisables.

C'est pourquoi nous allons tout d'abord présenter les grandes lignes du modèle de F-M tel qu'il a été proposé à l'origine pour ensuite en proposer une adaptation pour la simulation qui intègre les différentes analyses qui ont été faites sur ce modèle, tout en restant à un niveau d'abstraction suffisamment élevé de manière à ne pas lui associer une implémentation particulière.

6.3 Le modèle Influence/Réaction de Ferber & Müller

6.3.1 Notion d'état dynamique

Pour modéliser la simultanéité grâce au principe Influence/Réaction, F-M propose une extension du formalisme de Genesereth et Nilsson [Genesereth & Nilsson, 1987] (cf. section 3.1.4 page 50). F-M introduit ainsi la notion d'*état dynamique* pour prendre en compte à la fois l'état de l'environnement et les influences produites par le comportement des agents.

Un état dynamique, $\delta \in \Delta$, est donc une paire notée $< \sigma, \gamma >$ dans laquelle $\sigma \in \Sigma$ représente les variables d'état de l'environnement et $\gamma \in \Gamma$ décrit l'ensemble des influences produites par les agents et l'environnement sur le système. C'est ce dernier ensemble qui donne son caractère dynamique à δ. En effet, alors que de manière classique on utilise uniquement des variables d'état pour décrire le système à un instant t, l'ensemble γ est ici ajouté pour matérialiser les différentes **tendances d'évolution** auxquelles l'ensemble du système est soumis, c'est-à-dire les **influences**. Dans le cas du problème de la porte, on pourra par exemple considérer l'état dynamique suivant : $\delta =< door(closed), \gamma_1(push) >$ qui exprime le fait que la porte est fermée et que l'agent 1 tente de l'ouvrir.

6.3.2 Principe d'évolution d'un état dynamique

Calculer l'évolution au cours du temps d'un système ainsi défini consiste à élaborer une fonction $F : \Delta \mapsto \Delta$ qui, à partir d'un état dynamique δ, calcule la transformation du système dans un nouvel état dynamique δ'. Cette transformation s'effectue en deux phases. La première consiste à collecter les différentes influences et la deuxième concerne le calcul de la réaction de l'environnement à ces influences. Dans le modèle proposé par F-M, cette fonction est donc décomposée en deux fonctions : $Exec : \Sigma \times \Gamma \mapsto \Gamma$ calcule les influences qui sont produites pour le nouvel état dynamique, $React : \Sigma \times \Gamma \mapsto \Sigma$ calcule les nouvelles variables d'état de l'environnement[2] de telle manière qu'un état dynamique $\delta =< \sigma, \gamma >$ est transformé en $\delta' =< \sigma', \gamma' >$ avec :

$$\sigma' = Reac(\sigma, \gamma) \tag{6.2}$$

$$\gamma' = Exec(\sigma', \gamma) \tag{6.3}$$

[2]Nous ne donnons pas ici le détail de ces deux fonctions pour des raisons de simplicité. F-M introduit en effet la notion d'opérateurs (*operators*) et de lois (*laws*) pour décrire de manière générale les fonctions qui, respectivement, définissent la façon dont les influences sont produites à partir d'un état du monde et décrivent la manière dont les nouvelles variables d'état sont calculées à partir d'un ensemble d'influences et des variables d'état précédentes.

Là-dessus, F-M propose de définir l'évolution de l'ensemble du système comme une fonction récursive infinie, appelée $Evolution : \Sigma \times \Gamma \mapsto \tau$ qui consiste dans l'application cyclique de ces deux fonctions[3] :

$$Evolution(\sigma, \gamma) = Evolution(Cycle(\sigma, \gamma)) \tag{6.4}$$

où $Cycle$ est définie telle que :

$$\begin{aligned} &Cycle : \Sigma \times \Gamma \mapsto \Sigma \times \Gamma \\ &Cycle(\sigma, \gamma) =< \sigma', Exec(\sigma', \gamma) > \end{aligned} \tag{6.5}$$

avec $\sigma' = Reac(\sigma, \gamma)$, soit finalement :

$$Cycle(\sigma, \gamma) =< Reac(\sigma, \gamma), Exec(Reac(\sigma, \gamma), \gamma) > \tag{6.6}$$

6.3.3 Modélisation du comportement d'un agent

Différence majeure avec le formalisme originel, la délibération d'un agent n'est plus associée à une transformation de l'état global. Le cycle classique perception/délibération/action d'un agent aboutit maintenant à la production d'une influence $\gamma \in \Gamma$. Pour un agent tropique par exemple, la fonction $Reflexe_a$ est maintenant définie telle que $Reflexe_a : P_a \mapsto \Gamma$.

Outre ce changement de point de vue sur la manière dont le résultat de la délibération est considéré, F-M introduit par ailleurs une modification qui concerne le calcul de la perception d'un agent. F-M propose de considérer que les agents sont des d'entités qui sont influencées par leur environnement et produisent, en réaction, des influences sur celui-ci. Ainsi, alors que de manière classique la perception d'un agent est calculée comme une partition des variables environnementales de Σ, c'est-à-dire par une fonction telle que $Perception_a : \Sigma \mapsto P_a$, dans le modèle de F-M il est considéré que la perception d'un agent est une fonction qui calcule un percept à partir de l'ensemble des influences présentes : $Perception_a : \Gamma \mapsto P_a$. Les auteurs de F-M justifient cette vision des choses en considérant qu'ils préservent ainsi le principe de séparation entre les influences et la réaction et que ce modèle inclut de facto la localité de la perception :

> "*Agents perceive what influences them and are not influenced by the whole state of the environment.*"

Nous allons maintenant présenter une adaptation du modèle de F-M pour la simulation.

6.4 Un modèle Influence/Réaction pour la simulation

6.4.1 Introduction d'une variable temporelle explicite

La différence majeure entre le modèle de F-M et celui que nous allons maintenant présenter tient dans l'utilisation d'une variable temporelle dans les différentes équations. En effet, bien que la logique de l'évolution temporelle du système soit implicitement présente dans F-M, l'introduction explicite du temps dans le modèle permet de clarifier son application qui,

[3] Cette fonction prend en argument un état dynamique mais ne retourne pas de résultat du fait de sa boucle infinie. τ exprime cette particularité en désignant un domaine de valeur qui ne contient que des erreurs ou les états du monde qui sont impossibles

comme nous le verrons, est pour l'instant ambiguë dans certains cas. De plus, dans le cadre de la simulation, la modélisation du temps (et donc l'utilisation d'une variable temporelle) est incontournable et il est important de pouvoir dater les états du système. Les autres modifications importantes que nous allons proposer sont pour la plupart liées à l'utilisation d'une telle variable. Cependant, nous allons voir maintenant qu'il existe aussi une différence notable sur la manière dont la perception qu'un agent a de son environnement est calculée.

6.4.2 Modification de la fonction *Perception*

Bien que nous soyons tout à fait d'accord avec la précédente citation, la représentation de la fonction *Perception* telle qu'elle est énoncée dans F-M se révèle en fait très contraignante en ce qui concerne sa mise en œuvre dans le modèle. Considérer que ce qui *influence* le comportement d'un agent doit être uniquement modélisé par une influence $\gamma \in \Gamma$ entraîne une confusion entre ce qu'est une influence dans ce modèle, c'est-à-dire l'expression d'une tendance d'évolution, et ce qui *influence* le comportement d'un agent, c'est-à-dire sa manière de percevoir le monde. En effet, il y a une différence entre le fait de représenter qu'un agent n'a pas une perception directe de son environnement, car celle-ci est locale, subjective, altérée par ses capteurs, et le fait de la modéliser par une influence $\gamma \in \Gamma$.

Dàvila et Tucci ont fait d'ailleurs sur ce point la même analyse que nous et ils ont souligné le fait qu'il est assez inconfortable, pour représenter la perception, de considérer qu'un agent ne puisse pas percevoir des variables d'état environnementales statiques comme le fait qu'une porte soit fermée par exemple. C'est pourquoi ils ont modifié le modèle de F-M en proposant la fonction $Perception : \Sigma \times \Gamma \mapsto P_a$ que nous utiliserons nous aussi. Alors que dans le modèle de F-M le comportement d'un agent est représenté par une fonction $Behaviour_a : \Gamma \mapsto \Gamma$, nous représenterons donc le comportement d'un agent par la fonction suivante :

$$Behaviour_a : \Sigma \times \Gamma \mapsto \Gamma \tag{6.7}$$

Il est important de comprendre que cette modification n'est pas en désaccord avec l'esprit du modèle Influence/Réaction et qu'elle ne contredit pas la précédente citation. Au contraire, elle met en avant le rôle fondamental que la notion de perception doit jouer lorsque l'on modélise un système multi-agents situé : ici, c'est précisément le rôle de la fonction *Perception* de calculer, à partir d'un état dynamique du monde $\delta(t) =< \sigma(t), \gamma(t) >$, le percept $p_a(t) \in P_a$ que l'agent va finalement percevoir. En quelque sorte, elle joue le rôle d'un filtre. Elle peut donc inclure les notions de localité, subjectivité, etc.

Par ailleurs, un point très important est que cette fonction de filtrage n'est en principe pas du ressort de l'agent dans le sens où le code correspondant à ce traitement ne doit pas être confondu avec celui qui représente le comportement interne de l'agent. Il s'agit de faire en sorte que celui-ci n'ait jamais un accès direct aux variables d'état de l'environnement. Autrement dit, bien que la perception soit un processus conceptuellement lié à un individu en particulier, le calcul des données de perception doit être dissocié de ses fonctions comportementales. Il semble alors très judicieux d'associer la perception à une structure de données entièrement fournie par l'environnement. De cette manière un agent n'a pas à savoir de quelle manière l'environnement est implémenté et peu importe qu'il soit discret ou continu, en deux ou trois dimensions. Une perception peut ainsi consister dans une simple demande de l'agent à laquelle l'environnement sait quoi et comment répondre. L'environnement peut par exemple calculer la perception d'un agent en fonction de son type et ainsi lui fournir une structure de données qui possède toutes les informations nécessaires à sa délibération. La figure 6.3 illustre ce mécanisme.

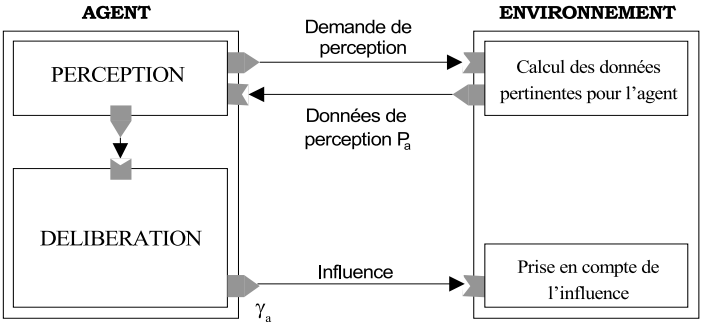

FIG. 6.3 – Construction de la perception d'un agent par l'environnement.

On retrouve cette idée dans la plupart des travaux qui traitent de la relation entre un agent et son environnement. Parmi les travaux que nous avons déjà cités, Soulié utilise par exemple dans son modèle de système multi-agents situé une entité appelée *transcodeur de perceptions* qui sert à transformer les données environnementales *de bas niveau* ($\sigma \in \Sigma$ dans notre formalisme) en données de perception de *haut niveau* qui seront ainsi intelligibles pour le système conatif de l'agent ($p_a \in P_a$) [Soulié, 2001].

6.4.3 Modification de la fonction *Evolution*

La modification du modèle originel que nous venons de faire nous oblige cependant à revoir la définition des fonctions *Exec* et *Reac* de manière à ne pas avoir d'incohérences temporelles. En effet, soit un état dynamique $\delta(t) =< \sigma(t), \gamma(t) >$, l'état dynamique suivant $\delta(t+dt) =< \sigma(t+dt), \gamma(t+dt) >$ est pour l'instant calculé de la manière suivante :

$$\begin{aligned} \sigma(t+dt) &= Reac(\sigma(t), \gamma(t)) \\ \gamma(t+dt) &= Exec(\sigma(t+dt)) \cup \bigcup_i Behaviour_i(\sigma(t), \gamma(t)) \end{aligned} \tag{6.8}$$

Ainsi, alors que les influences produites par le système sont calculées à partir de $\sigma(t+dt)$ ($Exec\ (\sigma(t+dt))$), les influences produites par les agents sont calculées en partie à partir de $\sigma(t+dt)$. On se retrouve donc avec un ensemble d'influences $\gamma(t+dt)$ que l'on peut considérer comme incohérent étant donné qu'il est calculé à partir de variables d'état n'appartenant pas à un seul et même état dynamique.

Face au même problème, Dàvila et Tucci ont préféré abandonner la fonction *Exec* et redéfinir *Reac* de manière à lui faire jouer le rôle des deux fonctions pour un instant t (son application est instantanée d'un point de vue temporel). En ce qui nous concerne, pour lever cette ambiguïté nous préférons ici redéfinir deux nouvelles fonctions plus adéquates qui permettent de clarifier l'application du principe Influence/Réaction d'un point de vue temporel. Pour cela nous allons reconsidérer la décomposition de la fonction *Evolution* en deux nouvelles fonctions, *Influence* et *Reaction*, définies de la façon suivante :

$$Influence : \Sigma \times \Gamma \mapsto \Gamma' \tag{6.9}$$

$$Reaction : \Sigma \times \Gamma' \mapsto \Sigma \times \Gamma \tag{6.10}$$

La fonction *Influence* définit de façon générale le nouvel ensemble d'influences, $\gamma'(t) \in \Gamma'$, obtenu étant donné l'état dynamique $\delta(t)$ du système à l'instant t. Outre les influences déjà présentes dans le système à l'instant t, cet ensemble contient les influences qui sont produites par les agents à l'instant t mais aussi celles qui sont générées par l'environnement lui-même pour signifier son évolution endogène. La fonction *Reaction* définit quant à elle la manière dont le monde se transforme pour donner un nouvel état dynamique, $\delta(t + dt)$, étant donnés l'ensemble des variables d'état du système et l'ensemble d'influences obtenu par l'application de la fonction *Influence*. L'évolution du monde, d'un état dynamique à un autre, correspond ainsi à un mécanisme à deux phases qui consiste dans l'application de la fonction *Influence* **puis** de la fonction *Reaction*. Un état dynamique $\delta(t) =< \sigma(t), \gamma(t) >$ évolue donc dans un nouvel état dynamique $\delta(t+dt) =< \sigma(t+dt), \gamma(t+dt) >$ en appliquant les équations suivantes :

$$\gamma'(t) = \mathit{Influence}(\sigma(t), \gamma(t)) \tag{6.11}$$

$$< \sigma(t + dt), \gamma(t + dt) >= \mathit{Reaction}(\sigma(t), \gamma'(t)) \tag{6.12}$$

Une autre différence que nous avons avec le modèle de F-M est que nous considérons que l'application de ces deux fonctions est instantanée dans le temps. Nous rejoignons ainsi le point de vue de Dàvila et Tucci. Autrement dit, entre deux états dynamiques distincts, l'état du système est indéfini. La raison en est simple : si l'on considère que l'état du système est cohérent entre l'application de ces deux fonctions, comme c'est le cas dans le modèle de F-M où l'état du système est à tout instant défini par la paire $< \sigma, \gamma >$ quel que soit le moment du cycle, cela veut dire qu'il existe des instants t de la simulation qui n'ont pas la même sémantique : certains sont liés au calcul de la réaction tandis que d'autres sont réservés à la production d'influences. Il devient alors difficile, voire impossible, d'établir une échelle temporelle cohérente et la dynamique du système dans son ensemble n'est pas intuitive. Ce point précis n'était pas abordé dans F-M car le modèle proposé était essentiellement descriptif et son application n'avait pas pour vocation la simulation[4]. En ce qui nous concerne la modélisation du temps est un point essentiel de notre problématique et il est important de pouvoir définir de manière non ambiguë d'une part, les instants pour lesquels le système est défini (c'est-à-dire les instants auxquels correspond une date t) et d'autre part, les mécanismes de changement d'état du système d'un instant t à l'instant suivant $t + dt$. Cette différence est matérialisée par le fait que l'ensemble d'influences $\gamma'(t)$ calculé par l'équation 6.11 est temporaire et qu'il va être directement utilisé par la fonction *Reaction* pour calculer le nouvel état dynamique. En effet, c'est précisément le rôle de la fonction *Reaction* de déterminer quel est le nouvel état dynamique pour l'instant suivant : c'est elle qui modélise les *lois de l'univers*. Par exemple, le système décisionnel d'un robot peut décider de tenter une action, et ainsi produire une influence (phase influence), qui peut n'avoir aucune incidence sur le monde si le robot est endommagé (phase réaction) : cette influence ne fera donc pas partie de l'ensemble $\gamma(t + dt)$ après le calcul de la réaction.

6.4.4 Décomposition de la phase influence

Pour préciser la façon dont l'ensemble des influences $\gamma'(t)$ est produit, on peut décomposer l'équation 6.11 de manière à représenter les influences produites par l'environnement et les influences générées par le comportement interne des agents. Pour prendre en compte les influences $\gamma'_w(t)$ issues de l'évolution naturelle du monde, nous utiliserons une fonction

[4]Il existe cependant une ambiguïté liée à ce problème dans le modèle de F-M dans le cas des agents hystérétiques qui avait été notée par les auteurs.

$Natural_w : \Sigma \times \Gamma \mapsto \Gamma'$ qui décrit de manière générale toutes les influences qui ne sont pas produites par les agents : objets en mouvement, évaporation d'une phéromone, etc. Comme nous l'avons dit, les influences produites par les agents sont le résultat d'une fonction $Behaviour_a : \Sigma \times \Gamma \mapsto \Gamma'$. Cependant, il est possible de décomposer cette fonction à la manière de F-M pour décrire plus avant le fonctionnement du comportement interne d'un agent dans ce modèle. Ainsi, la fonction *Behaviour* peut-être décomposée de la manière suivante :

Tout d'abord, un agent a une perception $p_a \in P_a$ de son environnement calculée par la fonction *Perception* :

$$Perception_a : \Sigma \times \Gamma \mapsto P_a \tag{6.13}$$

Ensuite, il compose cette perception p_a avec son état interne $s_a \in S_a$ pour calculer son nouvel état interne. C'est la fonction *Memorization* :

$$Memorization_a : P_a \times S_a \mapsto S_a \tag{6.14}$$

Pour finir, son nouvel état interne est l'argument de la fonction *Decision* qui détermine l'influence γ_a qui est produite par l'agent :

$$Decision_a : S_a \mapsto \Gamma' \tag{6.15}$$

Il faut noter que cette décomposition correspond au fonctionnement d'un agent hystérétique. Dans le cas d'un agent tropique, les deux dernières fonctions se résument en une seule application $Reflexe_a : P_a \mapsto \Gamma'$.

Finalement, d'un point de vue temporel, l'équation 6.11, qui correspond à l'application de la phase Influence, consiste dans l'application des fonctions suivantes :

$$\gamma'_w(t) = Natural_w(\sigma(t), \gamma(t)) \tag{6.16}$$

$$p_a(t) = Perception_a(\sigma(t), \gamma(t)) \tag{6.17}$$

$$s_a(t + dt) = Memorization_a(p_a(t), s_a(t)) \tag{6.18}$$

$$\gamma'_a(t) = Decision_a(s_a(t + dt)) \tag{6.19}$$

Avant d'aller plus loin, il convient maintenant de préciser quelle est la nature des variables qui modélisent l'état interne $s_a \in S_a$ d'un agent.

6.4.5 Distinction esprit/corps

Dans le troisième chapitre, nous avons vu que dans les travaux qui s'intéressent à la modélisation d'agents situés dans un environnement, l'importance d'une séparation claire entre les variables relatives au système conatif et les variables qui modélisent la partie physique d'un agent est de plus en plus souvent soulignée (cf. section 3.4.2 page 62) [Magnin, 1996, Soulié, 2001]. Magnin divise explicitement un agent en deux parties : cerveau et composante physique. Soulié fait lui aussi une distinction nette entre le système conatif de l'agent et les variables environnementales en formalisant ce qu'il appelle un lien bidirectionnel (pour la perception et l'action). En fait, on retrouve régulièrement cette distinction dans la littérature. Récemment par exemple, dans [Chang *et al.* , 2005] les auteurs utilisent cette distinction pour mettre en avant la différence qui existe entre le modèle qu'un agent a de son environnement, qu'il

construit par perception, et le modèle qui définit la réalité de l'environnement. Ils proposent ainsi de prendre en compte plusieurs modélisations suivant le point de vue considéré, celui de l'agent ou celui de la réalité : *mind model* et *reality model*. De plus, pour faire le pont entre les deux les auteurs définissent un troisième niveau de modélisation, *concept model*, qui permet aux agents de comprendre la réalité et d'en construire un modèle (le *mind model*). On retrouve ici l'idée que nous avions présentée dans la section 6.4.2 et selon laquelle il est intéressant de définir les données de perception en fonction de la nature des agents, et non en les construisant à partir des variables d'états environnementales brutes. Par ailleurs, outre l'intérêt pratique de ce troisième niveau, celui-ci matérialise bien l'idée qu'un agent n'a qu'une connaissance subjective de la réalité qui l'entoure. Il ne connaît pas les choses en soi. Autre exemple récent, dans [Okuyama *et al.* , 2005] les auteurs proposent un langage de description d'environnement pour les simulations multi-agents, ELMS (Environment description Language for Multi-agent Simulation) qui définit une syntaxe qui implique de définir explicitement non seulement l'environnement mais aussi le corps des agents, les perceptions et les actions. Les auteurs nous rappellent aussi que, du point de vue d'un agent, tous les autres agents font partie de l'environnement.

Résumons ici brièvement les avantages d'une telle décomposition. Premièrement, d'un point de vue génie logiciel, il est beaucoup plus simple de modifier ou de substituer les différents éléments d'une simulation (agents et/ou environnement) si l'action, la perception, le comportement interne d'un agent et l'environnement sont implémentés de façon modulaire. Deuxièmement, et c'est le plus important pour nous, d'un point de vue conceptuel il est très important de faire une différence explicite entre les variables (et les fonctions) sur lesquelles l'agent a un contrôle total, c'est-à-dire celles qui sont utilisées par le système décisionnel, et les variables qui modélisent la partie physique de l'agent et sur lesquelles celui-ci ne doit avoir aucun pouvoir de modification. La figure 6.4 illustre cette distinction.

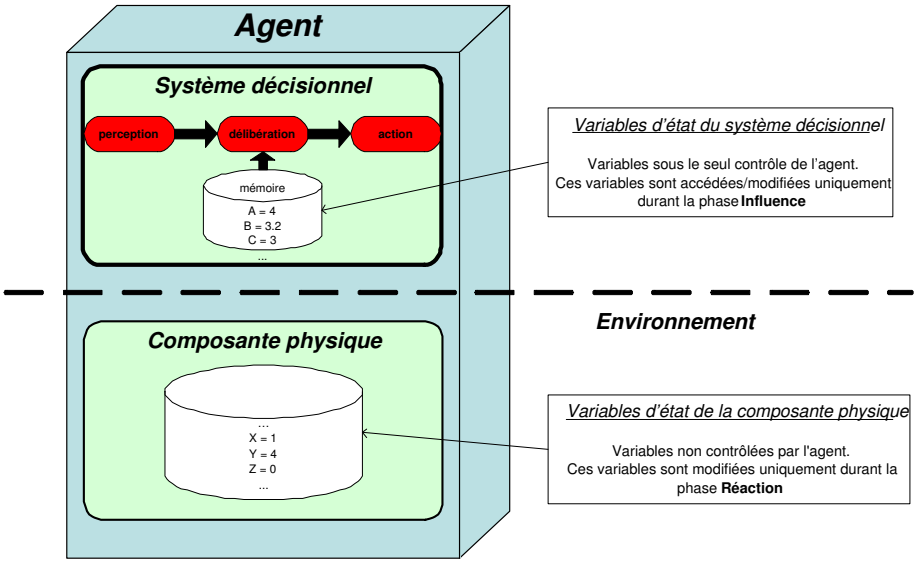

FIG. 6.4 – Distinction esprit/corps dans le contexte du principe Influence/Réaction.

Dans le contexte du principe Influence/Réaction, cette distinction est donc tout à fait fondamentale. En effet, le physique d'un agent faisant partie intégrante de l'environnement, il doit être modifié lors de la phase de réaction. La position ou les points de vie d'un agent sont des exemples représentatifs de variables appartenant au domaine physique. Un agent mobile verra sa position changer uniquement si son mouvement n'a pas été contrarié. Un agent

désireux de manger, ne verra son niveau de vie augmenté que s'il a effectivement mangé et qu'aucune autre action ne l'en a empêché. A l'opposé, l'état interne d'un agent représente quant à lui l'ensemble des variables modifiées et utilisées par le système décisionnel, respectivement lors des processus de *Memorization* et de *Decision*. Typiquement, des variables liées à un mécanisme de rétroaction positive font partie de l'état interne. Nous formalisons ici ces deux facettes de l'agent de la façon suivante :

- état interne : $s_a \in S_a$ où S_a décrit l'ensemble des états internes possibles pour un agent.
- état physique : $\phi_a \in \Phi_a$ avec $\Phi_a \in \Sigma$

Etant donné que $\Phi_a \in \Sigma$, on voit bien ici que la perception d'un agent concerne autant les variables de l'environnement que son propre état physique ou celui des autres, sachant que la perception dérive en partie de Σ. Par ailleurs, on voit que la notation utilisée pour décrire l'état global du système doit être augmentée de manière à signifier que S_a est un ensemble particulier distingué de Σ et de Γ. En fait, il s'agit de spécifier le fait que les variables d'état qui appartiennent à l'architecture interne d'un agent ne sont pas modifiées pendant la réaction et que celles-ci sont uniquement sous le contrôle de l'agent : elles ne sont ni dans Σ ni dans Γ. Ainsi, si l'on note $\omega(t) \in \Omega$ l'état total du système à l'instant t, celui-ci est en fait une combinaison telle que $\omega(t) =< \cup_{i=1}^{n} s_i(t) \times \delta(t) >$. Cependant, afin de ne pas alourdir la notation nous ne mentionnerons pas explicitement l'état total du système Ω dans nos futures équations. Il suffit de garder à l'esprit que S_a n'appartient pas à Δ et que l'état interne d'un agent est uniquement modifié durant la phase influence.

6.4.6 Calcul de la Réaction

La réaction : point-clé de la modélisation du système

Point-clé du principe Influence/Réaction et de notre modèle de simulation, le calcul de la réaction est aussi le plus délicat. De par la nature des systèmes considérés, ce calcul est en effet loin d'être immédiat. Les modèles que l'on souhaite simuler à l'aide du paradigme agent proviennent de domaines extrêmement divers (robotique mobile, système sociaux, écologie, etc). Les interactions peuvent donc être très hétérogènes : mouvement, actes de langages, reproduction, interactions de type théorie des jeux, etc. Alors que dans les simulations classiques on exprime une action par une modification directe de l'état du monde, ici on considère au contraire que le calcul du nouvel état dynamique doit aussi prendre en compte toutes les influences produites. Plus formellement, rappelons ici que la fonction $Reaction : \Sigma \times \Gamma' \mapsto \Delta$ est telle que :

$$\delta(t + dt) = Reaction(\sigma(t), \gamma'(t)) \tag{6.20}$$

Dans la section 6.4.4 nous avons décomposé dans le détail les différentes fonctions qui permettent de formaliser la phase influence. En ce qui concerne la réaction, on ne peut pas en faire autant. En effet, c'est par ce calcul que le modélisateur définit véritablement la dynamique du système dans sa globalité. Il lui appartient donc d'apporter sa propre réponse à cette équation en définissant *les lois de l'univers* qui conviennent le mieux pour décrire le système étudié. Il est donc difficile de faire dire plus de choses à l'équation précédente sans entrer dans la description d'un système en particulier et sans aborder le problème de sa modélisation proprement dit. Ce que nous réservons pour les chapitres suivants. Cependant, on peut tout de même remarquer que ce calcul va poser un problème récurrent, celui de sa complexité. En premier lieu se pose notamment la question de la combinatoire explosive des influences.

En effet, le nombre des configurations possibles pour un ensemble d'influences à un même instant augmente de façon exponentielle avec le nombre d'agents. Ainsi le calcul de la réaction nécessite une analyse fine de la composition des influences afin de proposer des *lois de l'univers* qui soient calculables et qui correspondent à ce que l'on souhaite modéliser. Pour réduire cette complexité, deux orientations sont a priori possibles : distribuer le calcul de la réaction et/ou établir une classification des influences.

La réaction : un calcul naturellement distribué

Pour simplifier le calcul de la réaction, une première approche intéressante est de prendre en compte le caractère local d'une influence. Toutes les influences n'ont pas un impact sur tout l'environnement. Il semble que ce soit plutôt le contraire en ce qui concerne les agents : en général ils n'ont le pouvoir d'influencer que leur entourage immédiat. Par exemple la consommation d'une ressource par un agent n'a pas besoin d'être prise en compte au niveau global. Seules les influences des autres agents qui souhaitent, eux aussi, consommer cette ressource ont un impact sur le calcul du nouvel état de la ressource. De la même manière le mouvement d'un agent n'affecte dans ces conséquences que la zone concernée par ce déplacement. D'ailleurs, cette problématique n'étant pas liée directement au principe Influence/Réaction, on retrouve cette idée dans des travaux comme [Theodoropoulos & Logan, 1999] où les auteurs définissent un modèle de simulation basé sur le concept de *sphères d'influence* (*spheres of influence*) de manière à caractériser la portée d'une action. Dans le contexte du principe Influence/Réaction, les récents travaux de Weyns et Holvoet adoptent aussi ce type d'approche et utilisent un algorithme basé sur la notion de *synchronisation régionale* (*regional synchronization*) qui prend en compte le rayon de perception des agents pour agréger puis combiner les influences en fonction de leur localisation [Weyns & Holvoet, 2003b, Weyns & Holvoet, 2003a].

Ainsi, il est clair que la réaction ne doit pas être un calcul centralisé qui intègre globalement l'ensemble des influences produites. Au contraire, celui-ci doit être distribué suivant des zones définies par la portée des influences. De cette manière on décentralise la prise en compte des influences et on distribue de façon logique et fonctionnelle le calcul de la réaction. Il s'agit de diviser pour régner. La figure 6.5 illustre cette idée.

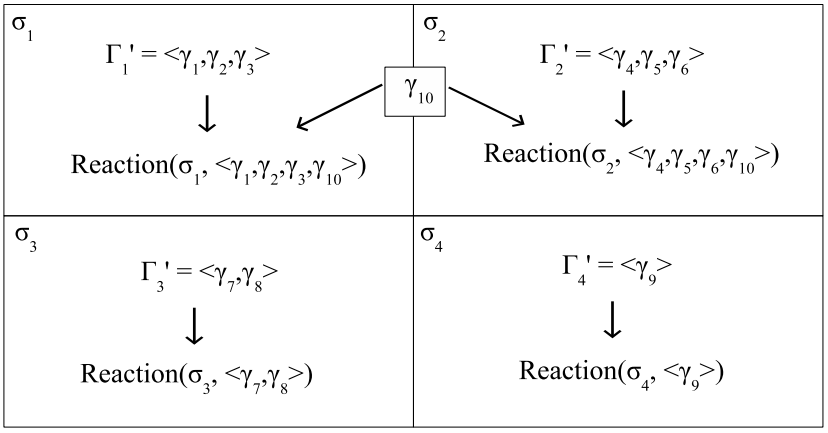

FIG. 6.5 – Décomposition du calcul de la réaction en fonction de la localité des influences.

De plus, cette décomposition est conceptuellement tout à fait correcte car la réaction est **elle-même** de nature décentralisée. Il est en effet évident que la réaction de l'environnement, en un point précis du monde, n'est pas affectée par les influences qui sont produites aux

antipodes. Cette vision des choses n'exclut cependant pas des phénomènes comme les *effets papillons* (le battement d'aile d'un papillon en Asie peut conduire, par amplifications progressives, à une tornade en Amérique). Bien qu'une influence ne puisse modifier à un instant t que son entourage immédiat, cette modification peut bien sûr entraîner, dans une zone adjacente, de nouvelles influences qui produiront de nouvelles modifications et ainsi de suite de proche en proche.

Classification des influences

Toujours dans le but de réduire la complexité du calcul de la réaction, il semble aussi intéressant de classer les influences selon leur type. Un agent qui souhaite bouger n'aura, a priori, pas d'impact sur son voisin qui lui souhaite consommer une ressource. Cette classification pourra par exemple être déterminée suivant la possibilité ou non que deux types d'influence puissent interférer. A ce propos, Weyns et Holvoet propose par exemple la classification décrite par la figure 6.6 :

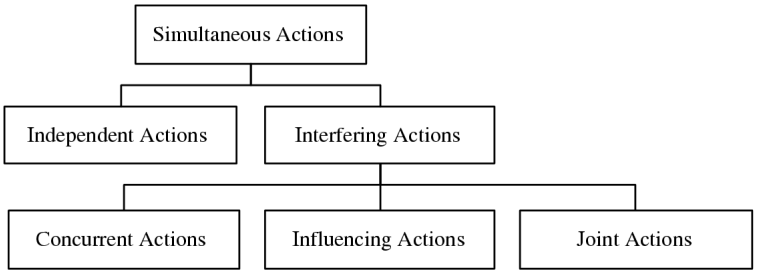

FIG. 6.6 – Une classification des actions simultanées [Weyns & Holvoet, 2003a].

Weyns et Holvoet définissent ces différentes classes de la manière suivante :

> "*We use the common name of simultaneous actions as general designation for actions that happen together. Further, we make a distinction between two kinds of simultaneous actions : independent actions and interfering actions. Independent actions are actions that do not interfere with one another. Interfering actions on the other hand, bring two or more agents directly in contact with each other. Depending on the nature of these interactions, we distinguish between concurrent actions, influencing actions and joint actions. Concurrent actions are of a conflicting nature. The result is typically nondeterministic, e.g. one arbitrary agent of the set of involved agents succeeds in his action while the other agents fail. Influencing actions are actions that positively or negatively affect each other. For this kind of interaction the outcome of the simultaneous actions is the resultant of the individual actions. Joint actions are actions that must be executed together in order to produce a successful joint result. In joint actions agents typically play complementary roles in a compound interaction. This classification for simultaneous actions takes the viewpoint of the observer of the actions. An observer interprets the interactions as a whole and distinguish types on the basis of the possible outcomes of the interactions. Whether or not the individual agents intend to, or are aware of their participation in the interaction is independent of the classification.*"

Inspirée par des travaux portant sur la planification et la collaboration dans les systèmes multi-agents, notamment [Kinny *et al.* , 1992, Griffiths & Luck, 1999], cette classification s'en distingue cependant car elle est définie du point de vue de l'observateur. En effet, bien que ces travaux traitent de la représentation d'actions simultanées, il n'est souvent question que de l'architecture interne des agents et la problématique abordée est celle de la coordination d'actions entre agents.

Cette dernière remarque nous permet d'aborder ici un point de compréhension important. La problématique liée à l'utilisation du principe Influence/Réaction n'a aucun rapport avec les travaux qui portent sur la coordination, telle qu'elle est définie par [Malone & Crowston, 1994] par exemple. La question n'est pas de savoir pourquoi ou comment les agents prennent la décision de réaliser des actions simultanées/coordonnées. Le problème est de calculer le résultat d'un ensemble d'influences produit par des entités **autonomes**. Ce n'est pas parce que deux agents très intelligents ont décidé de réaliser une tâche que les actions qu'ils vont produire sur l'environnement vont aboutir à sa finalisation. Les agents ne doivent pas avoir de contrôle sur les conséquences de leurs actes. Seul l'environnement est habilité à les calculer. Réciproquement, du point de vue de l'environnement, la structure interne d'un agent est inaccessible.

6.4.7 Récapitulation du modèle de simulation

Pour résumer :

- une influence $\gamma \in \Gamma$ n'est pas une action et ne modifie pas l'état du monde.
- l'état du monde à un instant t de la simulation est représenté par un état dynamique $\delta(t) =< \sigma(t), \gamma(t) >\in \Delta$ où $\sigma(t)$ définit les variables d'état du système : les variables environnementales et les variables physiques des agents $< \phi_{a1}(t), \phi_{a2}(t), .., \phi_{an}(t) >\in \sigma(t)$. Les agents ne peuvent pas modifier directement ces variables. $\gamma(t)$ représente l'ensemble des influences qui existent dans le système à l'instant t.
- la réaction est définie par l'application des *lois de l'univers* à un ensemble d'influences $\gamma' \in \Gamma'$ étant donné un état de l'environnement $\sigma \in \Sigma$. Le résultat de cette fonction définit les modifications effectives sur l'état global du monde.
- la perception $p_a \in P_a$ d'un agent équivaut à des informations sur une partie de $\delta(t)$.
- l'état interne $s_a \in S_a$ d'un agent est défini par les variables du système décisionnel. Elles ne sont utiles que lors des processus de *Decision* et de *Memorization*. L'agent a un contrôle actif sur ces variables.

l'évolution de la simulation entre t et $t + dt$ est défini par le passage d'un état dynamique à un autre :

$$Evolution : \Delta \mapsto \Delta \tag{6.21}$$

cette évolution se fait en deux phases distinctes appliquées de manière séquentielle :

$$Influence : \Delta \mapsto \Gamma' \qquad \textbf{Influence} \tag{6.22}$$

$$Reaction : \Sigma \times \Gamma' \mapsto \Delta \qquad \textbf{Reaction} \tag{6.23}$$

Pour prendre en compte le fonctionnement des agents et l'évolution endogène du monde, la phase influence se décompose de la façon suivante :

$$Natural_w : \Delta \mapsto \Gamma' \tag{6.24}$$

$$Perception_a : \Delta \mapsto P_a \tag{6.25}$$

$$Memorization_a : P_a \times S_a \mapsto S_a \tag{6.26}$$

$$Decision_a : S_a \mapsto \Gamma' \tag{6.27}$$

Finalement les deux phases du modèle de simulation sont régies par les équations temporelles suivantes :

$$\gamma'_w(t) = Natural_w(\delta(t)) \tag{6.28}$$

$$p_a(t) = Perception_a(\delta(t)) \tag{6.29}$$

$$s_a(t+dt) = Memorization_a(p_a(t), s_a(t)) \tag{6.30}$$

$$\gamma'_a(t) = Decision_a(s_a(t+dt)) \tag{6.31}$$

puis

$$\delta(t+dt) = Reaction(\sigma(t), \bigcup_i \gamma'_i(t)) \tag{6.32}$$

Ces équations permettent de mettre en évidence le fonctionnement d'un simulateur basé sur le principe Influence/Réaction. C'est donc à partir de ces équations que nous allons maintenant en proposer une implémentation.

6.4.8 Implémentation d'un simulateur

Pour implémenter un simulateur basé sur le modèle que nous venons de présenter, il faut mettre en place le mécanisme à deux phases sous-tendu par cette approche : phase influence puis phase réaction. La figure 6.7 illustre le déroulement d'un cycle Influence/Réaction.

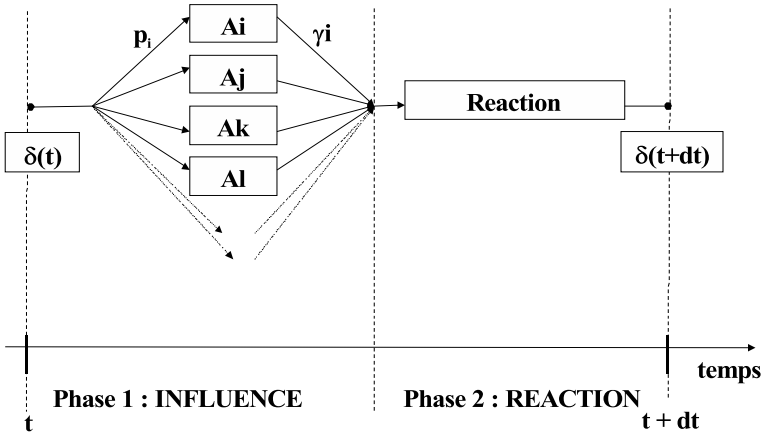

FIG. 6.7 – Déroulement d'un cycle Influence/Réaction.

Sur cette figure, on voit bien que l'ordre dans lequel les agents sont activés n'a aucune importance. En effet, la phase réaction ne peut commencer qu'à partir du moment où l'ensemble des influences ont été produites. Sachant que les influences n'entraînent aucune modification directe, les agents peuvent donc être activés dans un ordre quelconque. Il est notamment possible d'activer les agents en parallèle, ce qui facilite une éventuelle distribution de la simulation par exemple. Une fois l'ensemble des influences connu, on atteint un premier point de synchronisation qui signifie que le calcul de la réaction peut alors débuter. La fin de ce même calcul marque la terminaison du cycle Influence/Réaction et fait passer le système d'un état dynamique initial $\delta(t)$ à un nouvel état $\delta(t+dt)$.

A partir du pattern organisationnel que nous avons présenté dans la section 5.4, il est possible de définir un simulateur basé sur le principe Influence/Réaction qui tire partie de la

structure organisationnelle pour synchroniser les différentes étapes d'un cycle Influence/Réaction. Sachant que nous considérons que l'ensemble des entités qui participent à la simulation sont des agents, le principe consiste à élaborer un protocole de coordination entre les différents agents de la simulation pour synchroniser le début et la fin des deux phases d'un cycle. Le réseau de Petri de la figure 6.8 décrit un exemple de ce que peut être un tel protocole.

6.5 Application : le projet *Warbot*

6.5.1 Description

Nous avons vu que la distinction esprit/corps est intimement liée à la mise en place du principe Influence/Réaction. Elle permet en effet de faire une distinction claire entre les variables qui peuvent être modifiées par le système décisionnel, pendant la phase influence, et les variables sur lesquelles l'agent n'a aucun pouvoir de modification, c'est-à-dire ses attributs physiques et les variables d'état environnementales. Par ailleurs, cette distinction doit en principe aboutir à une modularité accrue au niveau des différents éléments logiciels utilisés pour le simulateur. On espère par exemple d'une telle distinction qu'il soit possible de remplacer entièrement le code correspondant au comportement d'un agent par un autre sans aucune difficulté. L'objectif initial du projet *Warbot* était de concrétiser effectivement cette distinction afin de mesurer les différents avantages qu'elle apporte : modularité, application du principe Influence/Réaction, etc. Par ailleurs, le projet comporte un volet pédagogique important dans la mesure où il est aussi destiné à l'apprentissage de la programmation orientée agent en troisième cycle.

Pour atteindre ces objectifs, la plate-forme *Warbot* consiste dans la simulation de plusieurs équipes de robots qui se livrent à un affrontement sans merci : le but d'une équipe est de détruire la base adverse. Elle constitue ainsi pour les étudiants le moyen de confronter, sous la forme d'une compétition, des comportements complexes élaborés de manière indépendante. Chaque robot est défini par un corps (*Body*) qui spécifie ses attributs physiques et les diverses influences que celui-ci est capable de produire.

On considère principalement deux types de corps : les explorateurs, qui se meuvent rapidement et possèdent un grand rayon de perception, et les guerriers qui bougent moins vite mais possèdent la capacité de lancer des roquettes meurtrières. A chaque corps est associé un module comportemental qui modélise l'esprit du robot (le *Brain* dans la version originale de *Warbot*).

6.5.2 Implémentation

Du point de vue de l'implémentation, le programme consiste dans l'application du mécanisme à deux phases définies par le principe Influence/Réaction. Durant la première phase, les agents produisent des influences qui sont dans un deuxième temps combinées afin de déterminer le nouvel état du monde. Lorsqu'un agent exprime le désir de bouger, celui-ci n'est donc pas toujours satisfait. Il peut par exemple avoir été tué par une roquette ou ne plus posséder d'énergie. A ce propos, il est très intéressant de noter qu'il est important pour un agent de pouvoir percevoir quel a été le résultat de son influence. Les robots possèdent par exemple une primitive de perception qui leur permet de savoir s'ils sont en train de bouger, et donc de vérifier que le déplacement se passe correctement.

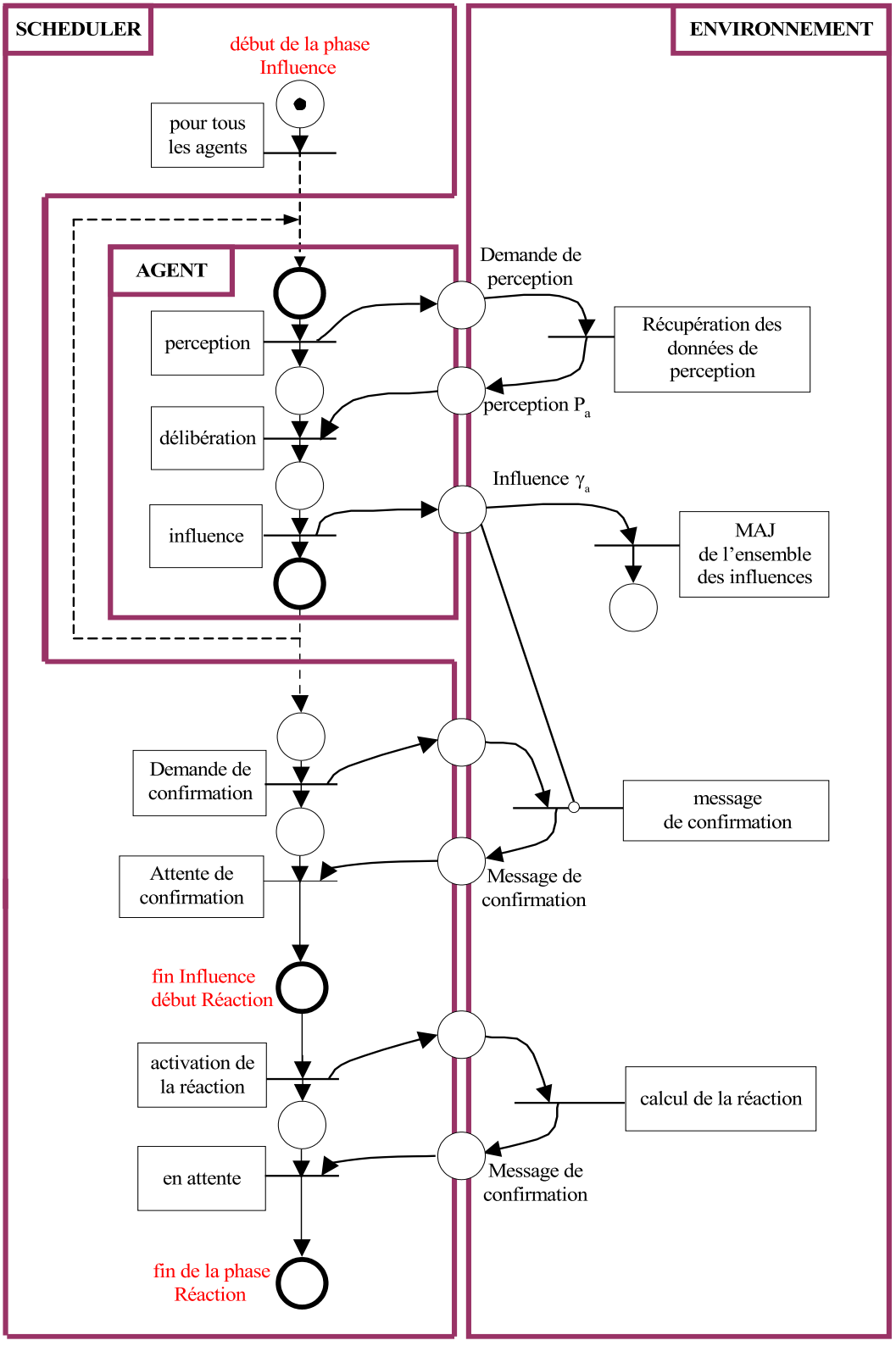

FIG. 6.8 – Implémentation d'un cycle Influence/Réaction.

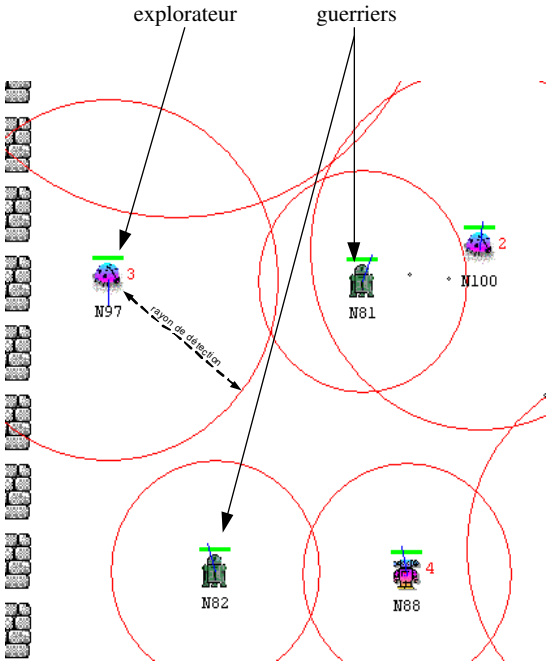

FIG. 6.9 – *Warbot* : deux équipes de robots se livrent à une guerre sans merci.

D'un point de vue pédagogique, ce genre de considérations présentent un intérêt certain car elles forcent le programmeur à prendre conscience qu'un agent est une entité qui évolue dans un environnement incontrôlable. Ce qui constitue l'une des principales difficultés dans la définition de comportements "intelligents".

Par ailleurs, l'indépendance qui existe entre le code du comportement et le corps auquel il est associé permet par exemple d'utiliser différents langages de programmation pour le codage du comportement, notamment Java et Python. Là encore, il s'agit de mettre en avant l'aspect modulaire d'un agent logiciel. L'ensemble de l'état initial de la simulation est décrit par un fichier XML qui spécifie les caractéristiques techniques et le code comportemental associé à chaque robot. En voici un extrait :

```
<node-desc name="greenMissileLauncher" class="warbot.kernel.RocketLauncher"
    category="dumb">
  <icon url="images/world/greenRobot1.gif"/>
  <property name="radius">12</property>
  <property name="team">green</property>
  <property name="detectingrange">80</property>
  <property name="energy">4000</property>
  <property name="brainClass">warbot.demo.RastaRocket</property>
  <graphic-element class="warbot.kernel.GBasicBody">
    <property name="imageaddress">images/world/greenRobot1.gif</property>
    <property name="labelLocation">4</property>
  </graphic-element>
</node-desc>
```

6.5.3 Les enseignements de l'expérience

D'un point de vue pédagogique, le projet s'est révélé très satisfaisant et les étudiants adhèrent assez rapidement au principe de l'expérience. La notion d'influence est notamment

très stimulante pour eux. Ne pas avoir de certitude quant à la conséquence des actes d'un agent est une contrainte extrêmement intéressante qui concrétise un peu mieux certains aspects de la notion d'agent logiciel. Comme nous l'avons dit, elle met en exergue le fait que la propriété d'autonomie d'un agent marque aussi son incapacité à maîtriser l'évolution de l'environnement dans lequel il se trouve.

Par ailleurs, il est amusant de voir la progression qualitative des stratégies qui sont imaginées d'année en année. Sur la base des améliorations issues des années précédentes, chaque promotion est en effet forcée de découvrir des stratégies toujours plus efficaces qui mettent en évidence l'importance de la collaboration dans les systèmes multi-agents.

Cette expérience a été aussi très enrichissante du point de vue de nos préoccupations de recherche, même dans ses aspects négatifs. Le volet pédagogique nous a tout d'abord conforté dans l'idée que la notion d'influence, et à travers elle l'abandon de la modélisation de l'action par transformation d'état, est un point tout à fait essentiel dans la modélisation des systèmes multi-agents. En revanche, le modèle de simulation obtenu s'est révélé frustrant en ce qui concerne la modélisation de la simultanéité. En fait, les différentes influences qui sont présentes dans le modèle (mouvements et tirs) ne se prêtent pas spécialement bien à une modélisation intéressante de leur simultanéité.

Comme nous l'avons dit précédemment, il est important que la simultanéité soit définie dans le modèle lui-même pour qu'elle soit concrétisée dans l'implémentation. Et dans le cas du modèle proposé dans *Warbot*, la combinaison des influences ne donne pas intuitivement des événements distingués qui augmenteraient le pouvoir d'expressivité du modèle. Par exemple, les mouvements sont traités de façon séquentielle alors qu'il aurait été possible de considérer que, lorsque deux agents avancent dans la même direction et qu'une collision va avoir lieu, aucun des deux agents n'est satisfait dans son mouvement, ou encore que l'espace disponible entre les deux agents est équitablement réparti entre les deux mouvements. Cependant, outre le fait que cela aurait significativement compliqué l'implémentation du système, ce souci de modélisation n'aurait pas eu un grand intérêt. En effet, le cadre expérimental n'était pas du tout lié à la modélisation quasi parfaite des mouvements, d'autres simulateurs le font très bien. Il aurait donc été assez maladroit de programmer une approche complexe de la simultanéité alors que la composante physique des agents est grossièrement représentée par un simple cercle dans le modèle. Il y aurait eu là une contradiction entre ces deux points du modèle. L'effort de modélisation doit être réparti équitablement dans les différents aspects d'un modèle.

Faut-il ici en déduire que l'utilisation du principe Influence/Réaction a uniquement contribué à rendre complexe un modèle qui aurait pu être encore plus simple ? Non. Au contraire, de cette frustration nous avons compris un aspect fondamental du principe Influence/Réaction : bien que celui-ci ait été pensé dans le but de fournir un moyen de modéliser concrètement la simultanéité, nous allons voir dans le chapitre suivant que son intérêt majeur réside dans le fait que son application définit deux contraintes d'implémentation fondamentales.

La première contrainte consiste dans l'abandon de la notion d'action comme une transformation d'un état global. La deuxième concerne la nécessité de distinguer les deux composantes d'un agent que sont le corps et l'esprit. La première contrainte va nous permettre de mettre en évidence l'intérêt d'isoler le traitement et l'implémentation de l'interaction dans un module distingué : le module des interactions. Ajoutée à la première, la deuxième contrainte va quant à elle nous permettre de donner une véritable sémantique computationnelle à l'une des plus importantes caractéristiques d'un agent : l'autonomie. Dans le chapitre qui suit nous allons développer et argumenter ces idées.

6.6 Résumé du chapitre

Dans ce chapitre nous avons tout d'abord présenté le problème de la modélisation de la simultanéité dans les systèmes multi-agents. Nous avons vu que la modélisation classique de l'action comme une transformation d'un état global se prête difficilement à une modélisation satisfaisante de la simultanéité. Nous avons ensuite soutenu l'idée que le principe Influence/-Réaction propose une solution efficace à ce problème et nous en avons proposé une adaptation pour la simulation. Finalement, l'expérience que nous avons acquise par l'intermédiaire du projet *Warbot* nous a poussé à considérer que le principe Influence/Réaction est non seulement une solution au problème de la simultanéité mais qu'il permet aussi de concrétiser plus formellement certains aspects du paradigme agent comme nous allons maintenant le voir.

Chapitre 7

Modélisation des interactions et cohérence paradigmatique

DANS le chapitre précédent nous avons présenté le principe Influence/Réaction comme une solution pertinente au problème de la modélisation d'actions simultanées. Nous avons vu qu'il permet de prendre en compte l'ensemble des actions produites pour un même instant t. Cependant, nous avons abordé la question du calcul de la réaction de manière générale, sans en préciser les enjeux conceptuels. La réaction constitue en fait le cœur de la modélisation d'un système multi-agents car elle définit sa dynamique sous la forme des *lois de l'univers*. Et c'est véritablement dans la réaction que les interactions entre les différentes entités d'un système sont concrètement modélisées. Il s'agit en effet d'apporter une réponse aux interactions engendrées par l'ensemble des comportements autonomes. Ce chapitre est consacré à cette question.

Dans un premier temps, nous allons voir pourquoi il est important de donner une définition technique de l'interaction qui soit relative au seul contexte de la simulation multi-agents. Cela nous permettra notamment d'isoler la problématique qui nous intéresse sous la forme du *module des interactions*. Nous verrons ensuite que la modélisation de ce module peut non seulement avoir une grande influence sur les résultats d'un modèle, mais aussi qu'elle peut parfois remettre en cause l'autonomie des agents telle que nous allons bientôt la définir. Nous montrerons alors qu'il est intéressant de considérer un nouvel aspect de la validation d'un modèle : la *cohérence paradigmatique*. Dans ce contexte, nous verrons que la nature des interactions joue un rôle important du point de vue de la manière dont elles doivent être modélisées. Relativement à ces considérations, nous essayerons de dégager deux grandes classes de situations interactionnelles que nous distinguerons en fonction du traitement qu'elles requièrent. Cela nous permettra de conclure sur l'idée que le principe Influence/Réaction et bien plus qu'un simple modèle de la simultanéité.

7.1 L'interaction : point essentiel d'un système multi-agents

7.1.1 Interaction : un terme trop générique

Prenons tout d'abord la définition du terme *interaction* telle qu'on peut la trouver dans un dictionnaire[1] :

Définition 7.1 (INTERACTION n. f. XIX^{e} siècle.) PHYS. *Action réciproque de deux ou plusieurs corps. La gravitation est un phénomène d'interaction entre deux corps...*
Par ext. *Influence qu'exercent les uns sur les autres des phénomènes, des faits, des objets, des personnes.*

Etant donnée cette définition, il est naturel que l'interaction soit une notion qui joue un rôle fondamental dans l'approche sous-tendue par le paradigme agent. Cependant, s'il est évident qu'il existe des interactions entre les entités d'un système complexe, cette définition très générique ne nous apprend pas grand-chose sur ce qui doit être considéré comme une interaction dans l'implémentation d'un système multi-agents. Comme tout mot-clé, le terme *interaction* est donc utilisé dans de nombreux contextes et avec différents vocabulaires : collaboration, coopération, compétition, coordination, etc [Ferber, 1999]. A ce propos, [Parunak *et al.* , 2003] a récemment proposé une taxonomie très intéressante qui tente de définir les questions qui se rapportent aux différentes caractéristiques que l'on prête aux interactions dans un système multi-agents.

Dans la majorité des travaux qui traitent de ces questions, on peut remarquer que la notion d'interaction a le plus souvent pour contexte celui de l'ingénierie logicielle multi-agents. Elle fait alors référence à l'aspect qualitatif des moyens utilisés par les agents pour communiquer (interagir) et atteindre leurs buts, notamment à travers l'utilisation de la notion d'*actes de langage*. Sans entrer dans le détail, il s'agit par exemple de savoir quelle est la nature des messages qui doivent être échangés entre les agents pour que le système arrive dans un état désiré (la vente d'un billet de train par exemple). Dans ce contexte, on parle par exemple de *protocoles d'interaction*. Dans [Huget, 2001], on retrouve une introduction exhaustive à cette problématique. Pour utiliser un raccourci, on peut dire qu'il s'agit là d'étudier la manière dont les agents doivent agir, et donc interagir, étant donné une situation et un objectif.

7.1.2 Définition de l'interaction dans le contexte de la simulation

Dans le cadre de notre thèse, nous allons utiliser le terme *interaction* pour désigner un tout autre aspect de l'implémentation d'un système multi-agents. En effet, dans le contexte de la simulation, la problématique précédente ne nous concerne pas directement. En fait, il s'agit là d'une question qui se situe à un niveau applicatif et qui ne concerne pas la manière dont un système est effectivement simulé. Lors de la modélisation d'un système multi-agents, le problème n'est pas de savoir pourquoi les agents agissent mais bien de savoir comment leurs actions, et donc leurs interactions, participent à la dynamique globale du système.

Notre idée est que le terme interaction doit impérativement être mis en relation avec une partie précise des spécifications. C'est pourquoi nous allons donner ici une définition technique qui va nous permettre d'identifier clairement ce que nous entendons par *interaction* d'un point de vue computationnel, dans le contexte de la simulation.

[1] Définition issue du dictionnaire de l'Académie (9^{e} édition) via le site *Le Trésor de la Langue Française Informatisée TLFI* : atilf.atilf.fr.

Définition 7.2 (INTERACTION dans la simulation multi-agents.) *L'**interaction** désigne, dans la modélisation/simulation d'un système multi-agents, la dynamique engendrée par les influences qui sont concomitantes à un instant t de la simulation. Par extension, la **gestion de l'interaction** correspond à un calcul réalisé par une fonction (la réaction) tel qu'il permet de déterminer le nouvel état du monde étant donné un ensemble d'influences et l'état du monde courant.*

Bien que cette définition utilise le vocabulaire du principe Influence/Réaction, elle peut être généralisée à tout modèle de simulation multi-agents car il est évidemment possible de traiter les influences sur le principe d'une fonction par transformation d'état. Dans ce cas, la gestion de l'interaction correspond à la manière dont les actions des agents sont validées, c'est-à-dire directement ou à l'aide d'une méthode de résolution de conflits. Dans le contexte du modèle Influence/Réaction, la gestion de l'interaction correspond aux calculs effectués pendant la phase de réaction.

Cette définition nous permet donc tout d'abord d'expliciter la partie de l'implémentation/modélisation à laquelle nous associons la notion d'interaction. Plus fondamentalement, elle souligne aussi le fait que nous considérons que l'interaction doit constituer à elle seule un point fondamental du modèle. A ce titre, elle doit faire l'objet d'une attention particulière, à l'instar de la modélisation du temps, de l'environnement et du comportement des agents.

7.1.3 Nécessité d'identifier un module interactionnel

A partir de la définition précédente, force est de constater que la description de la gestion de l'interaction est souvent le principal point faible des spécifications d'un modèle. En effet, alors que l'environnement, le comportement des agents et le modèle temporel sont en général décrits dans le détail, très peu d'informations sont disponibles quant au traitement de l'interaction. En fait, cette question est souvent diluée dans la gestion des autres modules.

Par conséquent, s'il est certain que les agents interagissent entre eux, il est finalement très difficile de dire où le code informatique qui concrétise cette interaction est localisé : est-elle gérée par les agents, par l'environnement, par l'ordonnanceur ? Où l'interaction est-elle calculée ? Lorsque ces questions n'ont pas été prises en compte a priori, la gestion de l'interaction devient alors un point d'ambiguïté important dans les spécifications d'implémentation.

Pour s'en convaincre, reprenons l'exemple de l'expérience menée par Edmonds et Hales que nous avons présentée dans la section 4.2.2 (page 82). C'est précisément sur cette question que les auteurs obtiennent en fait des résultats différents du modèle original du fait d'une interprétation différente de l'interaction. Comme le montre la figure 7.1, les auteurs ont en effet identifié un point d'ambiguïté où il était possible d'implémenter le modèle original de plusieurs façons étant donné l'imprécision des spécifications.

Comme on peut le voir, il s'agit bien d'une ambiguïté qui concerne la manière dont est modélisé le résultat de deux comportements considérés en interaction à un instant précis de la simulation. En effet, la méthode d'ordonnancement et le comportement des agents restent les mêmes dans les trois algorithmes. Seule la gestion de l'interaction est en fait modifiée.

C'est pourquoi nous considérons qu'il est fondamental de considérer la gestion des interactions comme une partie distinguée de l'ensemble du processus de modélisation. C'est-à-dire comme un module à part entière dans l'architecture du modèle et donc du simulateur. Outre l'intérêt pratique et conceptuel de séparer explicitement la modélisation et le traitement de

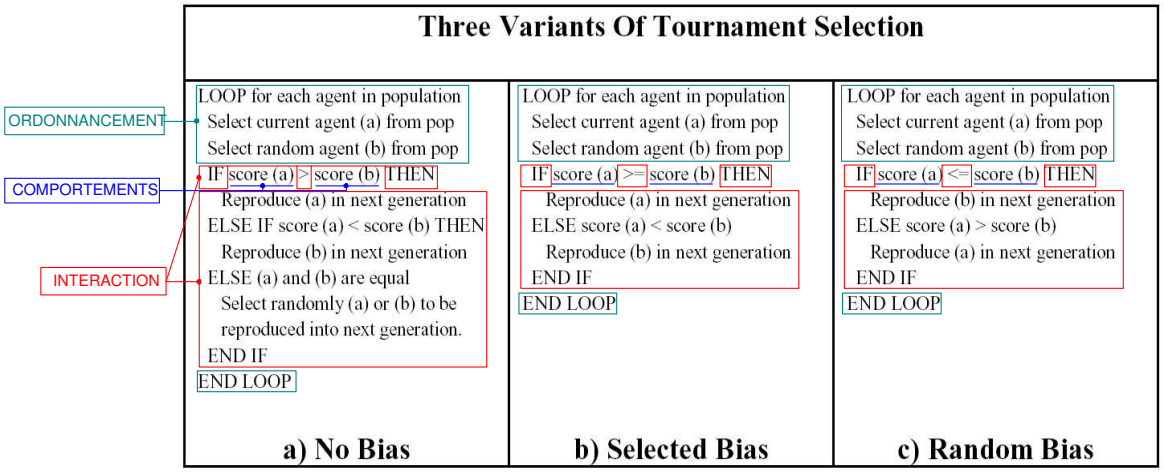

FIG. 7.1 – Identification du code lié à l'interaction dans [Edmonds & Hales, 2003].

l'interaction des autres parties du modèle, nous pensons que l'interaction ne doit pas se fondre dans les autres modules pour plusieurs raisons :

- en ce qui concerne le module des comportements. Nous avons plusieurs fois argumenté sur l'importance de ne pas laisser aux agents la capacité de modifier directement les variables de l'environnement. C'est pourquoi, ce n'est certainement pas dans le code qui représente le comportement d'un agent que doit se trouver la gestion des interactions.
- en ce qui concerne le module d'ordonnancement. Il est important de comprendre que, tel que nous avons défini le problème de la gestion de l'interaction, celui-ci n'est pas lié à une représentation du temps particulière. En effet, que le temps évolue de manière discrète ou suivant un enchaînement d'événements, nous ne nous intéressons qu'à ce qui se passe pour un unique instant t de la simulation. C'est pourquoi, quel que soit le modèle temporel utilisé, la gestion de l'interaction n'a pas de lien direct avec celui-ci.
- finalement, bien que la gestion des interactions soit en quelque sorte une concrétisation des lois environnementales (les *lois de l'univers* dans le contexte du modèle Influence/Réaction), nous pensons qu'il est important de ne pas confondre la modélisation de l'environnement, variables d'état et dynamique endogène, avec le calcul des interactions qui ont lieu entre les différentes entités du système. Il est notamment possible de concevoir des systèmes où il n'y a pas d'environnement situé mais où il existe des interactions, comme c'est le cas dans un système composé d'agents purement communicants par exemple.

7.2 L'exemple de l'interaction de reproduction

Dans cette section, nous allons tout d'abord présenter trois modèles différents qui peuvent être utilisés pour modéliser l'interaction de reproduction dans un système multi-agents. Nous montrerons en quoi ces trois modèles divergent et fournissent des résultats très éloignés les uns des autres.

7.2.1 Trois modèles de l'interaction de reproduction

1. [Epstein & Axtell, 1996]

Proposé à l'origine par Epstein et Axtell dans [Epstein & Axtell, 1996], le modèle *SugarScape*, dont il existe de multiples adaptations, est défini par un espace discrétisé sous la forme d'une grille en deux dimensions (figure 7.2) dans laquelle une ressource (le sucre) est répartie aléatoirement ou suivant une topologie précise. Les agents doivent collecter cette ressource

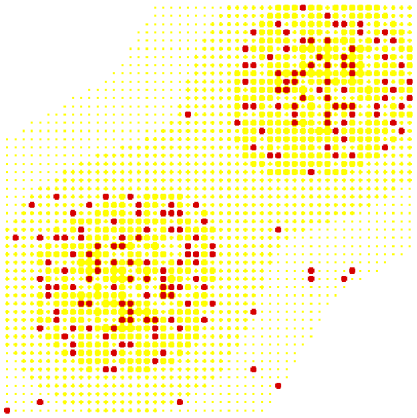

FIG. 7.2 – *SugarScape.*

pour rester en vie. Dans ce modèle, les agents sont définis par un ensemble de règles comportementales qui leur permettent de percevoir leur environnement suivant un certain rayon, de bouger, de collecter des ressources et de se reproduire entre eux. L'interaction de reproduction est donc une composante fondamentale de ce modèle. Dans [Epstein & Axtell, 1996], pour modéliser cette interaction, les agents en âge de procréer (fertiles) se reproduisent suivant la loi comportementale **S** telle que :

Algorithme *Agent sex rule* **S :**
(∗ [Epstein & Axtell, 1996] ∗)
1. Select a neighboring agent at random
2. **if** the neighbor is fertile **and** of the opposite sex **and** at least one of the agents has an empty neighboring site
3. **then** a child is born
4. **repeat** for all neighbors

De plus, la simulation utilise un modèle du temps discret où tous les agents sont activés à chaque pas de temps dans un ordre aléatoire et leurs actions immédiatement validées. Comme le remarquent les auteurs eux-mêmes, l'application de cette règle comportementale pose un problème. Elle peut occasionnellement conduire à ce qu'un agent se reproduise plus d'une fois par tour. Par exemple, si trois agents se trouvent à proximité, il est possible d'obtenir six nouvelles entités issues de l'interaction de reproduction. Et bien que l'objet de l'expérience soit l'étude d'une société électronique virtuelle, la validité de la modélisation de cette interaction est discutable sachant que les comportements considérés dans [Epstein & Axtell, 1996] sont inspirés par l'humain. Epstein et Axtell expliquent à ce propos que chaque agent fournit un certain nombre de points de vie pour le processus de reproduction et qu'un minimum est nécessaire pour que celle-ci puisse avoir lieu, ce qui rend assez rare la situation précédente. De plus, après avoir considéré l'ajout de conditions supplémentaires pour éviter ce genre de situation, les auteurs ont préféré conserver ce modèle de manière à avoir des règles les plus

simples possibles, malgré les biais de simulation engendrés. Cette position a d'ailleurs valu aux auteurs de virulentes critiques au sujet de l'interprétation des résultats (cf. [Terna, 2001]).

2. [Lawson & Park, 2000]

Dans cette adaptation de *SugarScape*, l'interaction de reproduction a été modifiée de manière à la rendre plus *réaliste*, pour reprendre le terme exact de l'article. Afin d'éviter que la situation précédente ne puisse se produire, une période de gestation a été rajoutée :

> "*We have modified the agent reproduction rule to be more realistic, incorporating a gestation period... If a mate is found (i.e., there is at least one candidate agent), the female agent of the pair becomes pregnant. Throughout the gestation period, neither the male nor the female agent can move or attempt to reproduce.*"

Bien que cette technique permette d'éviter les situations exotiques qui étaient observées dans [Epstein & Axtell, 1996], nous verrons quelle pose cependant des problèmes vis-à-vis des fondements du paradigme agent.

3. [Michel, 2001]

Dans le cadre d'une simulation de type proies/prédateur, nous avons proposé un autre modèle de l'interaction de reproduction qui utilise le principe Influence/Réaction. Dans ce modèle, l'interaction de reproduction nécessite que deux agents souhaitent se reproduire en même temps. Autrement dit, la validation du succès de la reproduction ne se trouve pas dans le code représentant le comportement d'un agent, contrairement aux deux modèles précédents. Ainsi, les agents produisent, entre autres, des influences de reproduction qui sont interprétées par l'environnement pour valider ou non une nouvelle naissance suivant les couplages qui peuvent être réalisés étant donnés les desiderata de chacun.

7.2.2 Divergence entre les trois modèles

Les trois modèles que nous avons présentés sont clairement trois manières très différentes de représenter l'interaction de reproduction. Cependant, il est important de comprendre que si les spécifications du modèle de simulation ne sont pas clairement définies, notamment à propos de la gestion des interactions (module interactionnel), les implémentations qui correspondent à chacun de ces modèles sont toutes les trois envisageables par la personne qui va implémenter le simulateur. Et cela peut conduire à des résultats extrêmement différents suivant les choix qui ont été décidés pour l'implémentation.

Pour s'en convaincre, nous allons ici reprendre les différentes techniques utilisées pour représenter la reproduction dans le contexte d'une simulation minimaliste (sans environnement physique) impliquant uniquement deux agents compatibles (fertiles et de sexe opposé). Comme résultats, nous comptabiliserons le nombre de naissances au cours du temps sans ajouter les nouveaux agents dans le système. Pour clairement identifier l'impact de l'implémentation du module interactionnel sur les résultats de cette simulation, nous allons fixer une fois pour toutes les autres modules. En ce qui concerne l'échéancier, nous utiliserons une simulation à pas de temps constant classique. Le comportement des deux agents, A et B, (module comportemental : table 7.1) est quant à lui défini par une probabilité qui modélise le souhait de se reproduire ou de faire une autre action, peu importe laquelle (*none*) : Ces deux composantes

TAB. 7.1 – Modèle du comportement interne des agents

$Pr(A_{repro}) = \alpha$ and $Pr(A_{none}) = 1 - \alpha$
$Pr(B_{repro}) = \beta$ and $Pr(B_{none}) = 1 - \beta$

du modèle de simulation étant fixées, il existe quatre situations d'interactions différentes qui devront être traitées par le module interactionnel : table 7.2.

TAB. 7.2 – Les quatre situations d'interactions possibles

$Pr(A_{repro}$ and $B_{repro})$	$= Pr(A_{repro}) \times Pr(B_{repro})$	$= \alpha\beta$
$Pr(A_{repro}$ and $B_{none})$	$= Pr(A_{repro}) \times Pr(B_{none})$	$= \alpha - \alpha\beta$
$Pr(A_{none}$ and $B_{repro})$	$= Pr(A_{none}) \times Pr(B_{repro})$	$= \beta - \alpha\beta$
$Pr(A_{none}$ and $B_{none})$	$= Pr(A_{none}) \times Pr(B_{none})$	$= 1 - \beta - \alpha + \alpha\beta$

Nous allons maintenant définir successivement trois modules interactionnels inspirés par les modèles que nous avons présentés précédemment.

Premier module interactionnel : "à la [Epstein & Axtell, 1996]"

Dans cette première version du modèle interactionnel, l'action d'un agent est prise en compte immédiatement et un comportement de reproduction entraîne toujours une naissance étant donnée la compatibilité des agents (table 7.3) : On voit ici qu'avec cette implémentation

TAB. 7.3 – Premier module interactionnel

situations	birth(s)	probabilty
A_{repro}, B_{repro}	2	$\alpha\beta$
A_{repro}, B_{none}	1	$\alpha - \alpha\beta$
A_{none}, B_{repro}	1	$\beta - \alpha\beta$
A_{none}, B_{none}	0	$1 - \beta - \alpha + \alpha\beta$
Résultats pour un pas de temps		
$Pr(births = 2) = \alpha\beta$		
$Pr(birth = 1) = \alpha + \beta - 2\alpha\beta$		
$Pr(birth = 0) = 1 - \beta - \alpha + \alpha\beta$		

de l'interaction de reproduction, on retrouve la possibilité pour un agent de se reproduire plus d'une fois pour un seul pas de temps.

Second module interactionnel : "à la [Lawson & Park, 2000]"

Une deuxième solution consiste à éviter qu'un agent puisse se reproduire plus d'une fois par tour comme le propose [Lawson & Park, 2000]. Pour cela, dans cette version du module interactionnel, lorsque l'agent qui est activé en premier décide de se reproduire, le deuxième agent subit un marquage qui préviendra une deuxième naissance (table 7.4) : On voit ici

TAB. 7.4 – Second module interactionnel

situations	birth(s)	probability
A_{repro}, B_{repro}	1	$\alpha\beta$
A_{repro}, B_{none}	1	$\alpha - \alpha\beta$
A_{none}, B_{repro}	1	$\beta - \alpha\beta$
A_{none}, B_{none}	0	$1 - \beta - \alpha + \alpha\beta$
Résultats pour un pas de temps		
$Pr(birth = 1) = \alpha + \beta - \alpha\beta$ $Pr(birth = 0) = 1 - \beta - \alpha + \alpha\beta$		

qu'avec cette implémentation de l'interaction de reproduction, il n'est plus possible d'obtenir une situation où deux nouveaux agents sont créés.

Troisième module interactionnel : "à la Influence/Réaction "

Dans cette dernière version, nous allons appliquer le principe Influence/Réaction à l'interaction de reproduction. Cette fois, il est nécessaire que les deux agents souhaitent se reproduire en même temps pour qu'une nouvelle naissance soit effective (table 7.4) : Comme on peut le

TAB. 7.5 – Troisième module interactionnel

situations	birth(s)	probabilty
A_{repro}, B_{repro}	1	$\alpha\beta$
A_{repro}, B_{none}	0	$\alpha - \alpha\beta$
A_{none}, B_{repro}	0	$\beta - \alpha\beta$
A_{none}, B_{none}	0	$1 - \beta - \alpha + \alpha\beta$
Résultats pour un pas de temps		
$Pr(birth = 1) = \alpha\beta$ $Pr(birth = 0) = 1 - \alpha\beta$		

voir, dans cette version du modèle interactionnel, il n'y a plus qu'une seule situation qui engendre une naissance. On voit ici toute la différence avec les deux premières versions dans lesquelles les agents valident eux-mêmes le résultat de leurs actes.

Résultats

La figure 7.3 montre les différents résultats obtenus en fonction de chaque module interactionnel pour une centaine de simulations de chaque version. L'ensemble de courbes A correspond à l'application de la première méthode, B à celle de la deuxième et C de la troisième. Ces résultats montrent clairement l'impact que peut avoir l'implémentation des interactions sur les résultats d'une simulation. En effet, bien que le comportement interne des agents n'ait pas été modifié, les résultats divergent fortement.

Avec cette expérience, notre objectif est de montrer qu'il est indispensable de faire une différence claire entre le module comportemental des agents, l'échéancier et le module inter-

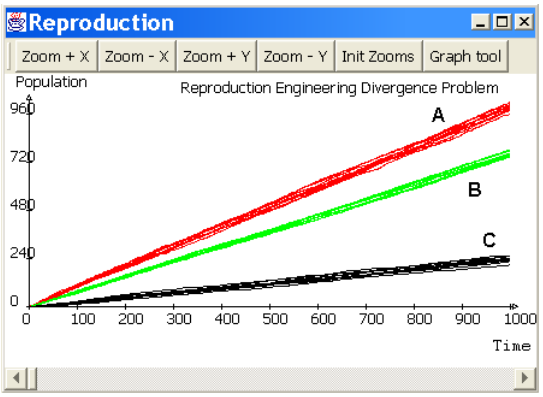

FIG. 7.3 – Résultats obtenus avec les trois modules interactionnels.

actionnel. Si aucun modèle interactionnel n'est défini, les chances d'avoir un phénomène de divergence implémentatoire sont extrêmement grandes. L'exemple que nous avons donné est extrême dans le sens où les spécifications initiales étaient caricaturalement pauvres en information et que vraisemblablement personne ne les aurait implémentées sans demander plus de détails sur le modèle. Cependant, il suffit d'imaginer la complexité de certains modèles qui sont aujourd'hui proposés pour la simulation multi-agents pour comprendre l'importance que peut avoir le manque de spécifications dans un modèle multi-agents. C'est pourquoi nous sommes persuadé que l'identification systématique d'un module des interactions est un pas important vers des spécifications sans ambiguïté/oubli, et donc vers une réduction des phénomènes de divergence implémentatoire. En définissant clairement quelles sont les différents aspects fondamentaux d'un modèle multi-agents simulé, le but de notre démarche est de faciliter une mise en correspondance directe entre les différentes parties d'un modèle et leurs implémentations. Il s'agit ainsi de faciliter l'étude et la vérification de la relation de simulation.

7.2.3 Y a-t-il un modèle conceptuellement plus valide que les autres ?

Etant données les trois modélisations de l'interaction de reproduction que nous venons de présenter, il faut se demander s'il convient d'en choisir une plutôt que l'autre. En effet, nous nous retrouvons avec trois possibilités de modéliser l'interaction de reproduction et cela pose la question de leur comparaison : faut-il en sélectionner une ou systématiquement explorer les trois manières de procéder lorsqu'on souhaite modéliser la reproduction dans un système multi-agents ?

A priori, cette question peut sembler hors de propos. En effet, on peut en toute légitimité considérer que chacun de ces modèles est très différent de l'autre et qu'ils sont tout simplement incomparables parce qu'ils ne représentent pas le même système. Cependant, c'est une question comme celle-là qui a poussé Lawson et Park à considérer l'élaboration d'une nouvelle règle de reproduction afin de rendre l'ensemble du modèle *SugarScape* plus *réaliste*. Ils font ainsi explicitement une comparaison entre les deux versions et considèrent donc que le modèle original ne représente pas correctement le système que l'on souhaite modéliser. Que faut-il penser d'une telle analyse ? Disposons-nous de moyens méthodologiques qui nous permettraient de juger de la validité d'un modèle multi-agents ? Si ces questions se rapportent toujours aux problématiques liées à la vérification et à la validation d'une expérience de simulation, elles ne se situent plus dans le contexte de la relation de simulation mais abordent la question de la *relation de modélisation*. Nous allons maintenant aborder cette problématique et nous

allons montrer que l'analyse de la gestion des interactions constitue un facteur important de l'évaluation de la validité d'une modélisation multi-agents.

7.3 Etude de la relation de modélisation dans les simulations orientées multi-agents

Jusqu'à présent, nous nous sommes principalement intéressé aux problèmes liés à l'étude de la relation de simulation. Comme nous l'avons vu dans la section 2.8.3 du chapitre 2 (page 41), un autre aspect fondamental de la théorie de la M&S consiste dans l'analyse de la *relation de modélisation*. Cette relation soulève la question suivante : le modèle étudié est-il valide étant donné le cadre expérimental considéré ? Autrement dit, il s'agit là d'évaluer la qualité d'un modèle et non plus de savoir si son implémentation est correcte. Il ne s'agit plus de vérifier concrètement la mise en œuvre du modèle mais de valider l'approche qui a été utilisée pour le construire.

7.3.1 La validité d'un modèle multi-agents : question vaste et difficile

De par la nature et la complexité des modèles envisagés, il est dans certains cas très difficile de juger de la validité d'une simulation multi-agents. D'une part, on ne dispose pas toujours d'une base de données comportementale sur le système, auquel cas les résultats de la simulation ne peuvent pas être vérifiés d'un point de vue quantitatif. D'autre part, en s'intéressant à la modélisation d'un ensemble d'entités autonomes en interaction, les simulations multi-agents cherchent à représenter des processus complexes pour lesquels on ne pourra sans doute jamais posséder un modèle définitif. A l'heure actuelle, si l'on peut simuler un système mécanique réel avec une grande précision en utilisant les lois de la physique newtonienne, nous sommes encore loin de pouvoir prétendre modéliser avec la même confiance des processus aussi complexes que ceux que l'on peut trouver dans une société humaine par exemple.

Par ailleurs, lorsque le système source étudié est virtuel, l'objectif de la simulation est avant tout de disposer d'un outil de réflexion supplémentaire. Par exemple, dans le domaine de la *vie artificielle*, la simulation est beaucoup plus un exercice de modélisation/compréhension que de prédiction, comme le souligne très justement Cariani ([Cariani, 1991]) :

> "*The interesting emergent events that involve artificial life simulations reside not in the simulations themselves, but in the ways that they change the way we think and interact with the world.*"

Ainsi, il semble en effet très difficile de traiter les différents aspects liés à l'étude de la relation de modélisation dans le cadre de la simulation multi-agents. La *cohérence reproductive*, la *cohérence évolutive* et la *cohérence structurelle* (cf. section 2.8.3 page 41) sont autant de notions que nous sommes généralement dans l'impossibilité d'étudier.

7.3.2 Approche classique

D'une manière générale, l'aspect du modèle dont la validité est étudiée concerne l'adéquation entre celui-ci, les résultats obtenus et les connaissances du spécialiste du domaine. Par exemple, le fait qu'un agent puisse se reproduire plus d'une fois par tour a été jugé peu réaliste par Lawson et Park et cette remarque les a amenés à modifier le modèle. Dans ce

cadre, c'est la qualité conceptuelle du modèle qui est analysée. L'une des principales conclusions que Lawson et Park tirent par exemple de leurs travaux est qu'un principe événementiel convient mieux qu'une simulation à pas de temps constant pour la modélisation d'une société artificielle. Il s'agit ainsi pour eux de valider une approche de modélisation par rapport à une autre en fonction d'un cadre expérimental précis. La plupart des travaux qui traitent de la validité des simulations multi-agents se focalisent sur cette problématique (cf. [Axtell, 2000a, Huberman & Glance, 1993] par exemple). En général, une telle étude s'accompagne d'une analyse de sensibilité du modèle aux différents paramètres qui le composent. Ce qui facilite l'identification des points du modèle où les principes de modélisation utilisés ont une grande importance.

Bien qu'une telle analyse soit indispensable, elle est cependant très difficilement généralisable. En effet, la validation de ces aspects de la modélisation est une tâche extrêmement subjective. En effet, elle est en grande partie liée à l'interprétation que l'expert du domaine fait des résultats observés. Dans la section 4.3.1 (page 84), nous avons suggéré que le programmeur est sujet à de nombreuses tentations. Il en va de même en ce qui concerne le modélisateur. Celui-ci peut être tenté de valider la modélisation lorsque les résultats obtenus par la simulation sont en accord avec sa propre intuition et, inversement, il peut considérer que le modèle doit être modifié si les résultats escomptés ne sont pas au rendez-vous. En fait, dans le domaine de la simulation multi-agents, on est beaucoup plus dans une approche d'appréciation que de validation des modèles.

7.3.3 Nécessité d'intégrer de nouveaux aspects dans la validation des simulations orientées multi-agents

Il n'existe ainsi aujourd'hui aucun critère objectif qui permette de déterminer la validité du processus de modélisation d'un système multi-agents. En l'état, rien ne nous permet donc de dire que l'un des trois modèles que nous avons présentés pour l'interaction de reproduction est plus valide que les autres. Tout au plus peut-on dire que les deux derniers semblent plus cohérents car ils interdisent les reproductions multiples, ce qui reste un point de vue. Ce dont nous avons ici besoin, c'est de moyens pragmatiques et invariants d'évaluer la validité intrinsèque d'un modèle multi-agents. Sans cela, modéliser un système multi-agents reste un exercice scientifique pour lequel il n'existe pas de lignes directrices et où le bon sens fait office de *principe de bonne modélisation*. [David *et al.* , 2002] propose une introduction et une analyse très intéressantes de cette problématique. Nous rejoignons complètement le point de vue de ces auteurs qui font l'analyse suivante de la situation :

> "*The logic underlying this strategy is that if a program is correctly verified then its outputs are entailed by the conceptual model specification. This assertion would in fact be correct if we could rigorously verify the correctness of reasonable complex code ... In our vision, reliability is presently the fundamental problem in ABS (Agent Based Simulation). The problem of validation has been an important research issue, but are we adopting the right principles ? Should we insist with the use of classic approaches and assumptions in regard to verification ? ... Nevertheless, is there an alternative software process for ABS ? At the present maturing stage of ABS there is not yet an answer to these questions.*"

Une des idées très intéressantes de cet article est de proposer une spécification qui fait le lien entre le modèle conceptuel et son implémentation. Les auteurs proposent notamment d'étudier les liens qui existent entre la modélisation du niveau micro et la modélisation du

niveau macro grâce à une définition non ambiguë de ces deux notions, aussi bien dans le modèle que dans son implémentation. Ce qui facilite la traçabilité de ces mécanismes, et donc leur analyse. Nous pensons que l'idée majeure qu'il faut retenir de cet article est la volonté affichée par les auteurs de désenchanter la complexité de la dynamique d'un système multi-agents. En identifiant concrètement, dans le processus de modélisation, des concepts comme l'interaction du niveau micro avec le niveau macro, elle en permet effectivement une étude objective et pragmatique.

Nous pensons nous aussi qu'une telle approche est aujourd'hui fondamentale et qu'il est important que la modélisation d'un système multi-agents soit effectuée dans un contexte où le vocabulaire que nous utilisons désigne des aspects concrets et invariables du modèle et de son implémentation. C'est dans ce but que nous avons identifié un module des interactions ; pour en permettre l'étude. Nous pensons en effet qu'une grande partie de la dynamique d'un système multi-agents se joue dans la représentation de l'interaction et qu'il est donc important de la considérer comme un point de modélisation à part entière, comme le montre très bien l'expérience de Edmonds et Hales. L'idée principale est ici de fournir des éléments de réflexion généraux qui, en intégrant les spécificités du paradigme agent, permettent effectivement d'étudier des aspects de la relation de modélisation qui ne soient pas uniquement liés au domaine d'application. Il est aujourd'hui fondamental d'essayer d'exhiber des principes de modélisation multi-agents qui soient applicables quelle que soit la nature du modèle considéré. Dans cette optique, nous allons ici proposer une réflexion sur une notion centrale du paradigme agent : l'autonomie. Une fois que nous aurons clairement défini ce que nous comprenons de cette notion, nous disposerons alors d'un élément d'appréciation objectif qui nous permettra de proposer une analyse de la validité du module des interactions suivant un nouveau principe de cohérence que nous introduirons : la *cohérence paradigmatique*.

7.4 Retour sur la notion d'autonomie

7.4.1 Interprétations classiques

L'une des notions les plus importantes du paradigme agent est la propriété d'autonomie que l'on attribue à ce type d'entité (cf. section 3.1.1 page 45). A l'instar des autres propriétés fondamentales d'un agent, l'autonomie est une caractéristique qui possède de nombreuses interprétations suivant le contexte où elle est considérée. Gouaïch identifie deux approches différentes dans la manière dont cette propriété est comprise dans la littérature [Gouaïch, 2003].

La première est associée au point de vue de Steels qui propose d'envisager cette propriété suivant une perspective biologique [Steels, 1995]. Son idée est que l'autonomie est une propriété inhérente de tout être vivant et qu'elle peut donc être attribuée à une entité qui exhibe un comportement qui s'en rapproche :

> "*It starts from the idea that agents are self sustaining systems which perform a function for others and thus get the resources to maintain themselves. But because they have to worry about their own survival they need to be autonomous, both in the sense of self-governing and of having their own motivations.*"

Ici, la notion d'autonomie est donc considérée comme une conséquence de l'instinct de survie. Pour un agent logiciel, il s'agit d'accomplir ses propres buts en assurant lui-même la pérennité de son fonctionnement. Il doit notamment être capable de s'adapter à un environnement extérieur qui évolue. L'idée fondamentale de cette approche est de considérer qu'un agent

est une entité autonome car il possède un comportement proactif **dirigé par ses propres buts** (*goals-directed behaviour*). Sur cette idée, [Guessoum & Briot, 1999] ont par exemple proposé une architecture agent basée sur une extension d'un système de programmation par objets concurrents (ou objets actifs) de manière à obtenir des entités qui soient capables de contrôler leur comportement et de raisonner sur les tâches qu'elles doivent accomplir.

La deuxième approche consiste à définir l'autonomie d'un agent dans le contexte du réseau social auquel il appartient. [Sichman *et al.* , 1994] introduit pour cela la notion de réseau de dépendance sociale (*social dependence network*). Bien que le propos ne soit pas de définir l'autonomie d'une manière générale, ce travail consiste à utiliser cette notion pour évaluer le degré d'indépendance dont un agent dispose vis-à-vis de ses congénères pour réaliser ses buts[2]. L'approche distingue ainsi plusieurs niveaux d'autonomie pour un agent suivant son indépendance en termes de ressources ou de compétences. Un agent est considéré complètement autonome s'il ne dépend d'aucune autre entité pour accomplir ses propres objectifs. Dans ce contexte, l'autonomie est donc synonyme d'**indépendance**. De nombreux travaux utilisent cette interprétation.

7.4.2 Nécessité de définir un point de vue computationnel

Comment faut-il comprendre ces différentes approches du point de vue de l'implémentation concrète d'un agent ? L'indépendance d'un agent vis-à-vis de ses ressources d'exécution semble être la notion la plus simple à identifier. Le fait qu'un agent s'exécute sur un système informatique indépendant par exemple [Balasubramaniyan *et al.* , 1998]. La vision de Steels est plus abstraite car elle repose sur une appréciation qualitative du comportement de l'entité considérée. Steels conclut d'ailleurs son article par le constat qu'il n'existe aucun système qui puisse être considéré comme un agent véritablement autonome. Malgré cela, cette vision a l'intérêt d'être plus générale car elle ne dépend pas de la relation que l'individu entretient avec les autres entités du système. Elle permet ainsi de définir l'autonomie en considérant uniquement des caractéristiques liées à la nature du fonctionnement interne de l'entité. Ce qui, du point de vue de l'ingénierie logicielle, semble plus avantageux car il n'est pas nécessaire de considérer l'ensemble du système dans lequel l'agent est plongé. L'autonomie reste ainsi une propriété individuelle et il est théoriquement possible de dire si un agent est véritablement autonome en analysant uniquement son fonctionnement interne. C'est pourquoi nous retiendrons cette interprétation. Tout le problème est alors de savoir comment cette analyse doit être réalisée et, surtout, quels sont les critères objectifs qui permettent de déterminer si une entité logicielle est effectivement autonome.

Pour établir de tels critères, il est donc nécessaire de définir l'autonomie du point de vue de l'implémentation [Weiss *et al.* , 2003]. Sans quoi l'autonomie reste une notion subjective. Pour cela, on ne peut pas dériver des principes d'implémentation directement de la définition de Steels. En effet, il n'est pas possible de spécifier une fois pour toute ce que doit être une implémentation qui exhiberait un comportement autonome au sens de Steels. Cependant, on peut extrapoler certaines conséquences de cette vision pour l'implémentation. En particulier le fait qu'un agent soit autonome implique certaines contraintes quant à la manière dont ses variables d'état sont manipulées. Ainsi, une entité logicielle qui ne possède pas un contrôle total sur les variables (et fonctions) qui modélisent son état interne ne peut être considérée comme autonome car ces variables ne doivent faire l'objet d'aucune intervention extérieure. Autre-

[2]Comme le note Gouaïch, l'approche utilisée impose une uniformité en ce qui concerne le modèle de description utilisé par un agent pour se représenter les autres. Ce qui n'est pas une hypothèse valide dans le cadre des systèmes ouverts où la nature des agents est a priori inconnue.

ment dit, rien ne doit pouvoir modifier le déroulement du processus décisionnel d'un agent. Sa délibération doit être entièrement déterminée par sa seule perception. Dans le contexte de l'ingénierie logicielle, Gouaïch parle de l'*intégrité interne* de l'agent. Celui-ci explique que l'intégrité interne de la structure logicielle (données et traitements) d'un agent est une condition nécessaire à son autonomie : si cette structure est accédée ou modifiée par un autre agent, alors la propriété d'autonomie est perdue. Nous adhérons totalement à ce point de vue. En fait, si l'intégrité interne d'un agent n'est pas respectée, il faut en conclure que l'agent n'est pas maître de son destin et qu'il n'est donc pas autonome. Cette interprétation de la notion d'autonomie a l'avantage d'être entièrement fondée sur des critères purement informatiques.

Concrètement, si l'on applique la distinction esprit/corps, les variables d'état qui sont utilisées dans le processus décisionnel d'un agent ne doivent être en aucun cas accédées ou modifiées par une entité externe, que ce soit l'environnement ou un autre agent. De plus, les fonctions qui utilisent ces données ne doivent elles aussi faire l'objet d'aucune influence extérieure : le respect de la délibération d'un agent concrétise son autonomie. La figure 7.4 illustre cette interprétation de l'autonomie.

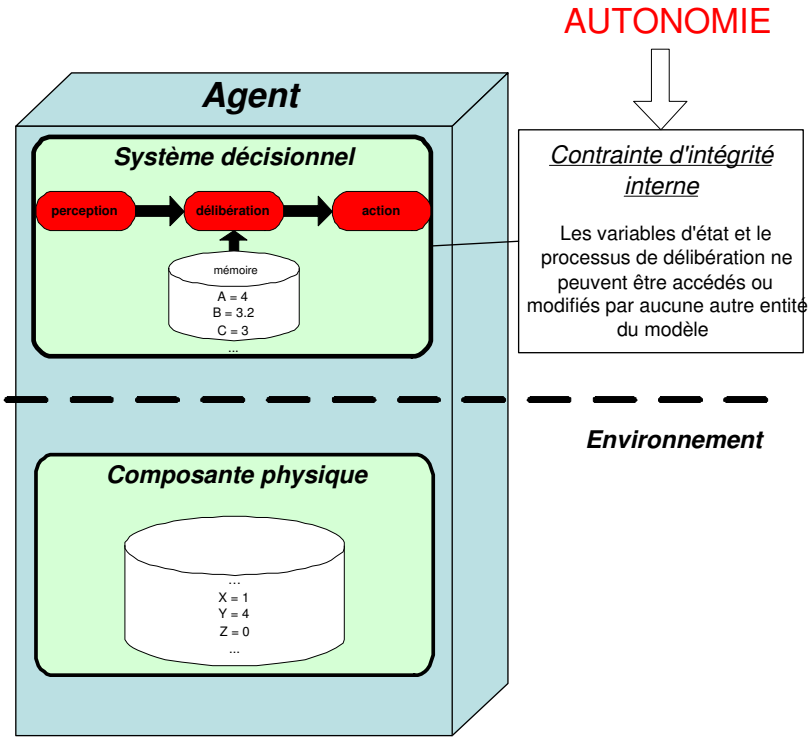

FIG. 7.4 – L'intégrité interne d'un agent : condition nécessaire à son autonomie.

7.4.3 Un agent simulé peut-il être autonome ?

L'interprétation de la propriété d'autonomie que nous venons de donner doit nous faire poser une question. Peut-on considérer que les agents qui sont exécutés dans un simulateur sont des entités autonomes ? A priori, ces entités logicielles semblent très loin d'être douées d'une quelconque autonomie. En effet, tous les agents d'une simulation sont exécutés et contrôlés par une autre entité : l'ordonnanceur. En fait, dans le cadre de la simulation, pour que l'intégrité interne d'un agent ne soit pas remise en cause, il suffit de garantir que son architecture interne

n'est pas accédée ou modifiée par un autre agent. Peu importe la manière dont il est exécuté, ce qui compte c'est la manière dont il est effectivement simulé : il doit être autonome dans le modèle. Ainsi, de notre point de vue, le respect de l'intégrité interne d'un agent est une condition suffisante pour qu'un agent soit considéré comme effectivement autonome dans la modélisation.

Par ailleurs, il est très intéressant de remarquer que cette vision de la propriété d'autonomie est entièrement consistante avec tout ce que nous avons dit jusqu'à présent. Tout d'abord, elle constitue un nouvel argument en faveur d'une séparation explicite entre le corps et l'esprit d'un agent. En effet, il s'agit encore une fois de faire une distinction claire entre, d'un côté les variables sur lesquelles l'agent a un contrôle total et qui font partie de son système décisionnel, et de l'autre les variables qui modélisent l'aspect physique d'un agent et sur lesquelles celui-ci n'a aucun pouvoir de modification directe.

De plus, elle est aussi une conséquence logique de la contrainte d'intégrité environnementale que nous avons exposée dans le chapitre 3 (section 3.4.2 page 62) et qui stipule qu'un agent ne doit pas être en mesure de modifier directement les variables d'état de l'environnement. Rappelons ici que pour un agent donné, tous les autres agents font partie de l'environnement. C'est pourquoi il est tout à fait naturel qu'un agent ne puisse pas accéder à la structure des autres entités. Ainsi, les variables d'état représentant le physique d'un agent ne peuvent être modifiées par aucun agent, y compris l'agent modélisé par ces variables.

7.5 Cohérence paradigmatique

7.5.1 Motivations

La réflexion que nous venons de conduire sur l'autonomie nous donne un moyen concret d'étudier la modélisation d'un système multi-agents suivant une nouvelle perspective. Comme nous l'avons dit, bien que l'analyse de la validité d'un modèle que l'expert du domaine réalise soit indispensable, elle est cependant insuffisante. En effet, c'est oublier que le cadre expérimental d'une simulation multi-agents ne se réduit pas au système source que l'on souhaite étudier. Celui-ci s'inscrit dans un contexte plus général : le paradigme agent lui-même. Et ce paradigme définit des contraintes de modélisation qu'il faut prendre en compte lorsqu'on élabore un modèle de simulation. Nous allons bien sûr nous intéresser ici plus particulièrement aux problèmes liés à la question de la modélisation de l'autonomie.

Notre idée est la suivante : si la modélisation des interactions ne respecte pas l'autonomie des agents, alors celle-ci n'est pas cohérente avec le cadre expérimental (le paradigme agent) et la relation de modélisation n'est pas vérifiée. Nous définissons ainsi un nouvel aspect de la validation des systèmes multi-agents qui consiste à étudier ce que nous appelons la *cohérence paradigmatique*. Pour constituer notre propos, nous allons maintenant étudier cette problématique dans le cadre des trois modélisations de l'interaction de reproduction que nous avons présentées dans la section 7.2.2 (page 142).

7.5.2 Violations de l'autonomie

Bien que les deux premiers modèles de la reproduction proposent des implémentations différentes de cette interaction, ils sont tous les deux critiquables en ce qui concerne le respect

de l'autonomie décisionnelle des agents. Pour s'en rendre compte, examinons dans le détail la modélisation de l'interaction qui conduit au processus de reproduction.

Que ce soit dans le premier ou le deuxième modèle, lorsqu'un agent décide de se reproduire, il accède alors directement aux variables d'état du partenaire afin de signifier la concrétisation de la reproduction. Dans le premier cas, cela revient à lui ôter un certain nombre de points de vie. Dans le deuxième, l'idée proposée par Lawson et Park pour rendre le modèle précédent plus "réaliste" consiste à faire en sorte que le partenaire sélectionné par un agent ne puisse pas se reproduire lorsque viendra son tour. Pour cela, l'agent initiateur modifie encore une fois directement une variable d'état du partenaire. Par ailleurs, pendant toute la durée de la gestation, les deux agents ne peuvent pas bouger. Comme nous l'avons dit, l'accès direct aux variables d'état est en soi une violation de l'autonomie telle que nous la concevons. Nous faisons donc ici une constatation technique du non-respect de la propriété d'autonomie.

En outre, le plus intéressant est que ce constat purement technique correspond effectivement à une perte d'autonomie au sens de Steels dans le modèle conceptuel. En effet, que faut-il penser du fait que l'agent qui a été sélectionné n'ait pas eu son mot à dire sur sa participation au processus de reproduction ? En fait, il faut remarquer que son comportement a été annihilé par la décision d'un autre agent qui a donc décidé pour lui. Autrement dit, **ses propres buts** ne sont à aucun moment pris en compte. Rappelons que dans les deux cas, la reproduction implique une perte de points de vie. Peut-être que cet agent aurait préféré faire une autre action pour une raison vitale. Est-il alors normal qu'il se retrouve embrigadé dans un processus interactionnel qu'il n'a pas explicitement choisi ? Peut-il encore être considéré comme une entité autonome ? Ce qui est sûr, c'est qu'il n'a pas été ici en mesure de délibérer sur sa propre survie.

7.5.3 Quand l'habit fait le moine

Outre la perte d'autonomie que nous avons constatée pour l'agent qui subit l'interaction, il est aussi très intéressant d'analyser ces modélisations du point de vue de l'agent initiateur. En effet, comment faut-il interpréter le fait que celui-ci dérive de sa perception que l'agent cible est tout à fait d'accord pour se reproduire ?

Dans les deux modèles, un agent sélectionne son partenaire étant donné ses caractéristiques physiques. Autrement dit, il vérifie que celui-ci est compatible étant donné la perception qu'il en a. A partir de ce percept, il va alors prendre la décision de déclencher le processus de reproduction et ainsi d'impliquer le deuxième agent dans celui-ci. Ce que l'on fait ici n'est ni plus ni moins que d'associer l'apparence physique d'un agent avec un comportement attendu. Il y a là un véritable problème : est-il satisfaisant de considérer qu'un agent, parce qu'il présente certaines caractéristiques extérieures, doit nécessairement agir d'une certaine façon ? En tout cas, c'est effectivement ce qui se passe ici.

Finalement, on voit bien que le fait que l'interaction de reproduction soit déclenchée par la décision d'un seul agent est un véritable problème du point de vue de la cohérence paradigmatique du modèle. L'approche multi-agents voudrait que ce genre d'interaction soit le fruit de deux comportements distincts et non le fait d'une seule entité. Dans les deux cas, on ne se trouve pas en présence d'un système dont la dynamique correspond à la composition de comportements individuels et concurrents. Autrement dit, bien que le paradigme agent ait été explicitement utilisé pour modéliser le système, les spécifications du modèle ne sont pas en accord avec celui-ci.

7.5.4 Y a-t-il un agent dans la simulation ?

C'est pourquoi, lorsque les auteurs de [Drogoul *et al.* , 2002] interpellent la communauté pour lui demander *où sont passés les agents dans les simulations orientées agents*, on ne peut que leur donner raison. Ceux-ci expliquent qu'il est effectivement très rare de trouver dans l'implémentation d'une simulation multi-agents des entitées qui aient les caractéristiques qu'on leur prête dans le modèle conceptuel :

> "*Agents can be found in the domain model, as metaphors of autonomous, proactive, communicating and interacting "individuals" of the target system. They can also be found in the design model, as a conceptual support used to formalize the definition of these individuals in terms of behaviors, communication models, etc. But, as far as we know, they cannot be found, yet, in any of the operational models presented so far in the conferences dedicated to MABS. Instead of using agent-oriented languages, people tend to use either object-oriented or logical, procedural or functional languages to implement the specifications described in the conceptual agents. This means that the resulting "computational agents" (if we can still call them this way) do not possess any of the properties generally assigned to the agents used in MAS or DAI : they do not have any structural nor decisional autonomy, are not proactive, and cannot, for instance, modify their knowledge and behaviors through learning or adaptation.*"

[Drogoul *et al.* , 2002] expliquent que cet état de fait est en majeure partie une conséquence du manque de spécifications dont la notion d'agent est victime, notamment en ce qui concerne son implémentation. Nous avons d'ailleurs déjà cité cet article dans la section que nous avons consacrée aux problèmes de l'interdisciplinarité (cf. section 4.3.2, page 85). L'absence de définitions claires pour les différentes propriétés d'un agent est encore une fois à l'origine du problème.

Cela nous conforte dans notre idée qu'il est aujourd'hui fondamental de proposer des interprétations purement informatiques à ces différentes propriétés, comme nous l'avons fait ici pour l'autonomie. Vouloir définir une fois pour toute ce qu'est l'autonomie d'un agent peut sembler réducteur mais nous sommes convaincu que c'est au prix de telles démarches que le paradigme agent deviendra un véritable outil de modélisation, et plus seulement un concept métaphorique.

7.5.5 Respect de l'autonomie grâce au principe Influence/Réaction

Si l'autonomie des agents doit être respectée, on voit bien que l'interaction de reproduction nécessite un modèle de l'action plus évolué. Dans les deux premières modélisations, le non-respect de l'autonomie est une conséquence du fait qu'un agent puisse décider seul de la concrétisation de l'interaction et de la modification environnementale correspondante (un nouvel agent). Autrement dit, et c'est ce qu'il faut ici retenir, la représentation de l'action comme une transformation d'un état global ne permet pas de respecter ici l'intégrité interne des agents. Encore une fois, un agent n'est pas une entité en mesure de calculer la conséquence de ses actes sur l'environnement. En conséquence, pour que l'interaction de reproduction soit effectivement le fruit du comportement de deux entités autonomes distinctes, il est nécessaire de considérer l'ensemble des comportements avant de pouvoir décider de leurs résultats.

Comme le lecteur l'aura deviné, nous pensons que seule l'application du principe Influence/Réaction permet de modéliser de manière cohérente cette interaction, c'est-à-dire en respectant le principe d'autonomie.

En effet, dans la troisième modélisation que nous avons présentée, l'autonomie des agents est garantie. Premièrement car les agents n'ont pas le pouvoir de modifier directement leur environnement, et donc a fortiori les variables d'état des autres agents. Deuxièmement car le processus décisionnel d'un agent n'est jamais remis en cause par l'action d'un autre agent. Son comportement sera toujours pris en compte. En effet, pour calculer le résultat de l'interaction, on prend d'abord en compte les autres influences produites au même instant pour décider, dans un deuxième temps, du résultat de leur composition, ce qui constitue la *réaction* de l'environnement à l'ensemble de ces influences.

Par exemple, si de deux agents, le premier souhaite bouger, on peut poser qu'un comportement de reproduction de la part du deuxième n'entraînera pas de naissance. Autrement dit, **on ne décide pas à la place des agents mais on décide de la dynamique du système**.

Par ailleurs, dans le cadre du modèle minimal que nous avons présenté dans la section 7.2.2, on peut faire une constatation très intéressante en ce qui concerne les résultats obtenus. Un calcul de probabilité simple montre que seul le dernier résultat correspond, dans un langage probabiliste, à la composition de deux événements indépendants. Ce qui correspond pour nous au fait que les deux agents ont effectivement délibéré de manière autonome, et donc indépendamment l'un de l'autre. En effet, si on considère des agents **autonomes**, le processus décisionnel de l'agent A est indépendant de celui de B. Ainsi la probabilité de l'événement *naissance* correspond à une probabilité composée où les événements A_{repro} et B_{repro} sont indépendants :

$$\begin{aligned} Pr(naissance) &= Pr(A_{repro}) \textbf{ et } Pr(B_{repro}) \\ &= Pr(A_{repro}) \times Pr(B_{repro}) \\ &= \alpha\beta \end{aligned}$$

Notre propos n'est pas d'expliquer que cette modélisation donne des résultats qui doivent être nécessairement jugés plus valides d'un point de vue quantitatif ou conceptuel. Il ne s'agit pas de dire que cette modélisation est plus correcte que les deux autres relativement à une hypothétique réalité. Par contre, là où nous prenons clairement position, c'est sur l'inadéquation des deux premières modélisations avec le paradigme qu'elles utilisent. Autrement dit, elles considèrent un cadre expérimental sans en tirer les conséquences sur la nature de la modélisation qui doit en découler. De notre point de vue, cela invalide l'approche utilisée pour modéliser le système.

Il s'agit là d'un point très délicat et il y a évidemment matière à discussion car notre raisonnement se fonde sur la définition que nous avons choisie pour l'autonomie. Cependant, nous avons vu tout au long de ce document que nous ne sommes pas les seuls à penser qu'il est aujourd'hui fondamental que la simulation orientée agents se donne des moyens méthodologiques et formels d'établir une correspondance concrète entre, des **principes de modélisation** d'un côté, et des **principes d'implémentation** de l'autre. D'où la notion de cohérence paradigmatique que nous avons proposée.

Bien sûr, la cohérence paradigmatique ne constitue pas à elle seule une condition suffisante pour la validation et elle ne doit pas occulter les autres cohérences liées à cette problématique. Cependant nous pensons qu'elle doit être en première ligne car elle définit des contraintes objectives qui ne dépendent pas d'un domaine d'application en particulier.

7.6 Interaction faible et interaction forte dans les systèmes multi-agents

7.6.1 Gestion systématique de la simultanéité : solution ultime pour la modélisation de l'interaction ?

Dans la section précédente, nous avons vu que seule l'application d'un principe Influence/-Réaction permet de modéliser l'interaction de reproduction sans que l'autonomie des agents soit remise en cause. Notamment car il s'agit de prendre en compte l'ensemble des comportements pour calculer leurs conséquences sur l'environnement. Dans le cas de l'expérience minimale que nous avons conduite, nous avons considéré qu'il est nécessaire que deux agents souhaitent se reproduire pour que l'interaction de reproduction soit validée. Autrement dit, le fait que deux agents souhaitent se reproduire constitue dans ce système un événement distingué qui aboutit à une situation particulière : la naissance d'un agent. C'est donc dans la **prise en compte de la simultanéité des comportements** que toute la modélisation de ce système se joue : dans le troisième modèle elle est effective alors qu'elle est ignorée dans les deux autres.

Cette expérience soulève une question : la prise en compte de la simultanéité des événements joue-t-elle systématiquement un rôle crucial relativement à la qualité de la modélisation et aux résultats obtenus ? En effet, on peut déduire de l'expérience précédente que celle-ci est nécessaire pour une modélisation correcte et que les résultats sont très différents suivant qu'elle est appliquée ou non. Dans le cas de l'interaction de reproduction, nous avons vu à quel point la gestion des comportements produits à un même instant t influence la nature des résultats suivant la technique employée. Il convient donc de se demander si ce lien de cause à effet est généralisable ou s'il n'est lié qu'à ce modèle en particulier.

7.6.2 Deuxième expérience : consommation d'une ressource commune par deux agents

Pour répondre à la question précédente, nous allons maintenant présenter une deuxième expérience qui met en jeu une autre interaction : la consommation d'une ressource. Encore une fois, notre problème sera ici d'étudier les différentes possibilités qui s'offrent à nous lorsque deux agents souhaitent consommer la même ressource R à un même instant t de la simulation. Nous allons donc ici aussi considérer un modèle minimal basé sur un modèle du temps discret qui va nous permettre de nous concentrer uniquement sur ce problème.

Description d'un modèle minimal

Soient deux agents A et B. Chaque agent possède une variable d'état représentant un nombre de points de vie notée E telle que $0 <= E <= max\_E$ et où max_E représente le maximum d'énergie qu'un agent peut avoir. De plus, cette énergie décroît aléatoirement d'une valeur comprise entre 0 et 10 à chaque pas de temps. Pour survivre, chaque agent a donc pour but de consommer la ressource afin de maintenir son énergie au dessus d'un certain seuil S initialement fixé tel que $0 < S < 100$. Ainsi, dès que l'énergie d'un agent passe en dessous du seuil, celui-ci consomme la ressource pour régénérer ses points de vie le plus possible. La figure 7.5 illustre ce modèle. Dans ce modèle très simple, un agent exhibe uniquement deux comportements suivant qu'il se trouve au dessus ou en dessous du seuil : *consommer la ressource* ou

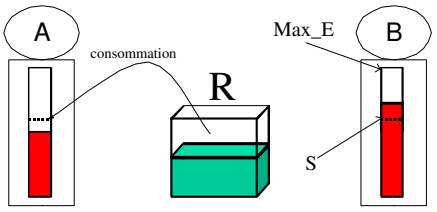

FIG. 7.5 – Consommation d'une ressource accédée par deux agents.

ne rien faire, ce que nous noterons respectivement par $Agent_{consommer}$ et $Agent_{rien}$. Lorsque l'énergie d'un agent atteint 0 la simulation s'arrête. Par ailleurs, la ressource est régénérée à chaque pas de temps d'un montant α. Dans ce cadre, nous allons mesurer la moyenne de l'énergie des agents suivant différentes configurations initiales.

Implémentation des différents modules

Encore une fois, nous ne ferons varier ici que la modélisation de l'interaction et les trois autres modules resteront les mêmes tout au long de l'expérience.

- **Ordonnancement** : pour le module d'ordonnancement, nous utilisons une discrétisation régulière du temps où la variable temporelle t est incrémentée d'une unité pour chaque pas de temps.
- **Comportements** : le comportement d'un agent est de produire une influence étant donné son niveau d'énergie E : $Agent_{consommer}$ si $E < S$ ou $Agent_{rien}$ sinon.
- **Environnement** : contrairement à la première expérience, cette fois le module représentant l'environnement n'est pas inexistant, bien que toutefois très simple. Il correspond simplement à la modélisation de la régénération de la ressource. Soit R(t) la valeur de la ressource et $Agent_{consommation}(t)$ la quantité de la ressource effectivement consommée par un agent, la dynamique de l'environnement correspond à l'application de l'équation suivante : $R(t+1) = R(t) - A_{consommation}(t) - B_{consommation}(t) + \alpha$.
- **Interaction** : étant donnés le module des comportements et le module d'ordonnancement que nous avons choisis, la gestion de l'interaction consiste à gérer les quatre situations d'interaction qui sont possibles pour chaque instant t de la simulation. Ces différentes situations sont résumées dans le tableau 7.6.2. Pour étudier l'impact que peut avoir la gestion de l'interaction sur les résultats, nous avons utilisé successivement deux modules interactionnels différents.
  Le premier consiste à sélectionner aléatoirement l'ordre dans lequel les deux agents ont accès à la ressource et à valider directement leurs actions. Autrement dit, la ressource effectivement consommée par chaque agent est calculée comme s'il était le seul à agir à cet instant précis. Ainsi, une fois la ressource diminuée par le premier agent considéré, le deuxième pourra à son tour consommer la ressource mais uniquement sur la base de ce qu'il reste. On procède ainsi par modification directe de l'état global du système.
  La deuxième méthode que nous avons employée correspond à l'esprit du principe Influence/Réaction où la réaction de l'environnement est calculée en fonction de l'ensemble des comportements. Dans ce cadre, si les deux agents souhaitent consommer la ressource au même instant, celle-ci est équitablement partagée en fonction des desiderata de chacun. Autrement dit, nous avons ici le souci de prendre en compte la simultanéité les actions dans le calcul de leurs résultats.

TAB. 7.6 – Les quatre situations interactionnelles possibles pour un instant t

$A_{consommer}$	and	$B_{consommer}$
$A_{consommer}$	and	B_{rien}
A_{rien}	and	$B_{consommer}$
A_{rien}	and	B_{rien}

Résultats

Pour chacune des deux versions du modèle, nous avons effectué un nombre significatif de simulations pour différentes valeurs initiales de S et α. Notamment, il convient de distinguer les situations suivant la valeur du paramètre α. En effet, lorsque la valeur du paramètre α est suffisamment grande pour couvrir les besoins des deux agents, on se trouve dans une situation d'abondance de la ressource par opposition à une situation de compétition lorsque celle-ci ne se régénère pas suffisamment pour subvenir, en moyenne, aux besoins des deux agents.

En situation d'abondance, les résultats obtenus étaient tout à fait prévisibles. Quelle que soit la gestion de l'interaction employée, on obtient exactement la même valeur pour l'espérance de vie des deux agents. Par contre, nous avons été surpris de constater le même résultat pour des valeurs faibles de α. En fait, nous nous attendions à trouver une bien meilleure espérance de vie dans le cas de l'application du principe Influence/Réaction étant donné le partage équitable qui est réalisé sur la ressource. Il n'en a rien été. Les résultats ont été rigoureusement équivalents dans les deux cas. Dans cet exemple, et contrairement à nos attentes, la prise en compte de la simultanéité d'un vif n'a donc pas influencé les résultats.

7.6.3 Nécessité de distinguer plusieurs types d'interaction

Suivant les deux expériences que nous avons présentées dans ce chapitre, il nous faut essayer d'expliquer pourquoi il existe une telle différence entre les impacts que la gestion des interactions peut avoir sur un modèle multi-agents. En effet, dans le cadre du premier modèle, une grande partie de notre raisonnement a été de démontrer l'importance qu'il y a à considérer la simultanéité des comportements. Nous en avons même conclu que la gestion de la simultanéité était une condition nécessaire au respect de l'autonomie des agents. Il est donc assez embarrassant de constater que la gestion de de la simultanéité n'a eu aucun effet sur les résultats dans la deuxième expérience.

Pour expliquer ces résultats, il est une nouvelle fois très intéressant d'étudier le deuxième modèle du point de vue de sa cohérence paradigmatique. En fait, il faut remarquer que quelle que soit la technique utilisée pour gérer l'accès des deux agents à la ressource, l'autonomie des agents n'est jamais remise en cause. En effet, même lorsque les deux agents sont activés séquentiellement, les processus décisionnels ne sont pas affectés par une décision extérieure et les agents n'accèdent pas à des variables d'état qui ne sont pas leurs propriétés. Autrement dit, alors que dans la première expérience, la gestion de la simultanéité des comportements est une condition sine qua non du respect de l'autonomie, celle-ci n'apporte absolument rien du point de vue de la cohérence paradigmatique dans le deuxième modèle. Il y a donc ici une différence fondamentale entre ces deux interactions. Il nous faut maintenant identifier quels sont les fondements de cette différence afin de généraliser cette conclusion.

Dans cette optique, nous pensons qu'il peut être intéressant de distinguer plusieurs sortes d'interaction suivant la manière dont elles doivent être gérées dans la modélisation. Notre idée est que, du point de vue de la cohérence paradigmatique d'un modèle, certaines interactions **nécessitent** de considérer l'ensemble des comportements simultanément alors que d'autres non. Nous sommes ici à un point crucial de notre analyse. En fait, la véritable question n'est pas celle de la modélisation de la simultanéité de deux événements en particulier. Le problème n'est pas d'apporter systématiquement une réponse à la composition de deux événements produits par le système à un même instant t, mais de comprendre que certaines interactions n'ont de sens que dans la simultanéité des comportements qu'elles impliquent. Une tentative de reproduction n'a de sens que lorsqu'un autre agent ayant les capacités requises est présent à proximité. C'est un comportement dont le **but** ne peut être réalisé que s'il existe une conjonction entre plusieurs comportements. C'est pourquoi nous considérons qu'il s'agit ici d'une *situation d'interaction forte*. A contrario, consommer une ressource est une action dont la réalisation n'implique pas les choix qui sont faits par une autre entité, que l'agent soit satisfait ou non. La présence d'un autre agent n'est pas nécessaire pour justifier un tel comportement. C'est pour cette raison que dans ce deuxième cas, l'autonomie des agents n'est pas remise en cause par la gestion des interactions. Ce que nous identifions par une *situation d'interaction faible*. Autrement dit, bien qu'il soit possible d'implémenter une solution à la simultanéité de l'accès à la ressource, cela n'apporte rien du point de vue de la cohérence paradigmatique.

En soulevant la question d'une différence forte entre ces deux types d'interaction, nous souhaitons ici fournir des éléments de réflexion qui permettent de guider la modélisation et de cerner les techniques qui doivent être utilisées suivant les cas [Michel *et al.* , 2003]. A ce propos, on peut trouver une démarche similaire dans d'autres travaux récents comme ceux réalisés par Weyns et Holvoet sur les actions simultanées [Weyns & Holvoet, 2003a] (cf. section 6.4.6 page 129). Dans un contexte plus général, [Parunak *et al.* , 2003] propose aussi une analyse poussée des différents modes d'interaction qui peuvent exister entre agents. Dans cet article, il faut notamment retenir la notion de *corrélation* entre plusieurs agents qui définit un ensemble d'*informations comportementales communes* (*behavioral joint information*). Des agents sont considérés comme *corrélés* si leurs actions (les résultats correspondants dans le sens de l'article) sont statistiquement dépendantes des actions des autres. Par rapport à ces travaux, notre analyse propose un point de vue supplémentaire dans la mesure où nous focalisons ici toute notre attention sur le problème de la modélisation de l'autonomie et sur l'étude de la cohérence paradigmatique.

7.6.4 Interaction forte

Nous proposons ainsi de définir une situation d'interaction forte de la façon suivante :

Définition 7.3 (Interaction forte) *Plusieurs actions constituent une situation d'interaction forte lorsque la réalisation du but de chaque agent dépend de l'action d'autres agents. Pour être réalisé, l'événement correspondant nécessite la conjonction de plusieurs comportements autonomes particuliers. Le résultat d'une telle interaction ne peut être calculé sans prendre en compte la délibération de l'ensemble des agents concernés, sans quoi leur autonomie décisionnelle ne peut être garantie.*

Ainsi, une interaction forte est définie relativement à la qualité du résultat qu'elle engendre sur l'environnement plutôt que sur la nature des actions qu'elle implique. Cette définition a par ailleurs une conséquence directement observable sur le système : le résultat (en cas de

succès) d'une interaction forte ne peut être observé si le nombre d'agents présents dans le système n'est pas suffisant. Par exemple, si un seul agent est présent dans le système, aucune naissance ne sera observée. Du point de vue de l'agent, celui-ci ne tentera d'ailleurs jamais de se reproduire car le but recherché est impossible à satisfaire.

Il est intéressant de rapprocher cette notion d'interaction forte avec ce qui a pu être proposé dans d'autres contextes que celui de la simulation. Par exemple, dans le cadre de l'intelligence artificielle distribuée, de nombreux travaux utilisent des notions d'*actions/intentions jointes* (*joint action/intention*) proposée par [Cohen & Levesque, 1991]. Cependant, les problématiques soulevées par ces travaux sont en fait assez éloignées de nos préoccupations car fortement orientées sur des questions liées à la coordination d'agents cognitifs[3]. Cependant, on peut tirer de ces travaux certaines réflexions clés qui gardent tout leur sens dans le contexte nos travaux. Dans [Cohen & Levesque, 1991] par exemple :

> "*What is involved when a group of agents decide to do something together ? Joint action by a team appears to involve more than just the union of simultaneous individual actions, even when those actions are coordinated.*"

7.6.5 Interaction faible

La définition de l'interaction faible est complémentaire de la précédente :

Définition 7.4 (Interaction faible) *Plusieurs actions constituent une interaction faible lorsque le but de chaque agent peut être réalisé indépendamment des autres actions. Dans cette situation, bien que ces actions puissent interférer entre elles, et quelle que soit la gestion des interactions, l'autonomie des agents n'est pas remise en cause par la technique utilisée.*

Une telle interaction est un processus radicalement différent du précédent. Dans le cas de la consommation d'une ressource, même si les agents interagissent au travers de l'environnement en consommant la ressource, ils produisent des influences qui ne sont pas directement corrélées. La présence des autres agents dans le système n'est pas nécessaire pour pouvoir consommer la ressource. La faisabilité du but recherché par cette action n'est pas conditionnée par les actions des autres agents. Cela ne veut pas dire qu'il s'agisse d'une interaction foncièrement moins complexe ou plus facile à traiter. Comme nous l'avons fait, il est par exemple possible de considérer que les actions sont véritablement simultanées en ce qui concerne l'accès à la ressource et de proposer une solution particulière à cette situation. Cependant, traiter explicitement ce cas de figure ne modifie pas fondamentalement le vrai sens du modèle du fait de la relation faible qui existe entre les agents. En effet, peu importe la gestion des interactions, la propriété d'autonomie est toujours respectée car les deux comportements, c'est-à-dire le processus de délibération des agents, ne sont pas modifiés ou influencés par les autres. Même s'ils sont considérés de façon séquentielle, l'intégrité du processus décisionnel de chaque agent est effective et la cohérence paradigmatique n'est pas remise en cause.

Un très bon exemple de ce type d'interaction peut être étudié dans le modèle de simulation très connu des termites [Resnick, 1994]. Dans ce modèle, le système consiste dans un ensemble d'agents, des termites, et un ensemble de brindilles aléatoirement réparties dans un environnement discrétisé sous la forme d'une grille. Le comportement des termites suit un ensemble de règles très simples qui peuvent être résumées en deux points :

[3] Les problématiques abordées reposent d'ailleurs en majeure partie sur l'utilisation d'un modèle d'architecture interne de type BDI.

– Règle 1 : si je ne porte pas de brindille, j'en cherche une dans une direction aléatoire.
– Règle 2 : si je porte une brindille, je cherche une autre brindille pour la poser à côté.

La figure 7.6 montre quatre étapes successives d'une simulation de ce modèle où l'on peut voir qu'après un certain nombre d'itérations on obtient un seul tas de brindilles (la grille est toroïdale).

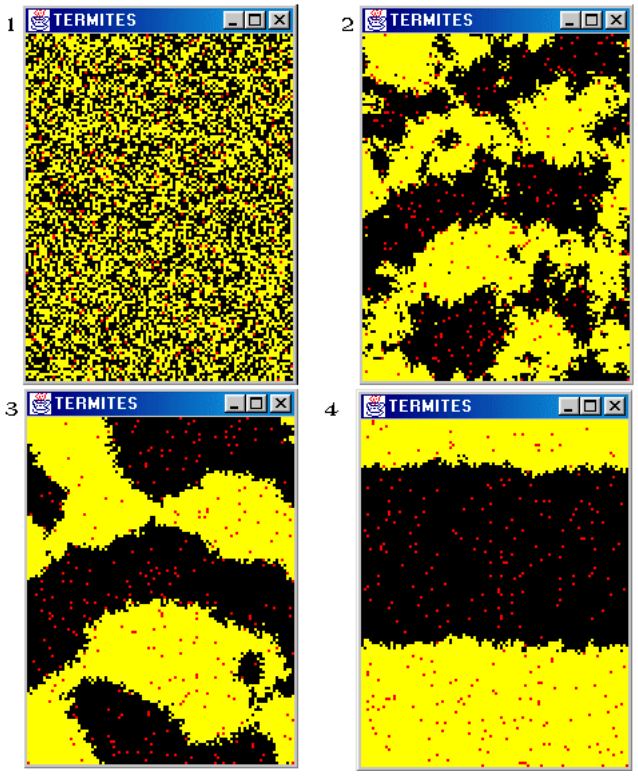

FIG. 7.6 – De l'interaction des termites émerge un seul tas de brindilles.

Dans ce modèle, l'ensemble des influences produites par les termites se réduit à des actions de mouvement et à la prise/dépôt d'objets dans l'environnement. On peut faire plusieurs remarques sur le comportement de ces termites électroniques. Aucune des actions effectuées par un agent ne suppose la présence d'une autre entité. Une termite n'a aucune représentation de ses congénères : pour elle, les autres n'existent pas. Par ailleurs, nous avons pu vérifier que le résultat est atteint quelle que soit la technique utilisée pour modéliser l'écoulement du temps, par événement ou par discrétisation. De plus, le résultat le plus intéressant est sans aucun doute le fait que le résultat final peut être obtenu même si une seule termite se trouve dans le système. Elle n'a pas besoin des autres pour aboutir à ce résultat. Il suffit d'être un peu plus patient.

7.6.6 Interprétation formelle

Pour mieux comprendre les enjeux d'une telle distinction, il est intéressant de lui donner une interprétation formelle à l'aide du modèle Influence/Réaction que nous avons présenté dans le chapitre précédent. Dans le cadre de ce modèle, il existe une différence notable dans la manière dont les influences peuvent être traitées lors de la réaction. Dans la pratique, il est en effet possible de réaliser le calcul d'une réaction de deux manières différentes : en décomposant le calcul global en réactions élémentaires et indépendantes, ou alors, en calculant la réaction de

l'environnement en traitant l'ensemble des influences de manière simultanée, comme un tout. Par abus de langage, nous parlerons respectivement de *réactions linéaires* et de *réactions non linéaires*, en rapport à la manière dont les interactions sont traitées (cf. figure 7.7).

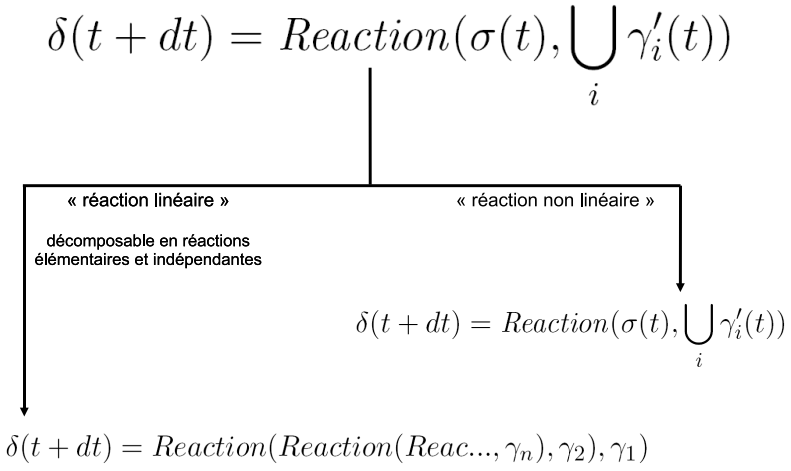

FIG. 7.7 – Réaction linéaire et réaction non linéaire.

Considérons maintenant l'exemple suivant : soient $\delta(t)$ l'état dynamique d'un système lambda à l'instant t et deux influences γ_1 et γ_2 produites à ce même instant. De prime abord, le nouvel état dynamique $\delta(t+dt)$ du système est donné par l'application d'une fonction réaction telle que : $\delta(t+dt) = Reaction(\delta(t), \gamma_1, \gamma_2)$. Essayons maintenant de voir ce qui se passe, pour chaque type d'interaction, lorsque l'on essaie de décomposer la réaction en traitant indépendamment les influences de manière séquentielle.

Plaçons-nous tout d'abord dans l'hypothèse où ces deux influences correspondent à des actions identifiées comme ayant pour conséquence une interaction faible. Comme nous l'avons vu, notre hypothèse de travail est de considérer que, dans ce cas précis, les influences peuvent non seulement être traitées simultanément mais aussi indépendamment et séquentiellement sans pour autant remettre en cause la cohérence paradigmatique et la sémantique des calculs. Plus formellement, cela veut dire qu'il est possible de calculer une réaction de l'environnement à la première influence γ_1, dans le sens où celle-ci peut engendrer à elle seule une modification de l'environnement. Autrement dit, il est possible de calculer un état dynamique intermédiaire δ' en appliquant la fonction de réaction : $\delta' = Reaction(\delta(t), \gamma_1)$. De la même manière, on peut maintenant calculer les conséquences de la deuxième influence sur δ' de telle sorte que $\delta(t+dt) = Reaction(\delta', \gamma_2)$. Bien sûr, il est aussi possible de procéder dans l'ordre inverse et de considérer tout d'abord la deuxième influence. En tout cas, que ce soit dans un cas ou dans l'autre, les deux équations utilisées ont toutes les deux un sens indépendamment l'une de l'autre : elles expriment une réaction, calculable, du système étant donnée une influence (la diminution d'une ressource par consommation d'un agent par exemple). Elles gardent une sémantique. Dans un langage mathématique, il est donc possible, et potentiellement pertinent, de développer l'équation pour appliquer un principe de réaction linéaire.

Comme le souligne la définition 7.4, cela ne veut pas dire pour autant que la modélisation d'une situation d'interaction faible ne nécessite jamais l'application d'une réaction non linéaire. En effet, on peut vouloir modéliser la simultanéité de l'accès à une ressource ou le fait que deux actions peuvent interférer entre elles. Par exemple, il est important de résoudre le conflit associé à l'accès d'une même brindille si l'on considère que les termites agissent simultanément, sans quoi on peut avoir des incohérences de simulation (la duplication d'une brindille par

exemple). Cependant, dans le cas des interactions faibles, l'utilisation d'un principe de réaction non linéaire correspond à un choix de modélisation, et non à une obligation liée au paradigme utilisé. Il s'agit de répondre à la question suivante : le modèle considéré nécessite-t-il de traiter explicitement la simultanéité ? Le modélisateur a donc ici la **possibilité** de faire un choix qui est uniquement lié au grain de modélisation souhaité, celui-ci ne remettant pas en cause la cohérence paradigmatique.

Plaçons-nous maintenant dans l'hypothèse inverse, où les deux influences impliquent une situation d'interaction forte. Cette fois, les différentes équations n'ont aucun sens lorsqu'elles sont prises séparément. Pour s'en rendre compte, il suffit de reprendre les mêmes équations et de voir que, lorsqu'elles sont considérées indépendamment l'une de l'autre et séquentiellement, cela ne permet pas de calculer le véritable nouvel état du système. En effet, lorsqu'une influence est de nature à provoquer une interaction forte, elle n'a absolument aucun effet sur le système lorsqu'elle est considérée indépendamment des autres influences du même type. Autrement dit, comme nous l'avons expliqué, ce type d'actions ne se suffit pas à lui-même. Plus formellement, la réaction de l'environnement à cette influence est invariablement nulle, c'est-à-dire que l'on a $Reaction(\delta(t), \gamma_1) = \delta(t)$. De la même manière, on obtiendra aussi un résultat nul lorsque viendra le moment de calculer la réaction due à la deuxième influence. Dans ce cas précis, il est donc absolument nécessaire de considérer les deux influences simultanément pour que l'équation de réaction garde un véritable sens, et surtout pour qu'elle soit calculable. Autrement dit, la seule équation valable reste associée à un calcul où les deux influences sont considérées simultanément : $\delta(t + dt) = Reaction(\delta(t), \gamma_1, \gamma_2)$. Par ailleurs, il est intéressant de voir que cette équation est encore plus explicite si on applique d'abord une règle de réduction sur les influences de manière à les factoriser. Autrement dit, γ_1 **et** γ_2 peuvent être sommées pour donner une nouvelle influence γ_3 qui exprime la vraie nature de leur composition et qui pourra ainsi être traitée plus explicitement par la fonction de réaction. Par exemple, dans le contexte de la reproduction entre agents, on pourrait avoir $\gamma_1 = reproAttemp_a$, $\gamma_2 = reproAttemp_b$ et $\gamma_3 = (\gamma_1 \oplus \gamma_2) = reproduction_{a+b}$. En d'autres termes, la somme de ces deux influences représente une influence particulière dont le résultat sur l'environnement ne correspond pas à la somme des résultats qui seraient obtenus si ces influences étaient considérées séparément. Nous verrons comment cette idée peut être concrètement mise en œuvre dans le cadre du modèle formel que nous utiliserons dans le chapitre suivant. Ainsi, à l'inverse d'une situation d'interaction faible, l'utilisation d'une réaction non linéaire ne correspond pas à un choix de modélisation mais à une obligation directement liée à la nature des interactions fortes : le choix est cette fois **impossible** car une interaction forte nécessite la composition explicite des comportements. La figure 7.8 illustre ces différentes idées.

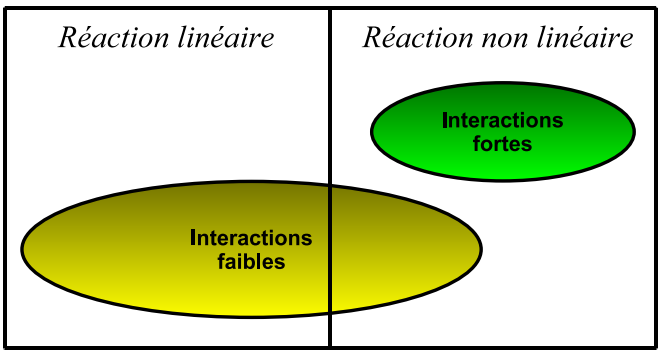

FIG. 7.8 – Une interaction forte est nécessairement liée à une réaction non linéaire et à la composition explicite des comportements.

Au-delà de ces différences fondamentales, nous avons aussi une intuition à propos des résultats issus des différentes manières de calculer les conséquences d'une interaction faible. D'un point de vue global, nous pensons que les différentes manières de calculer le nouvel état du monde sont équivalentes. En fait, bien qu'elles puissent ne pas donner les mêmes valeurs pour les variables d'état du système dans son ensemble, nous pensons que les comportements globaux obtenus peuvent être considérés comme équivalents dans la mesure où chacune des trois solutions ne remet pas en cause la cohérence du calcul. Ce qui n'est pas vrai dans le cas des interactions fortes où le biais de modélisation peut remettre en cause la cohérence et le sens premier du modèle. Autrement dit, dans le cas où deux influences impliquent une interaction faible, nous faisons l'hypothèse que :

$$Reaction(\delta(t), \gamma_1, \gamma_2) \approx Reaction(Reaction(\delta(t), \gamma_1), \gamma_2) \approx Reaction(Reaction(\delta(t), \gamma_2), \gamma_1) \tag{7.1}$$

Bien qu' il ne s'agisse que d'une intuition, il y a là un point d'importance qui constitue à n'en pas douter une perspective de travail intéressante.

7.6.7 Du modèle conceptuel à l'implémentation

La distinction que nous venons de proposer a essentiellement pour but de montrer la nécessité de mieux spécifier nos modèles théoriques en fonction des problèmes soulevés par l'implémentation. La distinction interaction faible/interaction forte est un pas dans cette direction. Elle constitue un critère qui permet de guider la modélisation et l'implémentation des interactions. Ainsi, à partir des définitions que nous avons données de ces deux notions, l'analyse des interactions contenues dans un modèle peut être effectuée suivant le principe illustré par la figure 7.9.

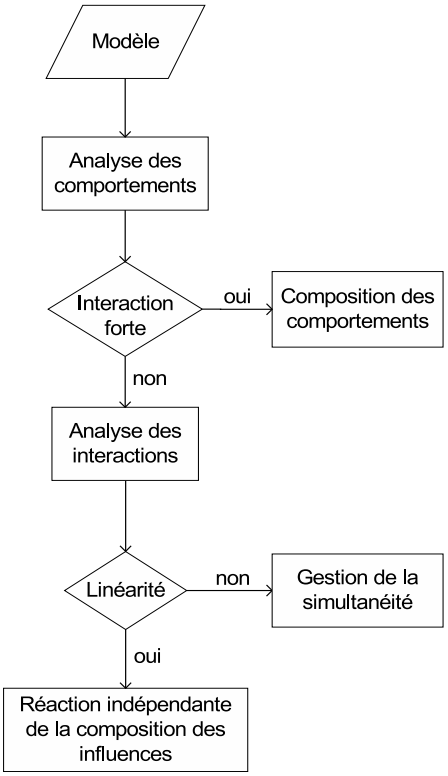

FIG. 7.9 – Principe d'analyse pour la modélisation et l'implémentation des interactions.

Tout d'abord, une analyse des comportements permet de déterminer la nature des interactions qu'ils engendrent : des interactions fortes dans le cas où les comportements doivent être nécessairement composés pour donner un résultat et des interactions faibles dans le cas contraire. Ensuite, une analyse des interactions faibles est effectuée afin de déterminer s'il convient d'appliquer une réaction non linéaire ou pas étant donné le grain de modélisation souhaité. L'implémentation des interactions peut alors être considérée en fonction de cette analyse. Nous avons vu que les situations d'interaction forte nécessitent une attention toute particulière car la gestion des interactions peut être en contradiction avec le principe d'autonomie décisionnelle des agents. Au contraire, l'implémentation des interactions faibles est beaucoup plus souple dans le sens où une implémentation cohérente de ces interactions ne repose pas nécessairement sur des synchronisations de comportements. Par ailleurs, cela montre que ces deux catégories d'interaction ne sont pas incompatibles au sein d'une même simulation. De manière classique, les simulations reposent souvent sur un seul principe de gestion des interactions qui est appliqué à l'ensemble du système. Ce qui est sans aucun doute l'une des raisons pour lesquelles il est assez difficile d'intégrer des interactions fortement hétérogènes de façon satisfaisante dans un même modèle.

7.7 Discussion

La classification que nous venons de proposer a donc pour but de fournir des éléments de réflexion qui permettent de faire un lien direct entre le modèle conceptuel et la manière dont il doit être implémenté. Dans un modèle contenant des interactions fortes, la gestion de ces interactions a non seulement de lourdes conséquences sur les résultats mais elle peut aussi remettre en cause la cohérence paradigmatique de la modélisation. C'est pourquoi il sera incontournable de programmer une réaction de l'environnement qui prenne en compte l'ensemble des comportements individuels. Au contraire, la gestion des interactions faibles, bien qu'elle puisse sans aucun doute avoir une influence sur les résultats, n'oblige pas à considérer explicitement la simultanéité pour que la cohérence paradigmatique soit respectée.

Cela explique ici la difficulté et la frustration que nous avions eues en programmant la simultanéité des événements dans l'application Warbot (cf. section 6.5.3 page 134). En effet, bien que nous disposâmes d'un bon modèle de la simultanéité incarné par l'application du principe Influence/Réaction, nous nous sommes retrouvé à gérer certaines influences de façon séquentielle. Nous pouvons maintenant l'expliquer par le fait que certaines des influences présentes dans ce modèle n'offrent tout simplement pas une combinaison intuitive. Finalement, le fait que les agents bougent de manière séquentielle et non simultanée est-il réellement un problème dans ce modèle ? Cela ne heurte pas spécialement notre sens commun. Dans le monde réel, cet ensemble de robots ne serait a priori pas commandé par une horloge temporelle globale.

Au contraire, l'interaction de reproduction n'a de sens que dans la simultanéité. Mais, en fait de simultanéité, il serait peut-être plus approprié de parler de conjonction des comportements. En effet, si nous concevons très bien que deux événements peuvent être simultanés, il est beaucoup plus difficile d'utiliser cette notion de simultanéité en ce qui concerne des comportements. Peut-on vraiment rattacher le désir de se reproduire à un instant t de la simulation ? Nous voici à un point crucial de notre analyse qui nous ramène aux conclusions que nous avions formulées à la fin du chapitre précédent à propos du principe Influence/Réaction. Nous pensons que son intérêt dépasse largement son aptitude à modéliser la simultanéité des comportements.

Tout d'abord, en supprimant la représentation des actions par modification d'un état global grâce à la notion d'influence, il permet de donner une véritable sémantique à la gestion des interactions lors de la phase de réaction. Comme nous l'avons dit, de manière classique la modélisation/programmation de l'interaction est difficilement identifiable car elle est le plus souvent diluée dans les autres parties du modèle[4]. Ce qui prête à confusion dans la modélisation du système et facilite les phénomènes de divergence implémententatoire. L'application du principe Influence/Réaction favorise au contraire l'isolation d'un module des interactions, à la fois dans le modèle conceptuel et dans le programme du simulateur.

Ensuite, l'application du principe Influence/Réaction implique de clairement distinguer les variables d'état qui appartiennent aux systèmes décisionnels de l'agent de celles qui concernent son aspect physique et qui font partie intégrante de l'environnement. Et nous avons vu qu'il s'agit là d'un point essentiel dans la mise en œuvre d'agents véritablement autonomes. Sans cela nous n'aurions pas pu définir l'autonomie ainsi que nous l'avons fait, c'est-à-dire en utilisant la notion d'intégrité interne d'un agent qui stipule que son architecture interne ne doit être accédée ni en lecture ni en écriture. Couplée à l'utilisation du principe Influence/Réaction, la distinction esprit/corps est en effet le seul moyen de faire interagir les agents sans violer leur intégrité interne.

A l'origine, le modèle Influence/Réaction a été élaboré pour répondre aux problèmes de la modélisation de la simultanéité dans les systèmes multi-agents. C'est d'ailleurs historiquement pour cette raison que nous l'avons personnellement étudié pour la première fois. Aujourd'hui, les conséquences de son utilisation en tant que principe de modélisation nous apparaissent encore bien plus fondamentales. Parmi ces conséquences, l'abandon de la représentation de l'action comme une modification de l'état global est pour nous la plus révolutionnaire et la plus prometteuse. En effet, il s'agit là d'abandonner un modèle de l'action qui est aujourd'hui très fortement ancré dans nos pratiques. Il est non seulement un des héritages des systèmes "mono agent" issus de l'intelligence artificielle classique, mais il découle aussi de l'utilisation des langages orientés objet pour la programmation des systèmes multi-agents. D'ailleurs, s'il paraît évident qu'un agent est une entité qui ne devrait pas être en mesure de modifier son environnement à l'aide d'un simple appel de méthode, dans la pratique c'est très souvent le cas pour des raisons que l'on pourrait presque qualifier de culturelles. Autrement dit, notre sentiment profond est qu'il est aujourd'hui nécessaire de changer la manière dont nous modélisons et programmons les systèmes multi-agents. La notion d'influence joue en cela un rôle fondamental. Nous pensons que c'est de cette notion que découlent les autres conséquences intéressantes issues de l'application du principe Influence/Réaction : la distinction esprit/corps, le respect de l'autonomie des agents et l'identification du module des interactions. Finalement, tout au long de ce manuscrit, nous avons essayé de dégager, pour le paradigme agent, ce que nous pourrions appeler des *principes de bonne modélisation/programmation* en clin d'œil à l'ingénierie logicielle classique. L'application du principe Influence/Réaction permet d'en respecter l'intégralité. C'est pourquoi nous soutenons ici que l'idée proposée par le principe Influence/Réaction garde toute sa raison d'être, **même lorsqu'il n'est pas question de simultanéité** dans le modèle. Dans ce chapitre nous avons argumenté ces idées d'une manière quelque peu informelle. Dans le chapitre suivant, nous illustrerons les différents aspects de notre analyse à travers l'utilisation d'un modèle formel qui nous permettra une approche plus concrète de ces différents points.

[4]Généralement, la majeure partie de l'interaction se trouve dans le code représentant le comportement des agents. En effet, dans le cas d'une représentation de l'action par modification d'un état global, c'est finalement le plus souvent à eux que revient la tâche de calculer ce qu'il advient de leurs actions.

7.8 Résumé du chapitre

Dans ce chapitre, nous avons abordé le problème de la modélisation des interactions entre agents. Nous avons vu qu'il est aujourd'hui important de donner une dimension purement informatique aux questions soulevées par cette problématique. C'est pourquoi nous avons proposé une définition technique de l'interaction qui nous a permis d'isoler le traitement informatique qui lui correspond. Ce qui nous a permis d'argumenter sur l'intérêt de considérer ce traitement comme un module distingué du modèle et de son implémentation. Ce traitement est en effet trop souvent dilué dans différentes parties de la modélisation et nous avons vu que cela contribuer aux phénomènes de divergence implémentatoire.

Ensuite, nous avons abordé la question de la relation de modélisation dans les systèmes multi-agents. Nous avons alors essayé de démontrer l'intérêt de considérer un nouvel aspect dans la validation d'un modèle multi-agents : la cohérence paradigmatique. Ce qui nous amené à rediscuter la notion d'autonomie et à en proposer une définition une nouvelle fois fondée sur des considérations techniques. A partir de quoi nous avons constaté la non-conformité de certains modèles du fait d'une violation manifeste de l'autonomie des agents. Cela nous a permis de montrer que le principe Influence/Réaction est un moyen de respecter l'intégrité interne des agents, et donc de rendre leur autonomie à ces entités grâce à une prise en compte de la simultanéité des comportements.

Cependant, nous avons vu que la gestion systématique de la simultanéité des influences produites par les agents n'était pas toujours une condition nécessaire au respect de l'autonomie des agents suivant la nature des interactions impliquées. Nous avons ainsi proposé de distinguer deux grandes classes de situations interactionnelles en fonction des besoins de modélisation requis par le respect de la cohérence paradigmatique. Les situations d'interactions fortes qui nécessitent de considérer les comportements simultanément dans la réaction, et les situations d'interactions faibles qui peuvent être traitées sans que la gestion de la simultanéité soit une obligation, toujours relativement à la cohérence paradigmatique, la qualité/validité conceptuelle du modèle étant ici une question qui implique d'autres aspects de la cohérence du modèle.

Finalement, nous en avons conclu que le principal apport du principe Influence/Réaction n'était pas dans la gestion de la simultanéité qu'il permet par ailleurs. L'ensemble de ce chapitre constitue en fait un nouvel argumentaire en faveur de l'abandon de la modélisation de l'action par transformation d'un état global, au profit de la notion d'influence. Dans notre idée, les différents principes de "bonne" modélisation que nous avons énoncés jusqu'à maintenant (distinction esprit/corps, respect de l'autonomie, contraintes d'intégrités, etc.) impliquent de considérer la délibération d'un agent comme une influence et vice versa.

Dans le chapitre suivant, nous montrerons comment ces différents aspects peuvent être concrètement mis en œuvre dans le cadre d'un modèle algébrique formel de systèmes multi-agents appelé MIC* (Mouvement Interaction Calcul). Nous montrerons en quoi ce modèle constitue une réponse à de nombreux problèmes que nous avons soulevés grâce au rôle central qu'il donne à la notion d'environnement. Nous illustrerons ensuite son utilisation pour la simulation à travers un exemple complet. Ce qui nous permettra de mettre en pratique les principes de bonne modélisation/programmation que nous allons maintenant récapituler.

Chapitre 8

Modélisation et simulation avec le modèle MIC*

TOUT au long de ce document, nous avons essayé d'identifier certaines particularités du paradigme agent afin de mieux cerner la manière dont les systèmes complexes qui se fondent sur cette approche doivent être modélisés. Dans ce chapitre, nous allons tout d'abord récapituler les différents points de cette analyse. Ensuite nous présenterons le modèle MIC*, {*Mouvement Interaction Calcul*}*, un modèle formel d'environnement d'exécution pour système multi-agents. Nous montrerons alors en quoi celui-ci constitue un moyen satisfaisant d'appliquer l'ensemble des principes de modélisation que nous avons identifiés de manière formelle.

8.1 Récapitulatif des contraintes conceptuelles et pratiques identifiées

Dans cette section nous allons résumer les différentes contraintes que nous avons précédemment identifiées quant à la modélisation et à la programmation d'un système multi-agents.

8.1.1 Contrainte de localité pour la perception et l'action

Dans le chapitre 3, nous avons vu que la notion d'agent est inséparable de celle d'environnement (cf. section 3.1.3 page 47). En fait, la majorité des contraintes qui pèsent sur la modélisation d'un système multi-agents concernent les relations qui existent entre un agent et son environnement. La première d'entre elles découle de la définition même d'un agent : c'est une entité qui se situe et évolue dans un environnement qui ne lui est pas accessible en totalité. Celui-ci ne peut donc accéder que **partiellement et localement** aux informations qui décrivent l'environnement. Symétriquement, les actions perpétrées par un agent n'ont qu'une **portée locale** (cf. section 3.4.2 page 60). Dans le contexte de ce document, c'est ce que nous avons désigné par la *contrainte de localité*.

8.1.2 Contrainte d'intégrité environnementale

Toujours concernant le couple agent/environnement, nous avons évoqué à de nombreuses reprises cette contrainte qui repose sur le fait qu'un agent n'est pas une entité en mesure

de calculer les conséquences de ses actes (cf. sections 3.4.2 page 62 et 6.2.1 page 118). A ce titre, il est important qu'un agent ne soit pas en mesure de modifier directement les variables d'état de l'environnement. Autrement dit, **les variables environnementales ne doivent pas être modifiées par les agents**. C'est ce que nous avons appelé la *contrainte d'intégrité environnementale*.

8.1.3 Autonomie et intégrité interne d'un agent

L'autonomie est une caractéristique fondamentale des agents. Cependant, nous avons vu que cette notion est rarement concrètement traduite en termes purement informatiques (cf. section 7.4 page 148). Par ailleurs, nous avons vu à plusieurs reprises que l'un des grands problèmes liés à la modélisation multi-agents tient au manque de correspondance qui existe entre les fondements conceptuels du paradigme et des structures informatiques concrètes (cf. section 4.3.2 page 85 par exemple). Ainsi, sans correspondance claire avec un principe de modélisation, il est alors impossible de dire que les agents que nous modélisons sont effectivement des entités autonomes dans la mesure où cette notion reste uniquement théorique.

Pour remédier à cela, nous avons utilisé une définition de l'autonomie basée sur des critères purement implémentatoires en utilisant la notion d'**intégrité interne d'un agent** (cf. section 7.4.2 page 149). Ce qui nous a permis d'argumenter sur la nécessité de valider la modélisation d'un système multi-agents suivant le *principe de cohérence paradigmatique* (cf. section 7.5 page 151). Pour résumer, **si l'intégrité interne d'un agent n'est pas respectée alors son autonomie est violée** et la relation de modélisation n'est pas vérifiée.

8.1.4 Diminuer la possibilité de divergence implémentatoire

Dans le chapitre 4, nous avons présenté le problème de la divergence implémentatoire (cf. section 4.2 page 80) et nous avons insisté sur le fait que l'origine de ce phénomène est en grande partie liée à un manque de spécification des modèles qui sont proposés dans la littérature (cf. section 4.2.2 page 81). Dans le même temps, nous avons vu que la complexité des modèles considérés ne facilite pas leur formalisation et que de nombreux autres facteurs sont à l'origine de ce problème (cf. section 4.3 page 84), notamment la complexité du logiciel obtenu pour simuler le modèle. De cela, nous avons tiré deux contraintes complémentaires :

- **rendre le plus explicite possible le modèle et son simulateur**.
- **élaborer des formalismes plus adéquats pour la modélisation de systèmes multi-agents**.

8.2 Solutions méthodologiques et pratiques proposées

8.2.1 Distinction esprit/corps

A plusieurs reprises, nous avons vu qu'il est important de faire une différence claire entre le système conatif d'un agent et la partie qui modélise son physique (cf. sections 3.4.2 page 62 et 6.4.5 page 125 par exemple) . Il y a là un intérêt à la fois pratique et conceptuel. Cela permet tout d'abord de respecter la contrainte d'intégrité environnementale en faisant une distinction explicite entre les variables sur lesquelles l'agent a un contrôle total (système conatif) et celles sur lesquelles il n'a aucun pouvoir (partie physique). Par ailleurs, nous avons vu que cette distinction est tout à fait fondamentale étant donnée l'interprétation de la propriété

d'autonomie que nous avons utilisée : le respect effectif de l'intégrité interne d'un agent dans la modélisation impose de faire cette distinction (cf. section 7.4.2 page 149). De plus, nous avons aussi souligné l'intérêt de cette séparation dans le cadre de l'application du principe Influence/Réaction (cf. section 6.4.5 page 125).

8.2.2 Réification des notions de perceptions/actions via capteurs/effecteurs

Pour prendre en compte la contrainte de localité, nous avons vu qu'il est très intéressant de considérer les moyens d'action et de perception d'un agent comme des fonctions distinguées de son système conatif (cf. sections 3.4.2 page 59 et 6.4.2 page 122).

En ce qui concerne la perception, cela permet de gérer le fait qu'un agent n'a qu'une vue subjective et partielle de son environnement. Un capteur permet en effet d'implémenter une fonction de filtrage des variables d'état environnementales afin de fournir aux agents des structures de données qui représentent des perceptions subjectives. De la même façon, la réification des effecteurs d'un agent permet d'effectuer le même type de traitement aux actions issues de la délibération d'un agent. D'une manière générale, il s'agit de faire le pont entre ce qu'un agent a à l'esprit et la réalité de l'environnement.

8.2.3 Application du principe Influence/Réaction

Dans le chapitre 6, nous avons présenté le principe Influence/Réaction comme une solution pertinente au problème de la modélisation de la simultanéité dans le système multi-agents. Dans le même temps, nous avons montré que l'idée-force de ce principe repose sur un abandon de l'action comme une transformation d'un état global (section 6.2.1 page 118).

Premièrement, cela permet de concrétiser la contrainte d'intégrité environnementale grâce à la notion d'influence : les agents ne modifient pas directement les variables d'état de l'environnement. Deuxièmement, nous avons vu que son application implique de faire la distinction entre les variables du système décisionnel et les variables qui représentent le physique d'un agent. Ce qui participe à une modélisation claire de la distinction esprit/corps. Dans une vision plus globale, nous avons longuement argumenté sur l'intérêt et l'importance d'utiliser le principe Influence/Réaction dans le contexte des systèmes multi-agents. Celui-ci constitue sans aucun doute la pierre angulaire de notre thèse.

8.2.4 Identification d'un module *interaction* dans la modélisation

Pour rendre explicite la modélisation d'un système multi-agents et les spécifications qui en découlent, nous avons soutenu qu'il était important de distinguer quatre modules différents : comportements, environnement, ordonnancement et interaction (cf. section 3.7 page 74). Si les trois premiers de ces modules sont généralement étudiés en tant que tels, dans le chapitre 7 nous avons insisté sur le fait que l'interaction n'a pas la place qu'elle mérite dans le processus de modélisation. Et nous avons motivé cette analyse en démontrant le rôle important que la modélisation de l'interaction joue dans les phénomènes de divergence implémentatoire. L'identification explicite de ce module constitue donc un point fondamental de notre approche.

Par ailleurs, dans la section 7.6.3 (page 157) nous avons vu que le respect de la cohérence paradigmatique nécessite d'apporter une attention toute particulière à la modélisation de certaines situations interactionnelles (*interactions fortes*).

8.2.5 Récapitulatif

Récapitulons ici les différents aspects que nous considérons importants et que nous mettrons en œuvre :

- modélisation et implémentation suivant quatre modules distincts : comportements, environnement, ordonnancement et interaction.
- réification des moyens de perception et d'action d'un agent.
- application de la distinction esprit/corps.
- utilisation du principe Influence/Réaction pour modéliser la dynamique du système.
- respect de la contrainte d'intégrité interne d'un agent : garantir la propriété d'autonomie.
- respect de la contrainte d'intégrité environnementale.
- respect du principe de cohérence paradigmatique dans la modélisation.

8.3 Le modèle {*Mouvement Interaction Calcul*}* : MIC^*

8.3.1 Motivations

Vers des logiciels *Anytime Anywhere*

En dehors du contexte de la simulation de systèmes complexes, le paradigme agent constitue aussi une alternative très séduisante pour l'élaboration de logiciels complexes et distribués. Un pan entier de la recherche utilise aujourd'hui les notions de proactivité et d'autonomie pour concevoir des logiciels voués à évoluer dans des univers dynamiques, distribués et ouverts dont Internet est le meilleur exemple.

Dans ce contexte, les défis technologiques qui doivent être relevés par les systèmes multi-agents sont nombreux. En premier lieu, il semble qu'il existe une contradiction entre les buts poursuivis et les moyens d'y parvenir. En effet, dans un premier temps il s'agit bien sûr d'élaborer des systèmes informatiques qui possèdent des propriétés classiques comme la stabilité, la fiabilité ou encore la sécurité. Dans le même temps, on souhaite aussi que ces systèmes soient adaptatifs, ouverts[1] et peuplés par des entités logicielles mobiles dont les comportements sont imprédictibles, car autonomes. En d'autres termes, la difficulté est de réaliser des systèmes dont l'évolution reste maîtrisée bien que produite par des entités incontrôlables, les agents.

Pour atteindre cet objectif, la métaphore sociale est aujourd'hui la plus en vogue. Il semble en effet assez logique d'essayer de transposer aux systèmes multi-agents les mécanismes qui font fonctionner les sociétés humaines ou animales : bien que chaque individu soit autonome du point de vue de son processus décisionnel, ces sociétés remplissent des fonctions globales qui sont le fruit d'un subtil mélange entre des buts individuels et des aspirations communes. Concevoir un logiciel comme une *société d'agents* est somme toute une conséquence naturelle de notre vision anthropomorphique des agents. Dans ce contexte, il existe deux approches complémentaires : les travaux basés sur des concepts organisationnels et les travaux fondés sur la définition de normes. Dans les premiers, la problématique repose sur l'élaboration de moyens permettant de structurer l'activité des entités [Ferber *et al.* , 2003]. Dans les deuxièmes, le propos est de définir un cadre qui permette de diriger et/ou de contrôler l'activité des agents grâce à un ensemble de règles [Stratulat, 2002].

[1] Accessibles à des agents ayant une architecture inconnue.

La notion d'*environnement de déploiement*

Que ce soit dans l'une ou l'autre des deux précédentes approches, l'idée fondamentale est de structurer l'**environnement** dans lequel évoluent les entités qui composent le système informatique. Dans le troisième chapitre (cf. section 3.1.3 page 47), nous avions déjà souligné l'importance de l'environnement dans la conception d'un système multi-agents : c'est lui qui définit les conditions d'existence et d'évolution des agents. En fait, bien plus que les agents eux-mêmes, l'environnement constitue le cœur de l'ingénierie des systèmes **ouverts**. En effet, dans ces systèmes, il est par définition impossible de faire des suppositions sur la nature de l'architecture interne des agents.

Par ailleurs, il est nécessaire de considérer que ces mêmes agents ne sont pas forcément bienveillants et qu'ils sont incontrôlables dans les décisions qu'ils prennent. Finalement, la seule entité logicielle sur laquelle on puisse garder une maîtrise absolue, tant au niveau de son design que de sa dynamique, c'est l'environnement. Il est donc nécessaire d'en identifier les caractéristiques fondamentales [Weyns *et al.* , 2005b].

Au plus haut niveau d'abstraction, c'est bien la structure logicielle sur laquelle les agents sont exécutés qui constitue l'environnement primordial. Pour bien comprendre ce point de vue, il est ici nécessaire de faire une distinction claire entre l'environnement tel qu'il est généralement compris et la vision qu'il faut ici en avoir. Dans un système multi-agents, la notion d'environnement désigne généralement le modèle dans lequel les agents se situent, on parle d'*agents situés* dans un environnement : une organisation, une société, une grille de cellules, etc. C'est dans cette acception que nous avons le plus souvent considéré l'environnement jusqu'à présent. Ici, nous sommes à un niveau d'abstraction supérieur : nous faisons exclusivement référence à la structure logicielle utilisée pour déployer concrètement le système multi-agents dans son ensemble : l'environnement situé et les agents. Cette entité logicielle constitue une problématique à part et nous utiliserons les termes *environnement de déploiement* pour désigner cet aspect d'un système [Gouaïch *et al.* , 2005].

Rarement étudié en tant que tel, l'environnement de déploiement est aujourd'hui cependant de plus en plus identifié comme une composante fondamentale de l'ingénierie logicielle des systèmes complexes : cela semble aujourd'hui le meilleur moyen de relever le défi des systèmes informatiques de demain qui seront **ouverts**, composés d'environnements **situés** (par lequel des entités autonomes pourront agir et être influencées), où les connaissances et le contrôle seront **distribués** et les **interactions locales** [Zambonelli & Parunak, 2002].

Ainsi, c'est dans le contexte de l'ingénierie des systèmes informatiques distribués que Gouaïch et Guiraud ont tout d'abord proposé le modèle MIC* [Gouaïch & Guiraud, 2002]. Le projet associé[2], dans lequel nous sommes directement impliqué, a pour objectif de proposer un modèle d'environnement de déploiement pour système multi-agents basé sur un modèle algébrique formel [Gouaïch *et al.* , 2003, Gouaïch, 2005]. Inspirée par des travaux sur les systèmes multi-agents, le code mobile et les modèles de calcul formel, l'idée fondamentale de cette approche est de faire une distinction explicite entre l'environnement de déploiement, sa dynamique (les lois qui régissent l'environnement situé qu'il définit) et les entités qui le peuplent. La figure 8.1 illustre cette approche. A cela s'ajoute l'idée que l'environnement de déploiement doit jouer un rôle actif et prépondérant dans la gestion des processus d'interaction qui existent entre les différentes entités du système.

[2] sourceforge.net/projects/mic.

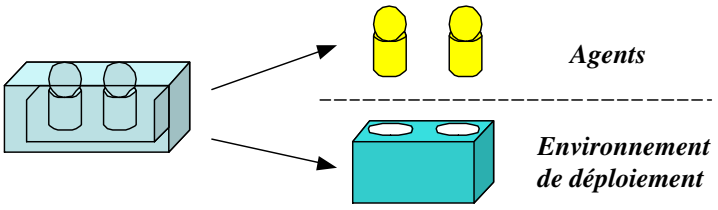

FIG. 8.1 – Séparation explicite entre l'environnement de déploiement et les agents.

Objectifs et principes de conception du modèle MIC*

A la différence des modèles formels de calcul mobile comme *Ambient* [Cardelli, 1999] ou le π-calcul [Milner, 1999] qui se focalisent sur les propriétés internes des processus de calcul, l'objectif du projet associé à l'élaboration du modèle MIC* est d'étudier les propriétés de l'environnement qui entoure les processus de calcul, c'est-à-dire les propriétés de l'environnement de déploiement. Dans cette optique, l'élaboration du modèle MIC* a été réalisée en gardant à l'esprit deux objectifs complémentaires : décrire de manière rigoureuse les concepts manipulés et proposer un modèle qui puisse être directement implémenté.

- **description rigoureuse**. Comme nous l'avons vu plusieurs fois, des concepts génériques comme l'interaction, la mobilité et les agents sont souvent utilisés avec ambiguïté alors qu'il est important de les faire correspondre avec des structures computationnelles concrètes. C'est pourquoi, sans pour autant imposer des définitions consensuelles, MIC* caractérise rigoureusement les concepts utilisés dans le cadre de son étude.
- **modèle implémentable**. En adoptant une démarche purement algébrique, la structure du modèle obtenue peut être implémentée par des structures informatiques qui correspondent directement aux objets algébriques manipulés. Par conséquent, il est possible de faire un lien direct entre les structures informatiques du système et son modèle algébrique. Ceci contraste avec les démarches formelles classiques qui définissent une syntaxe puis des fonctions d'interprétation de cette syntaxe. Le lien entre l'implémentation du modèle et le modèle est alors plus difficile à établir.

8.3.2 Description des concepts fondamentaux de MIC*

Description statique

L'ensemble d'une structure définie à l'aide du modèle MIC* se compose de trois concepts fondamentaux :

- les **processus de calcul**.
- les **objets d'interaction**.
- les **espaces d'interaction**.

Processus de calcul (**Agents**). Dans MIC*, un agent est une entité composée de trois briques fondamentales : sa structure interne, sa boîte de réception appelée *inbox* et sa boîte d'émission appelée *outbox*. Ces deux derniers concepts, généralisation des notions d'effecteurs et de senseurs, définissent les moyens par lesquels un agent existe, perçoit et agit dans l'environnement de déploiement. Symétriquement, la inbox et la outbox constituent pour l'environnement de déploiement la seule partie observable d'un agent. La structure interne d'un agent est quant à elle considérée comme un système dynamique classique, c'est-à-dire définie par un ensemble de variables d'état, nous parlerons de la *mémoire* de l'agent, et une dynamique in-

terne propre. L'environnement de déploiement n'a pas accès à cette structure. A tout instant, l'état global d'un agent est formellement défini par l'état de sa mémoire, de sa inbox et de sa outbox.

Objet d'interaction. Les objets d'interaction constituent dans MIC* le moyen concret par lequel les agents existent et interagissent dans un environnement de déploiement. En fait, la inbox et la outbox de chaque agent sont uniquement constituées d'objets d'interaction. Un objet d'interaction doit être vu comme une information typée qui décrit un ensemble de couples attributs/valeurs. Un agent est donc une entité qui perçoit des objets d'interaction depuis sa inbox et effectue un calcul local pour produire de nouveaux objets d'interaction dans sa outbox. Par ailleurs, les objets d'interaction sont formellement définis de telle manière qu'ils définissent une structure de monoïde. Autrement dit, il est possible de définir une loi de composition $+$, commutative, telle que, soient les objets d'interaction a et b, $a + b$ définit un *objet d'interaction somme* pouvant posséder une sémantique particulière. Nous verrons l'intérêt que cela peut avoir pour la formalisation des interactions fortes. De plus, un objet d'interaction vide I_0 est formellement défini pour exprimer l'objet d'interaction nul ou *objet d'interaction zéro*. Celui-ci sera par exemple utilisé pour exprimer de manière abstraite le fait que l'interaction entre deux entités est inexistante. Par ailleurs, chaque objet d'interaction possède, d'un point de vue formel, un opposé tel que $a + (-a) = I_0$.

Espace d'interaction. Dans MIC*, un espace d'interaction est une abstraction qui représente une localisation où différentes entités interagissent. Il est le lieu de l'interaction : c'est dans cette espace logique que la inbox et la outbox d'un agent sont définies. La présence d'un agent dans un espace d'interaction correspond simplement au fait que sa outbox n'est pas vide dans cet espace. Il est ici important de comprendre que le contenu de la outbox d'un agent dans un espace d'interaction, des objets d'interaction, constitue la représentation de l'agent dans ce même espace : il signifie l'existence de cet agent dans cette localisation logique du système. Lorsqu'un agent n'est pas présent dans un espace d'interaction, sa outbox est vide. Un espace d'interaction est aussi une entité active. Premièrement car il a la capacité d'altérer les objets d'interaction qui sont émis par les agents. Deuxièmement car il définit aussi localement les conditions dans lesquelles l'interaction se déroule : c'est suivant des règles locales à un espace d'interaction que la dynamique calculée à partir des objets d'interaction est définie. Par ailleurs, un agent peut appartenir à plusieurs espaces d'interaction, ce qui définit son ubiquité logique. La inbox et la outbox d'un agent sont alors définies sur chacun des espaces d'interaction dans lesquels l'agent se trouve. En conséquence, un agent ne pourra interagir qu'avec les entités présentes dans les mêmes espaces d'interaction[3].

La structure statique d'un modèle de MIC* est définie comme un composé de ces trois éléments de base. Plus formellement, soient les définitions suivantes :

- $(\mathcal{O}, +)$ le groupe abélien qui représente l'ensemble des objets d'interaction.
- $\mathcal{I}$ et $\mathcal{J}$, deux ensembles représentant respectivement les processus de calcul (les agents) et les espaces d'interaction.
- $(\mathcal{L}_i)$ avec $i \in \mathcal{I}$, une famille d'ensemble pointés où $\mathcal{L}_i$ représentent les termes du langage interne du processus i.
- $\mathcal{O}^{\mathcal{I}\times\mathcal{J}}$, l'ensemble des familles $(a_{i,j})_{(i,j)\in(\mathcal{I}\times\mathcal{J})}$ éléments de $\mathcal{O}$.
- l, l'ensemble des familles $(m_i)_{i\in\mathcal{I}}$ avec m_i élément de $\mathcal{L}_i$.

[3]Bien que le niveau d'abstraction ne soit pas le même, la notion d'espace d'interaction peut être rapprochée de celle de groupe dans le modèle AGR (cf. section 5.2.1 page 96). Rappelons que dans ce modèle un agent peut appartenir à plusieurs groupes et qu'il ne peut interagir qu'avec les membres des groupes dans lesquels il se trouve.

La structure statique d'un environnement MIC* est totalement définie par le triplet de matrices illustré sur la figure 8.2. L'ensemble de ces trois matrices constitue ce qui est appelé un *terme*

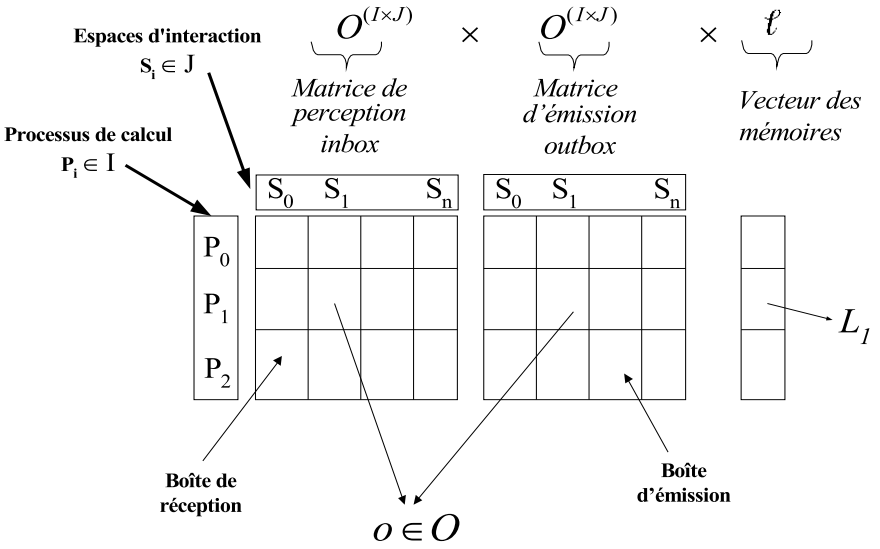

FIG. 8.2 – La structure statique d'un environnement MIC* : un triplet de matrices.

*MIC** . Dans le cas d'un système ouvert, la valeur du vecteur l, c'est-à-dire l'état interne des agents, est inconnue. Seules les matrices de perception et d'émission constituent la partie observable de l'état du système. Cette partie contient uniquement des objets d'interaction. Dans la suite nous noterons respectivement $out[i,j]$ et $in[i,j]$ les valeurs de la outbox et de la inbox du processus i dans l'espace d'interaction j et m_i la mémoire de i.

Une des propriétés intéressantes des termes MIC* repose sur le fait qu'ils sont naturellement composables par une simple opération d'addition matricielle. Ainsi, dans un contexte distribué, il est possible de connecter deux environnements distincts pour obtenir un seul environnement distribué. De façon duale, la déconnexion entre deux environnements consiste à réaliser une opération de soustraction entre les matrices. La figure 8.3 illustre cet aspect du modèle. D'une manière plus générale, la structure matricielle permet aussi de distribuer très facilement un même terme MIC* en répliquant les espaces d'interaction sur différents sites. Il suffira alors de transmettre les objets d'interaction créés par les agents pour maintenir la cohérence du système sur les différents sites.

Dynamiques d'un terme MIC*

La dynamique d'un terme MIC* s'effectue selon trois transformations fondamentales qui s'appliquent de manière séquentielle. On parle des trois lois d'évolution de MIC* :

- le **mouvement**.
- l'**interaction**.
- le **calcul** ou computation.

Mouvement. Le mouvement est une transformation d'un terme où la matrice des perceptions et le vecteur des mémoires restent inchangés. En fait, le mouvement peut être vu comme le déplacement des représentations d'un agent dans différentes localisations du système. Ce qui correspond au déplacement d'un ou de plusieurs objets d'interaction entre différents espaces d'interaction. Plus formellement, le mouvement est une application μ fonction de la valeur

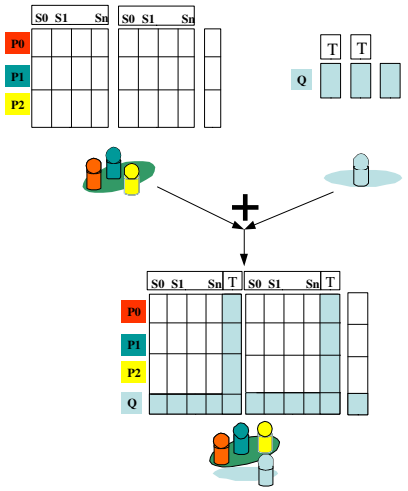

FIG. 8.3 – Composition de deux termes MIC* par addition matricielle.

des boîtes d'émission et qui ne modifie pas les boîtes de réception et la mémoire des agents. Un mouvement est une opération qui est la conséquence d'une loi localement définie dans un espace d'interaction. Autrement dit, elle est entièrement définie dans l'environnement et un agent ne maîtrise pas ses mouvements. En fait, un mouvement est déclenché lorsque la représentation d'un agent entraîne simultanément son expulsion d'un espace d'interaction initial et son aspiration dans un nouvel espace d'interaction. Soient j et h deux espaces d'interaction et a un agent, une application de mouvement de l'espace d'interaction j vers l'espace d'interaction h correspond à une fonction $mouvement_{j\to h} \in \mu$ telle que :

$$mouvement_{j\to h}(out[a,j]) = aspiration_h(expulsion_j(out[a,j])) = out[a,h] \times out'[a,j] \quad (8.1)$$

La figure 8.4 illustre l'application d'une loi de mouvement entre deux espaces X et Y. Dans ce schéma, l'objet d'interaction e peut par exemple représenter une requête à laquelle les espaces d'interaction X et Y sont sensibles : X expulse l'objet d'interaction e qui est ensuite aspiré par Y.

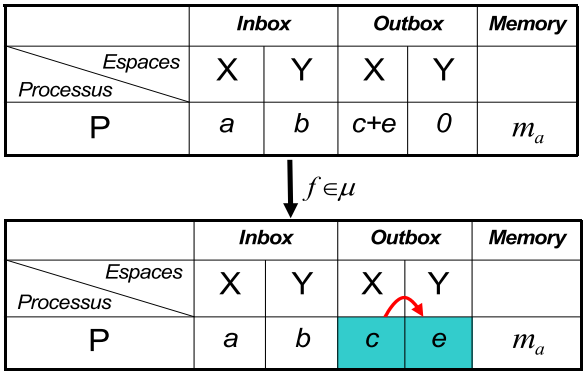

FIG. 8.4 – Application d'une loi de mouvement entre deux espaces d'interaction.

Interaction. L'interaction est une application φ correspondant à une transformation matricielle qui, à partir de la valeur des matrices d'émission, calcule la nouvelle valeur des matrices de perception et d'émission. Plus intuitivement, il s'agit là de calculer le résultat d'une

loi d'interaction locale engendrée par la confrontation des objets d'interaction appartenant à des agents se situant dans un même espace d'interaction. Ce résultat peut alors engendrer la création d'objets d'interaction dans les boîtes de réception des agents concernés. A l'instar d'une loi de mouvement, une loi d'interaction est définie localement étant donné un espace d'interaction. Plus formellement, soient a et b deux agents et j un espace d'interaction, l'application d'une règle d'interaction sur j est une fonction $interaction_j \in \varphi$ telle que :

$$interaction_j(out[a,j], out[b,j]) = in[a,j] \tag{8.2}$$

La figure 8.5 illustre l'application d'une telle loi sur l'espace d'interaction Y entre les objets d'interaction b et g. Dans ce schéma, l'objet d'interaction g peut par exemple représenter un message émis par l'agent Q qui interagit avec la représentation b de l'agent P sur l'espace Y, ce qui engendre la création de ce message dans la inbox de P.

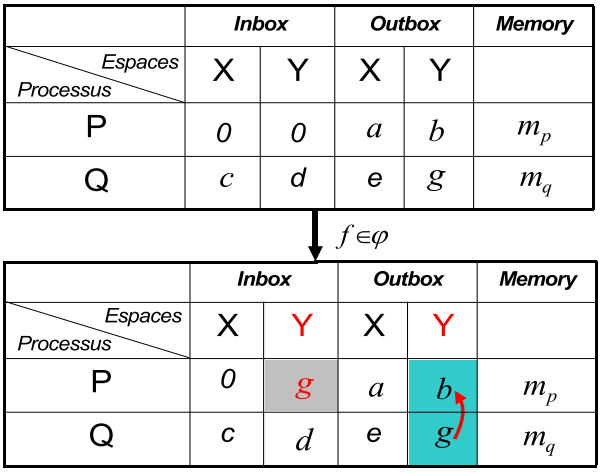

FIG. 8.5 – Interaction entre deux processus P et Q sur l'espace d'interaction Y.

Calcul. Le calcul, ou computation, est une fonction γ qui matérialise l'activation du processus décisionnel d'un agent. Elle correspond à la consommation des objets d'interaction qui se trouvent dans sa inbox –la perception–, à la modification de sa outbox –le résultat de la délibération–, et à la modification de sa mémoire –changement d'état interne–. Ainsi, c'est par la création/modification d'objets d'interaction qu'un agent tente d'influencer son environnement : il modifie sa représentation pour signifier son propre calcul interne. Soient a un agent et $inbox[a, \mathcal{J}]$ et $outbox[a, \mathcal{J}]$ les valeurs globales de sa inbox et de sa outbox sur tous les espaces d'interaction et m_a l'état de sa mémoire, le calcul de a correspond à l'application d'une fonction $computation_a \in \gamma$ telle que :

$$computation_a(inbox[a, \mathcal{J}], m_a) = (inbox'[a, \mathcal{J}] = 0) \times outbox'[a, \mathcal{J}] \times m'_a \tag{8.3}$$

La figure 8.6 illustre le résultat de la computation d'un agent P étant donnés les espaces d'interaction où il se trouve. La suppression des objets contenus dans la inbox et la création/-modification de nouveaux objets contenus dans la outbox constituent la seule manifestation observable du comportement d'un agent. Du point de vue de l'environnement, la structure interne d'un agent demeure inconnue.

Evolution globale d'un terme MIC*

L'évolution globale d'un terme MIC* à travers le temps correspond à l'application successive de ces trois transformations dans un ordre prédéterminé (ce qui sera le cas pour les

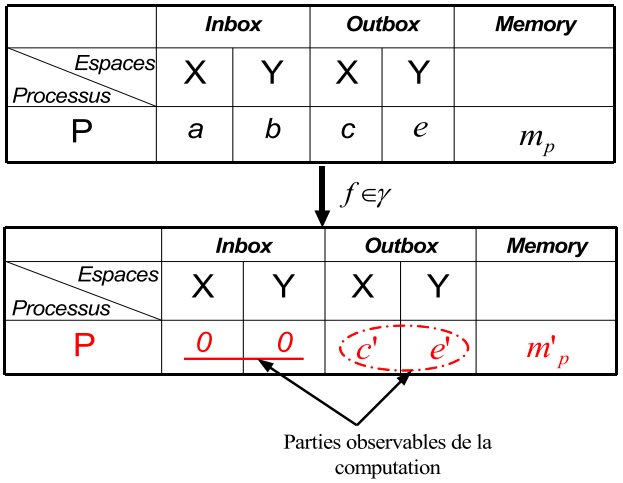

FIG. 8.6 – Computation d'un agent P : P vide sa inbox puis modifie sa mémoire et sa outbox.

applications de simulation) ou engendré par la réaction de l'environnement aux computations des agents dans le cas d'un système informatique classique (avec des agents évoluant à un rythme non contrôlé). La dynamique d'un terme MIC* correspond ainsi à l'application d'une suite de fonctions sur un terme MIC* initial, $\mu \circ \varphi \circ \gamma \circ \gamma \circ \gamma \circ \varphi \circ \mu \circ \gamma \circ \varphi$ par exemple (c'est-à-dire $M \circ I \circ C \circ C \circ C \circ I \circ M \circ C \circ I$).

8.3.3 Implémentations et disponibilité du modèle MIC*

Depuis son élaboration, le modèle MIC* a fait l'objet de plusieurs prototypes. Aujourd'hui, il existe essentiellement deux versions du modèle, l'une en Java et l'autre en C++. Le projet étant développé sur le principe du logiciel libre, ces différentes versions peuvent être récupérées sur la page Internet du projet : sourceforge.net/projects/mic. La figure 8.7 présente une version élaguée du diagramme de classe correspondant à la version du noyau Java qui implémente les principes du modèle MIC* [4].

8.4 Exemple : le *Jeu de la vie* en MIC*

8.4.1 Principe du *Jeu de la vie*

Décrit pour la première fois par John Conway dans la fin des années 60, le *Jeu de la vie* fait partie des applications qui concernent le domaine des automates cellulaires. L'automate cellulaire du Jeu de la vie est défini par une grille régulière où chaque cellule possède un état binaire : *vivante* ou *morte*. Une cellule a pour voisines les huit cases adjacentes (voisinage de *Moore*). Les règles d'évolution de chaque cellule sont les suivantes :

- lorsque la cellule est morte : si elle a exactement trois voisines vivantes, la cellule devient vivante. Dans les autres cas elle reste morte.
- lorsque la cellule est vivante : si elle a deux ou trois voisines vivantes, la cellule reste vivante. Dans les autres cas elle meurt.

[4] Il faut noter que ce diagramme peut ne plus correspondre exactement aux versions actuellement implémentées étant donné que les prototypes sont régulièrement modifiés.

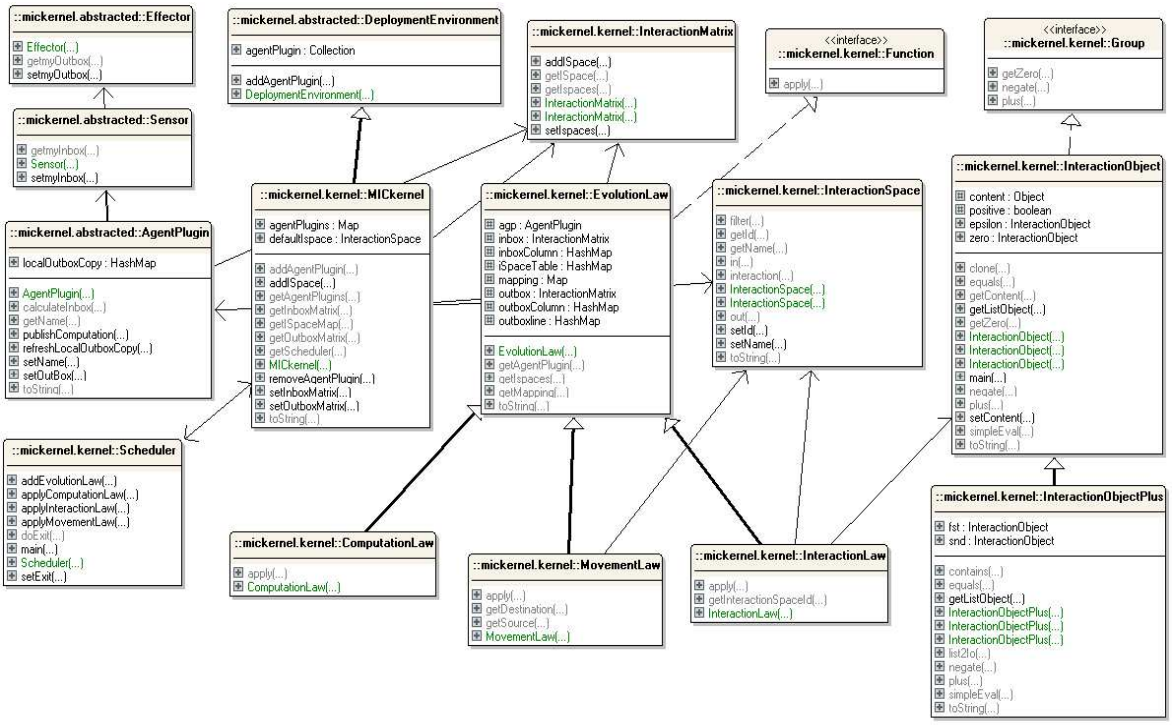

FIG. 8.7 – Implémentation du modèle MIC*.

Dans ce système, l'ensemble des cellules évolue de façon synchrone. Par ailleurs, la grille est toroïdale, c'est-à-dire que les cellules qui se trouvent au bord de la grille ont pour voisines les cases qui se trouvent de l'autre côté, y compris en diagonale. La figure 8.8 montre un exemple de cette dynamique pour six pas de temps.

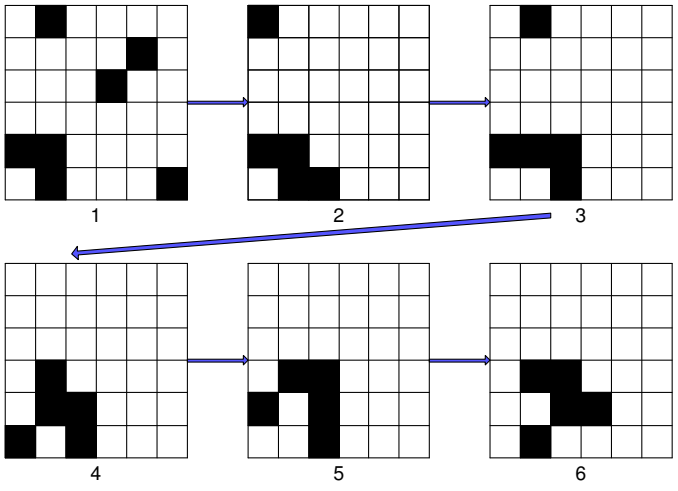

FIG. 8.8 – Exemple d'évolution de l'automate cellulaire du Jeu de la vie.

Pour illustrer la manière dont on peut modéliser un système à l'aide du modèle MIC*, nous allons maintenant modéliser cet automate cellulaire et sa dynamique en définissant un ensemble d'agents et un environnement de déploiement adéquats. Avant d'aller plus loin, il est important de noter que la modélisation que nous allons proposer ne représente qu'une solution parmi les nombreuses façons dont il est possible de modéliser ce système à l'aide de MIC*.

8.4.2 Description statique

Pour modéliser ce système, nous allons tout d'abord considérer qu'à chaque cellule de coordonnées (x, y) correspond un espace d'interaction $J_{(x,y)}$. A cela nous ajoutons un espace d'interaction particulier J_0 qui, bien que non indispensable, nous permettra d'illustrer une loi de mouvement.

Les espaces d'interaction que nous venons de définir seront peuplés par des agents dont le processus de calcul représentera la dynamique d'une cellule. On a donc un agent par cellule. Cependant, un agent sera non seulement présent dans l'espace d'interaction de la cellule qu'il représente mais aussi dans chaque espace d'interaction définissant les cellules voisines. Ce qui lui permettra d'interagir avec ses voisins. Pour cela, la outbox d'un agent sera définie par l'un ou l'autre des deux objets d'interaction suivants :

- *CellOwner* qui décrit la représentation de l'agent dans l'espace d'interaction de la cellule qu'il habite (une information booléenne : vivant au mort).
- *Neighbor* qui définit la représentation de l'agent dans les espaces d'interaction représentant les cellules voisines.

Pour peupler la grille, nous allons utiliser une loi de mouvement. Pour cela, tous les agents sont initialement placés dans l'espace d'interaction J_0 avec une représentation, un objet d'interaction *InitialAgentRep*, qui va engendrer pour chaque agent les neuf mouvements nécessaires à leur placement dans les bons espaces d'interaction et avec les bons objets d'interaction. La figure 8.9 illustre cette phase initiale et montre la manière dont les boîtes d'émission de l'agent représentant la cellule $(2, 2)$ sont définies sur chacun des neufs espaces d'interaction où l'agent existe.

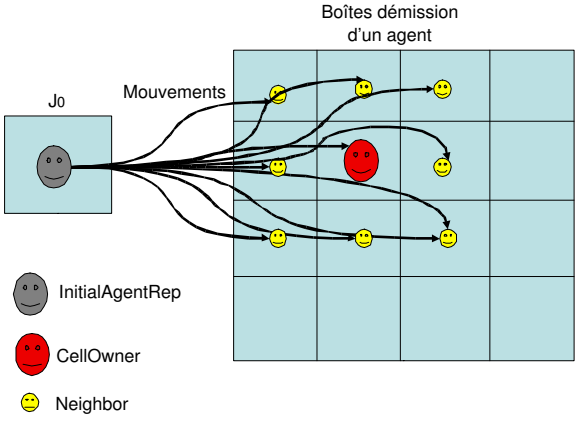

FIG. 8.9 – Un agent est présent dans neuf espaces d'interaction : sa cellule et ses voisines.

8.4.3 Description de la dynamique

Pour obtenir la dynamique du Jeu de la vie, nous allons maintenant définir la manière dont interagissent les différentes représentations des agents. Pour cela, nous allons spécifier une unique loi d'interaction très simple, commune à tous les espaces d'interaction, qui fait intervenir les deux objets d'interaction précédents. Nous allons poser qu'un objet d'interaction *CellOwner* réagit à un objet d'interaction *Neighbor* de manière à créer, dans la inbox de l'agent concerné, un nouvel objet d'interaction *NeighborComputation* qui va représenter la perception qu'un agent a de son voisin, ce qui lui permettra de connaître

son état. Plus formellement, cette loi correspond à l'application de la fonction suivante : $interaction_{jdl}(CellOwner, Neighbor) = NeighborComputation$. En Java par exemple, cette loi d'interaction se résume au code suivant :

```
public InteractionObject interaction(InteractionObject target, InteractionObject from)
{
        if(target instanceof CellOwner && from instanceof Neighbor)
                return target.interaction(from); //returns a NeighborComputation IO
        return InteractionObject.zero;
}
```

Ainsi, lorsque cette loi est appliquée à l'ensemble des espaces d'interaction et pour tous les couples d'objets d'interaction possibles, tous les agents obtiennent dans leur inbox un ensemble d'objets d'interaction *NeighborComputation* qui représente l'état dans lequel se trouvent ses voisins. La figure 8.10 illustre l'application de cette loi sur l'espace d'interaction de coordonnées $(2, 2)$.

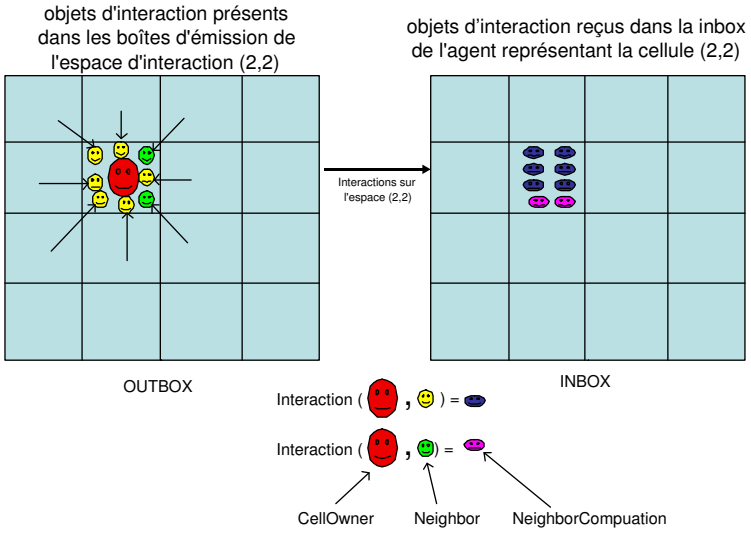

FIG. 8.10 – Application de la loi d'interaction sur l'espace de coordonnées $(2, 2)$.

A partir de là, un agent est donc capable de percevoir via sa inbox l'état dans lequel se trouvent ses voisins. Le deuxième temps de la dynamique du système consiste alors à faire calculer l'ensemble des agents pour que ceux-ci mettent à jour leurs représentations dans les différents espaces où il se trouve. Pour cela, un agent effectue un calcul très simple pour savoir l'état dans lequel il doit se trouver en appliquant les règles du Jeu de la vie.

Voici le code utilisé dans la structure interne d'un agent qui correspond à ce calcul[5] :

```
public void compute()
{
Map inbox = myPlugin.calculateInbox();
InteractionObject result=null;
for(Iterator i = inbox.values().iterator();i.hasNext();)
{
        result = (InteractionObjectPlus) i.next();
}
Debug.log("computing"+result);
Vector v = new Vector();
result.getListObject(v);
```

[5]La complexité apparente de ce code est principalement due au fait que le moteur MIC* écrit en Java est une version expérimentale ou de très nombreuses choses restent à encapsuler.

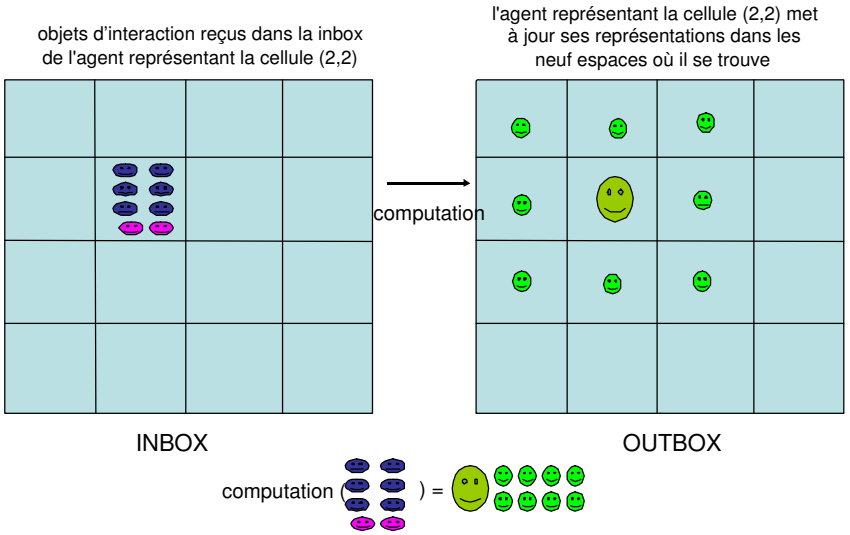

FIG. 8.11 – Computation dun agent d'un agent : perception via la inbox puis mise à jour de ses représentations dans sa outbox.

```
int aliveNeighbors = 0;
for(Iterator i=v.iterator();i.hasNext();)
        if( ((NeighborComputation) i.next()).getState())
                aliveNeighbors++;
if ( state && (aliveNeighbors < 2 || aliveNeighbors >3))
        state=false;
else if (aliveNeighbors == 3)
        state=true;
myPlugin.refreshLocalOutboxCopy();
for(Iterator i=getAgentPlugin().localOutboxCopy.entrySet().iterator(); i.hasNext(); )
{
        Map.Entry e = (Map.Entry) i.next();
        if(e.getValue() instanceof CellOwner)
                e.setValue(new CellOwner(state));
        else
                e.setValue(new Neighbor(state));
}
Debug.log("computing 2"+getAgentPlugin().localOutboxCopy);
getAgentPlugin().publishComputation();
}
```

Finalement, pour obtenir la dynamique globale qui correspond à l'évolution de l'automate cellulaire du Jeu de la vie, il suffit d'alterner l'application de la loi d'interaction avec la loi de computation des agents. La loi d'interaction permet dans un premier temps de récupérer l'état des voisins qui est ensuite utilisé dans un deuxième temps pour calculer l'état suivant d'un agent. Autrement dit, du point de vue du modèle MIC*, on applique la séquence suivante : $(I \circ C)*$. Dans l'annexe A, nous donnons quasiment l'intégralité du code correspondant à cette application : les objets d'interaction (section A.1 page 225), les espaces d'interaction (section A.2 page 226), le processus de calcul correspondant à une cellule (section A.3 page 227) et le programme principal qui contrôle la dynamique globale du système (section A.4 page 229).

8.4.4 Remarques

La modélisation que nous venons de présenter se distingue de la façon dont on programme généralement un automate cellulaire en intégrant les principes du paradigme agent. Tout

d'abord, il est intéressant de noter que les agents n'ont aucune représentation de leur localisation absolue, c'est-à-dire de leurs coordonnées dans l'environnement global : ils ne possèdent pas l'information (i, j). Par ailleurs, ils n'ont pas d'accès direct à l'état de leurs voisins et agissent uniquement en fonction de la perception qu'ils en ont. Leur perception est uniquement basée sur des critères locaux.

Autre point très important, dans un contexte classique, la dynamique du Jeu de la vie nécessite l'utilisation d'un état tampon dans lequel on stocke les nouvelles valeurs qui sont calculées pour l'état suivant d'une cellule. En effet, il est nécessaire de ne pas mettre à jour directement l'état d'une cellule de manière à ce que les cellules voisines puissent effectuer leurs propres calculs sur les bonnes données. Ici, cela n'a pas été nécessaire dans le sens où un agent peut mettre à jour son état interne et ses représentations dès le moment du calcul.

En fait, l'une des plus importantes caractéristiques du modèle MIC* repose sur le fait qu'il propose une distinction claire entre le **temps de l'interaction** et **le temps du calcul**. En fait, il s'agit ici bien sûr d'une mise en œuvre du principe Influence/Réaction. En effet, outre le fait que les objets d'interaction sont utilisés pour signifier l'existence des agents dans le système, ils peuvent aussi être vus comme des influences produites par le calcul des agents. La réaction de l'environnement à ces influences correspond alors à l'application des règles d'interaction sur les différents objets d'interaction.

MIC* ne propose pas uniquement une manière adéquate d'appliquer le principe Influence/Réaction mais il constitue aussi une formalisation élégante des différentes solutions que nous avons envisagées pour les contraintes que nous avons énumérées au début de ce chapitre. Nous allons maintenant mettre cela en évidence.

8.5 Adéquation du modèle MIC* avec les principes de modélisation multi-agents proposés

8.5.1 inbox et outbox : une réification des notions de capteurs et effecteurs

La inbox et la outbox d'un agent sont bien sûr une généralisation des notions de capteurs et d'effecteurs. Ils constituent l'unique lien qui existe entre un agent et son environnement. Un agent perçoit le monde par l'intermédiaire de sa inbox et il agit sur celui-ci en émettant des objets d'interaction dans sa outbox.

8.5.2 Distinction esprit/corps et intégrité interne d'un agent

Une séparation claire entre le système conatif et la partie physique d'un agent est aussi une conséquence directe de l'utilisation du modèle MIC*. En effet, ce qui est appelé processus de calcul constitue sans ambiguïté ce qui correspond au système décisionnel de l'agent et aucune autre entité du système n'a accès aux variables qui sont utilisées par le processus en interne. L'intégrité interne d'un agent et le respect de son processus délibératif sont donc garantis.

De la même manière, la partie physique de l'agent est entièrement décrite par les objets d'interaction qui se trouvent dans sa outbox. Ces objets constituent la représentation de l'agent dans l'environnement et caractérisent ses traits physiques. C'est la partie visible de l'agent.

8.5.3 Respect de la contrainte d'intégrité environnementale

Pour respecter cette contrainte, il est nécessaire qu'un agent ne soit pas en mesure de modifier directement les objets d'interaction de sa outbox car nous venons de voir que ces objets représentent sa composante physique. Et comme nous l'avons plusieurs fois mentionné, cette composante fait partie intégrante de l'environnement. En fait, nous n'avons pas donné l'ensemble du formalisme MIC* pour des raisons de simplicité de compréhension. Dans les faits, un espace d'interaction définit aussi des fonctions de filtrage qui sont appliquées sur les sorties produites par la computation des agents. Autrement dit, lorsqu'un agent modifie sa représentation en émettant des objets d'interaction dans sa outbox, ces objets sont tout d'abord filtrés par l'environnement avant de se retrouver effectivement dans l'espace d'interaction. Ce qui peut permettre de rajouter des informations supplémentaires par exemple. Finalement, l'agent propose et l'environnement dispose[6]. En ce qui nous concerne, nous n'aurons pas vraiment besoin d'utiliser cette fonction de filtrage car nous nous trouvons ici dans le cadre de la simulation où l'ensemble de la modélisation est contrôlé. Ainsi, les seuls objets d'interactions qui seront produits par les agents après computation correspondront pour nous à des influences. Les agents ne modifieront donc jamais directement les variables d'état de leur composante physique ou de l'environnement.

8.5.4 Modélisation suivant les quatre modules

L'utilisation du modèle MIC* va nous permettre d'appliquer très simplement un principe de modélisation et de programmation suivant quatre modules. Le module des comportements sera bien sûr modélisé par les processus de calcul représentant les agents. Les variables utilisées par ces processus de calcul représenteront l'état interne des agents, autrement dit leurs états mentaux. Dans MIC*, les variables d'état représentant l'environnement situé, à ne pas confondre avec la notion d'environnement de déploiement (la structure MIC* elle-même), seront entièrement définies grâce aux objets d'interactions qui se trouvent dans le système, qu'il s'agisse de l'état physique des agents ou de l'état de l'environnement. Lorsqu'elle est non nulle, la dynamique de l'environnement peut quant à elle être modélisée par un processus de calcul particulier ayant une dynamique invariante. Ce qui sera notre cas lorsque nous appliquerons le principe Influence/Réaction où nous aurons un processus de calcul représentant l'environnement et dédié au calcul de la réaction. Ce qui permettra par ailleurs aussi de modéliser l'évolution endogène de l'environnement.

En ce qui concerne la modélisation du temps, le modèle MIC* ne définit aucune technique a priori. Les différentes lois qui définissent la dynamique d'un terme MIC* représentent uniquement le moyen de faire évoluer le système. Comme nous le verrons, il est donc possible de définir en toute liberté la manière dont ces différentes lois sont appliquées, et ainsi d'utiliser des dynamiques temporelles discrètes ou basées sur un principe événementiel dans les applications de simulation. Point très important pour nous, le module des interactions est quant à lui entièrement défini en dehors de ces autres problématiques. Il consiste en effet à définir très clairement la manière dont les différents objets d'interaction sont mis à jour par les processus de calcul et interagissent suivant les lois d'interaction qui ont été définies. La figure 8.12 résume ces différents points de manière schématique.

[6] Ce procédé est notamment très utile dans le cadre de l'ingénierie des systèmes ouverts. Cela permet de vérifier que les agents, dans lesquels il ne faut par définition pas avoir confiance, n'émettent pas des objets d'interaction susceptibles de contrevenir au fonctionnement du système. Par exemple, dans une application de discussions ubiquistes, il est possible de filtrer les messages suivant une liste de mots interdits [Gouaïch *et al.* , 2003].

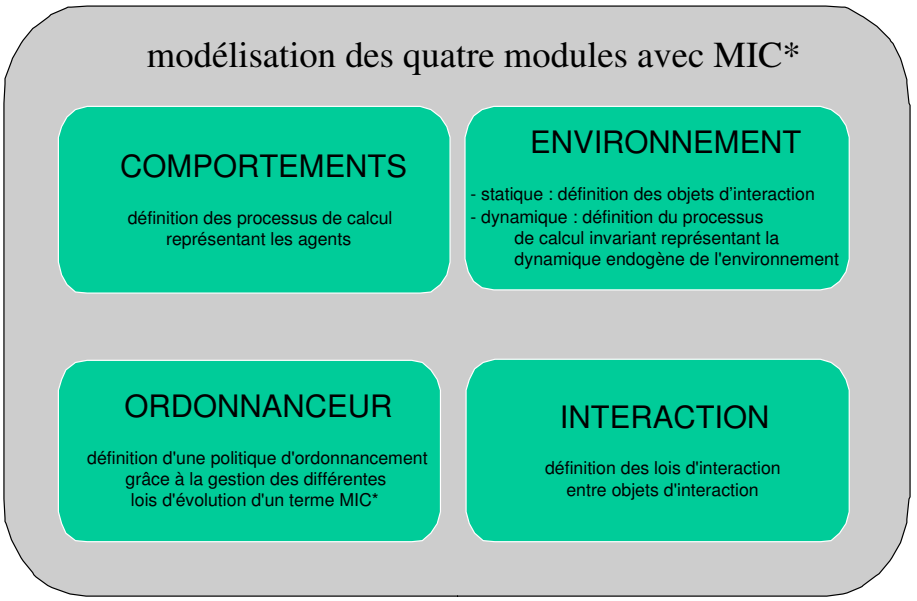

FIG. 8.12 – Modélisation des quatre modules avec le modèle MIC*.

8.6 Résumé du chapitre

Dans ce chapitre nous avons tout d'abord récapitulé les différents points de l'analyse que nous avons conduite tout au long de ce document. Nous avons ensuite présenté les principes fondateurs du modèle MIC* et nous avons détaillé les concepts et les mécanismes sous-tendus par son application. Finalement, nous avons donné l'ensemble des raisons pour lesquelles MIC* constitue un modèle qui permet d'appliquer les différentes contraintes de modélisation que nous avons identifiées. Nous avons notamment montré pourquoi le respect de ces contraintes est une conséquence naturelle de la structure du modèle MIC* et donc de son utilisation. Nous allons maintenant voir concrètement comment l'ensemble de ces différents aspects du modèle MIC* peuvent être mis en œuvre dans le contexte de la modélisation et de la simulation d'un système représentant une société artificielle de type *SugarScape*.

Chapitre 9

Modélisation et simulation d'une société artificielle d'agents de type *SugarScape*

DANS ce chapitre nous allons présenter une expérience de simulation multi-agents complète en détaillant les principales étapes de sa conception. Dans cette expérimentation, nous étudierons un système de type *SugarScape* ([Epstein & Axtell, 1996]) représentant une société artificielle d'agents et son évolution. A l'instar du travail exposé par Lawson et Park dans [Lawson & Park, 2000] (que nous allons maintenant abréger par L-P), notre objectif n'est pas d'élaborer un modèle de société artificielle complexe et original. Au contraire, notre but est ici d'étudier la modélisation et la simulation d'un système simple au regard des différents principes, contraintes et réflexions que nous avons précédemment exposés. De plus, pour pouvoir comparer notre approche avec ce qui a déjà été fait, nous allons reprendre en grande partie l'adaptation de *SugarScape* proposée par L-P de manière à posséder un point de référence. L'idée est de coller au plus près à ce modèle tout en intégrant nos principes de modélisation. Ainsi, la différence majeure entre ce précédent modèle et celui que nous allons donner tient dans la manière dont nous allons modéliser la dynamique du système, notamment en ce qui concerne la façon dont les interactions seront représentées. Il ne s'agit donc pas exactement d'une réplication dans le sens où nous allons volontairement modifier la dynamique du modèle. C'est pourquoi les résultats que nous obtiendrons seront quantitativement incomparables avec ceux qui ont été obtenus précédemment. Nous ferons cependant une comparaison qualitative entre les résultats obtenus par les deux approches. L'idée est ici que, issus d'un unique système source, il sera intéressant de comparer les comportements globaux dérivés par ces différentes modélisations. Autrement dit, à l'image de la comparaison effectuée par L-P entre leur adaptation et le modèle original, nous essaierons de déterminer dans quelle mesure le résultat de notre modélisation se rapproche, ou s'éloigne, de leur modèle.

Dans un premier temps nous présenterons le système source que nous souhaitons étudier et nous définirons le cadre expérimental dans lequel nous considérerons sa modélisation et sa simulation. Ensuite, nous proposerons une modélisation de ce système fondée sur les différentes approches que nous avons proposées : le modèle MIC*, le principe Influence/Réaction, etc. Finalement, nous présenterons un simulateur de ce modèle et nous discuterons des différents résultats obtenus par son utilisation.

9.1 Système source et cadre expérimental

9.1.1 Système source

Le système que nous allons étudier est une énième version du modèle de société artificielle proposé par [Epstein & Axtell, 1996]. Il sera plus précisément basé sur l'adaptation qui en a été faite par L-P[1]. Comme c'est souvent le cas dans le domaine de la vie artificielle, le système source[2] correspondant à cette société artificielle n'existe pas dans le sens où il n'a aucune réalité en dehors du modèle que nous allons en faire. Autrement dit, bien qu'il soit inspiré par des phénomènes que l'on trouve dans la nature, nous ne disposons a priori d'aucune donnée d'observation : la *base de données comportementale* du système n'existe pas (cf. section 2.8.3 page 40). Il ne s'agit donc pas de reproduire l'évolution d'un système réel au sens physique du terme. Dans ce contexte, nous allons ici simplement donner une description de ce système en langage naturel.

Le système que nous nous proposons de modéliser consiste dans un environnement situé en deux dimensions dans lequel un ensemble d'agents mobiles consomment une ressource répartie dans l'environnement de manière à assurer leur survie. Par ailleurs, ces agents sont capables de se reproduire de manière sexuée pour assurer la survie de l'espèce, la durée de vie d'une entité étant limitée. L'environnement sur lequel évoluent les agents est un espace discrétisé en cellules régulières sur lesquelles une certaine quantité de la ressource est présente. Lorsqu'un agent occupe une cellule, il peut alors consommer la ressource qui y est présente pour remettre à niveau son énergie qui décroît au fur et à mesure. La perception d'un agent consiste à obtenir des informations sur les cellules qui l'entourent et qui se trouvent dans son champ de vision (un certain nombre de cellules dans les quatre directions cardinales) : niveau de la ressource et occupants. Lorsque la ressource d'une cellule est consommée, elle se régénère ensuite jusqu'à un maximum de capacité défini pour chaque cellule. L'ensemble du système évolue dans le temps suivant les différentes actions perpétrées par les agents (mouvements, reproduction, consommation) et par la régénération des ressources. Nous allons maintenant définir dans quel cadre expérimental nous allons considérer ce système.

9.1.2 Cadre expérimental

Comme nous l'avons dit, le propos de notre expérimentation se focalise sur la manière dont nous allons modéliser cette société artificielle en fonction des principes que nous avons précédemment évoqués. Il n'est donc pas question pour nous que ce système représente une hypothétique réalité ou que les mécanismes que nous allons décrire puissent être considérés comme réalistes (les déplacements par exemple). Cependant, il faut garder à l'esprit que nous conserverons une attention toute particulière au respect de la cohérence paradigmatique du modèle que nous allons élaboré. Ce qui constitue notre objectif premier. C'est pourquoi nous ne ferons pas une analyse poussée des différentes variations qu'il est possible de faire sur la modélisation de la dynamique, comme cela a été fait dans [Meurisse, 2004] par exemple. Notre objectif est ici purement didactique et concerne principalement la modélisation plus que les résultats obtenus. Il s'agit pour nous de montrer la faisabilité de notre approche : modélisation en quatre modules distincts, respect des contraintes de modélisation, utilisation du principe Influence/Réaction et du modèle MIC*.

[1] On trouve une description plus complète du système source et du modèle proposé par L-P à l'adresse Internet www.cs.wm.edu/~bglaws/research/model.htm.

[2] Le système source est aussi souvent désigné par les termes *modèle conceptuel*.

Etant donné que nous nous basons sur le modèle proposé par L-P, il sera cependant intéressant de comparer le comportement de ces différentes modélisations. Pour cela, il nous faut bien sûr un point de référence qui nous permettra d'évaluer les différentes exécutions que nous ferons lors des simulations. Dans cette optique, la principale donnée d'intérêt sera définie par le comportement de l'évolution de la population globale des agents au cours du temps. Cette valeur nous donnera ainsi un point de vue simple et global sur le système. Cela nous permettra d'une part de mesurer la sensibilité du modèle aux différentes variations que nous en ferons, et d'autre part de comparer notre approche avec celle de L-P. Par ailleurs, comme dans L-P, nous définirons deux modèles temporels de l'évolution du système. Le premier sera basé sur une discrétisation régulière du temps et le deuxième consistera à utiliser un principe de simulation par événement. Pour résumer, le cadre expérimental de notre expérience concerne l'analyse de la modélisation et de la simulation plus que l'analyse des résultats en soi.

9.2 Modélisation

9.2.1 Méthodologie

Pour élaborer le modèle du système que nous venons de présenter, nous allons utiliser les différents principes de modélisation que nous avons établis. Nous allons notamment présenter une modélisation qui suit le principe de séparation en quatre modules. Par ailleurs, pour chacun de ces modules, nous donnerons d'abord une description mathématique basée sur l'utilisation du principe Influence/Réaction avant de donner la modélisation qui lui correspond dans le modèle MIC*.

9.2.2 Module *Environnement*

Dans cette section, nous allons décrire la modélisation de l'environnement dans son ensemble, c'est-à-dire la partie physique des agents et les différents objets du monde. Il s'agit donc ici de spécifier l'ensemble des concepts et notations que nous utiliserons pour décrire les variables d'état de l'environnement $\sigma \in \Sigma$ à tout instant de la simulation.

Modélisation de l'environnement situé

Le monde dans lequel vont évoluer les agents consiste dans un espace discrétisé sous la forme d'une grille de cellules possédant chacune une quantité de ressources variable et limitée. Plus formellement, nous utiliserons les notations suivantes :

- $X \in \mathbb{N}^*$ et $Y \in \mathbb{N}^*$ sont respectivement les dimensions de la grille en abscisse et en ordonnée.
- chaque cellule est repérée par ses coordonnées (x, y) avec $x \in [0, X-1]$ et $y \in [0, Y-1]$.

Par ailleurs, nous considérons une grille toroïdale où les cellules qui se trouvent au bord de la grille ont pour voisines les cases qui se trouvent de l'autre côté, y compris en diagonale. Autrement dit, quelles que soient $x' \in \mathbb{Z}$ et $y' \in \mathbb{Z}$, les coordonnées (x', y') représentent en fait la cellule de coordonnées (x, y) avec $x = ((x' \bmod X) + X) \bmod X$ et $y = ((y' \bmod Y) + Y) \bmod Y$. De plus, l'origine de la grille se trouvera en bas à gauche[3].

[3] Au contraire de L-P ou l'origine se trouve en haut à gauche. Il s'agit pour nous d'avoir un repère où l'on puisse utiliser la notion de sens trigonométrique.

Chaque cellule est caractérisée par sa capacité maximale, son taux de régénération et une variable d'état représentant la quantité courante de ressource disponible.

Plus formellement, nous utiliserons les notations suivantes :

- $C_{(x,y)} \in \mathbb{R}^+$: la capacité maximale de la cellule (x, y).
- $\rho_{(x,y)} \in \mathbb{R}^+$: le taux de régénération de la cellule (x, y).
- $R_{(x,y)}(t) \in \mathbb{R}^+$ avec $R_{(x,y)}(t) \leq C(x, y)$: la quantité de ressource à l'instant t.

Modélisation de la partie physique des agents

Nous allons ici décrire l'ensemble des variables qui définissent la partie physique des agents et qui font partie intégrante de l'environnement. Le corps d'un agent est ainsi modélisé par plusieurs caractéristiques qui lui sont propres, des constantes, ainsi que par un ensemble de variables d'état qui représentent sa condition physique courante.

Par ailleurs, chaque agent sera repéré par un numéro qui correspondra à son ordre d'instanciation et qui définira ainsi un ordre total entre les agents.

A partir du modèle de L-P[4], nous modélisons le corps d'un agent A_a de numéro a de la façon suivante :

- S_a : le sexe de l'agent a représenté par une variable booléenne (vraie pour mâle).
- $\beta_a \in \mathbb{R}^+$: la date de naissance de l'agent a.
- $\lambda_a \in \mathbb{N}^*$: la durée de vie de l'agent a avec $\lambda_a \in [60, 100]$.
- $\alpha_a \in \mathbb{N}^*$: l'âge où la capacité de se reproduire commence avec $\alpha_a \in [12, 15]$.
- $\omega_a \in \mathbb{N}^*$: l'âge où la capacité de se reproduire se termine, avec $\omega_a \in [40, 50]$ pour les femelles et $\omega_a \in [50, 60]$ pour les mâles.
- $\eta \in \mathbb{N}$: la durée de la gestation commune à toutes les femelles[5].
- $\Phi_a \in \mathbb{N}^*$: le champ de vision de l'agent a en nombre de cellules dans les quatre directions cardinales, avec $\Phi_a \in [1, 6]$.
- μ_a : la quantité d'énergie consommée par unité de temps par l'agent a.
- $w_a(t) \in \mathbb{R}^+$: le niveau d'énergie de l'agent a à l'instant t. Tous les agents commenceront avec $w_a(0) = 10$.
- $L_a(t) \in < X \times Y >$: la cellule occupée par l'agent a à l'instant t.

A l'image de L-P, la fertilité d'un agent sera considérée comme effective si les conditions suivantes sont remplies :

- l'agent est en âge de procréer : $\beta_a + \alpha_a \leq t \leq \beta_a + \omega_a$.
- l'agent n'est pas en train de se reproduire.
- l'agent ne mourra pas de mort naturelle avant la fin de la gestation : $t + \eta_a < \beta_a + \omega_a$.
- au moment de la naissance, l'agent possédera un niveau d'énergie supérieure à son niveau initial : $w_a(\beta_a) < w_a(t + \eta) = w_a(t) + \eta(\rho_{(L_a(t))} - \mu_a)$.
- il existe une cellule non occupée dans le voisinage de Von Neumann de l'agent.

[4]Nous n'avons pas été en mesure de reprendre l'ensemble de la notation utilisée par L-P pour des raisons de conflit avec les notations utilisées pour décrire le modèle Influence/Réaction

[5]Rappelons ici qu'il s'agit là d'un paramètre introduit par L-P et qu'il ne se trouve pas dans le modèle *SugarScape* original (cf. section 7.2.1 page 142). Cependant, lorsque $\eta = 0$, l'adaptation proposée par L-P est censée être équivalente.

Modélisation MIC* de l'environnement

Objets d'interaction. Du point de vue du modèle MIC*, la grille de cellules sera entièrement encapsulée dans un objet d'interaction appelé *Landscape*. C'est avec cet objet d'interaction que la représentation physique des agents va interagir, à la fois pour percevoir et produire des influences. De la même manière, la partie physique d'un agent sera encapsulée dans un objet d'interaction appelé $Body_a$. Il y aura ainsi un objet d'interaction de type *Body* pour chaque agent présent dans le système. A ce stade de la modélisation, on a donc un premier ensemble d'objets d'interaction appelé $\mathcal{O}_{physical}$ qui est tel que $\mathcal{O}_{physical} = \{\bigcup_i Body_i \cup Landscape\}$. En ce qui concerne la notation Influence/Réaction, $\sigma \in \Sigma$ correspond ainsi très exactement à l'ensemble des variables d'état contenu dans ces objets d'interaction.

Espaces d'interaction. Pour pouvoir faire interagir ces différents objets d'interaction, il nous faut ici définir un espace d'interaction appelée *world* qui concrétise un premier lieu logique dans lequel nous ferons interagir ces différentes entités. Par ailleurs, nous allons ici quelque peu anticiper sur la modélisation des modules suivants de manière à pouvoir donner immédiatement une première représentation schématique du terme MIC* correspondant au modèle que nous élaborons. Cela nous permettra aussi de faire évoluer cet exemple au fur et à mesure et ainsi de permettre au lecteur d'avoir une idée plus concrète de la modélisation MIC* sous-jacente. C'est pourquoi nous allons tout de suite introduire deux espaces d'interaction supplémentaires : l'espace des influences de mouvement *moves* et l'espace des influences de reproduction *repro*. Autrement dit, soit $\mathcal{J}$ l'ensemble des espaces d'interaction, on a $\mathcal{J} = \{world, moves, repro\}$.

Processus de calcul. Pour les mêmes raisons, nous allons aussi donner immédiatement l'ensemble du processus de calcul associé à la modélisation MIC*. Dans notre modèle, nous aurons un processus de calcul A_i par agent et un processus de calcul particulier *Environment* qui représentera la dynamique de l'environnement. Autrement dit, soit $\mathcal{I}$ l'ensemble des processus de calcul, on a $\mathcal{I} = \{A_1, A_2, ..., A_n, Environment\}$. Le processus de calcul A_i associé à l'agent i modélise son comportement proprement dit, c'est-à-dire son esprit. Le processus de calcul *Environment* va quant à lui représenter la dynamique de l'environnement, c'est-à-dire son évolution endogène et la réaction de celui-ci à l'ensemble des influences.

Nous pouvons maintenant donner une première représentation schématique de cette modélisation. La figure 9.1 illustre le terme MIC* qui lui correspond.

	inbox			outbox		
Espaces / Processus	World	Moves	Repro	World	Moves	Repro
environment				Landscape	Landscape	Landscape
$Agent_1$				$Body_1$		
$Agent_2$				$Body_2$		
$Agent_3$				$Body_3$		

FIG. 9.1 – Modélisation des objets de l'environnement sous la forme d'un terme MIC*.

Comme on peut le voir sur cette figure, chaque agent existe dans le système grâce aux objets d'interaction $Body_n$ situés dans l'espace d'interaction *world*. L'environnement possède quant à lui une représentation dans tous les espaces d'interaction définis grâce à l'objet d'interaction *Landscape*. Ce qui nous permettra de faire interagir l'environnement avec les influences produites par les agents.

A titre de remarque, il est intéressant de noter que nous aurions très bien pu nous contenter de définir un seul espace d'interaction par exemple. Encore une fois, un espace d'interaction ne doit pas être compris comme un lieu au sens physique du terme mais comme un espace logique qui permet de décomposer le système de manière à simplifier le calcul des interactions. L'idée qui se cache ici derrière la définition de plusieurs espaces d'interaction repose sur le fait que cela va nous permettre d'effectuer une classification automatique des influences qui seront produites par les agents et ainsi de faciliter le calcul de la réaction.

9.2.3 Module *Interaction*, première partie : perceptions des agents

Dans cette section, nous allons spécifier la manière dont la perception $p_a(t) \in P_a$ d'un agent est calculée. Ce qui constitue le premier cas d'interaction dans notre modélisation. Du point de vue du modèle Influence/Réaction, il s'agit là de la première étape de la phase influence ($Perception_a : \Delta \mapsto P_a$).

Description des types de perceptions

Commençons tout d'abord par définir l'ensemble des perceptions P_a qu'un agent peut avoir de son environnement. En fait, cet ensemble se réduit à deux unités : la perception des ressources qui l'entourent et la perception de ses congénères qui sont fertiles et de sexe opposé. Plus formellement, soit $P_a = \{Resource(dx, dy), Neighbor(A_i, dx, dy)\}$ avec :

- $Resource(dx, dy)$: la quantité de ressource disponible sur une cellule non occupée à une distance de (dx, dy) avec dx et dy telles que $0 < |dx| < \Phi_a$ et $0 < |dy| < \Phi_a$ et telles que l'agent ne voie que dans les quatre directions cardinales. Autrement dit, si $dx \neq 0$ alors $dy = 0$ et inversement.
- $Neighbor(dx, dy)$: la perception d'un voisin fertile de sexe opposé[6] où dx et dy sont telles qu'elles définissent le voisinage de Von Neumann de l'agent.

Pour que l'usage de l'aléatoire puisse être déterministe, il nous faut encore une fois créer un ordre total pour chaque type de perception. Ce qui permettra de les classer de façon non ambiguë. Pour créer un tel ordre entre les perceptions de type *Resource*, nous avons adopté la démarche suivante : les ressources sont classées de la plus proche à la plus éloignée en termes de nombre de cellules et dans l'ordre trigonométrique en cas d'égalité sur la distance. Pour ce qui concerne les perceptions de type $Neighbor$, nous utiliserons un ordre croissant basé sur le numéro de l'agent qui correspond à cette perception.

Calcul de la perception d'un agent

La perception d'un agent est une donnée propre à chaque agent et qui doit être calculée par l'environnement. Pour effectuer ce calcul, nous avons utilisé l'algorithme suivant :

[6] Cette perception contient par ailleurs le niveau d'énergie de l'agent concerné. Ce qui permettra à un agent de faire sa décision.

Algorithme *perceptionAgent*
(∗ calcul de la perception d'un agent n ∗)
1. Soit *perceptions* la liste des perceptions.
2. *perceptions* ←∅
3. Etant donnée $L_n(t)$ la position de l'agent A_n, ajouter à *perceptions* les perceptions $Resource(dx, dy)$ dont la valeur est maximale
4. Etant donnés $L_n(t)$ et S_n le sexe de l'agent A_n, ajouter à *perceptions* toutes les perceptions $Neighbor(dx, dy)$ adéquates
5. **return** perceptions

Modélisation MIC* du processus de perception

Objets d'interaction. Outre les processus de calcul et les espaces d'interaction, dans une modélisation effectuée à l'aide de MIC* tout est représenté par des objets d'interaction. Ainsi, les perceptions que nous venons de décrire correspondent très exactement en MIC* à deux nouveaux objets d'interaction : $Resource(dx, dy)$ et $Neighbor(dx, dy)$. On a donc ici un nouvel ensemble d'objets d'interaction $\mathcal{O}_{perceptions}$ et qui est tel que $\mathcal{O}_{perceptions} = \{Resource(dx, dy) \cup Neighbor(dx, dy)\}$. Du point de vue de la notation Influence/Réaction, on a donc $P_a = \mathcal{O}_{perceptions}$.

Première loi d'interaction : $agentPerception_{world}$**.** Pour décrire la dynamique de la perception d'un agent en MIC*, celle-ci sera matérialisée par la réception de l'agent, dans sa inbox, d'un ensemble comprenant des objets d'interaction de types $Resource(dx, dy)$ et $Neighbor(dx, dy)$. Pour cela, nous allons définir une loi d'interaction de l'objet *Landscape* sur les objets *Body* dans l'espace d'interaction $World$ de manière à calculer pour chaque agent la perception qui lui correspond et à la placer dans sa inbox. Soit $agentPerception_{world}$ cette loi d'interaction et a un agent, on a :

$$\begin{aligned} p_a(t) &= in[a, world] \\ &= agentPerception_{world}(out[a, world], out[environment, world]) \\ &= agentPerception_{world}(Body_a, Landscape) \\ &= \cup_{(i,j)} Resource(i, j) \bigcup \cup_{(k,l)} Neighbor(k, l) \end{aligned} \tag{9.1}$$

La figure 9.2 illustre l'évolution du terme MIC* que nous avons présenté précédemment une fois que la loi d'interaction $agentPerception_{world}$ a été appliquée à tous les agents A_i du système. Sur cette figure, on voit que la représentation d'un agent, l'objet d'interaction *Landscape*, interagit sur les objets d'interaction de type *Body* pour finalement produire les perceptions des agents.

On voit sur ce schéma que la perception d'un agent est calculée en fonction de ce dernier. Dans certains cas, aucune ressource ou aucun voisin fertile ne seront perçus par exemple. Par ailleurs, il est clair que, de la même manière que nous allons dans quelques instants classer les influences produites par les agents suivant les différents espaces interaction, nous aurions pu aussi classer les perceptions suivant ces mêmes espaces d'interaction. En mettant les objets d'interaction $Resource(dx, dy)$ dans la inbox de l'agent correspondant à l'espace d'interaction *moves* par exemple. Il s'agit ici purement d'un choix lié à l'implémentation qui sera faite de ce modèle. La majeure partie des calculs qui devront être effectués correspondent à la gestion des influences, non à la gestion des perceptions par les agents. Ceci dit, nous tenions à souligner

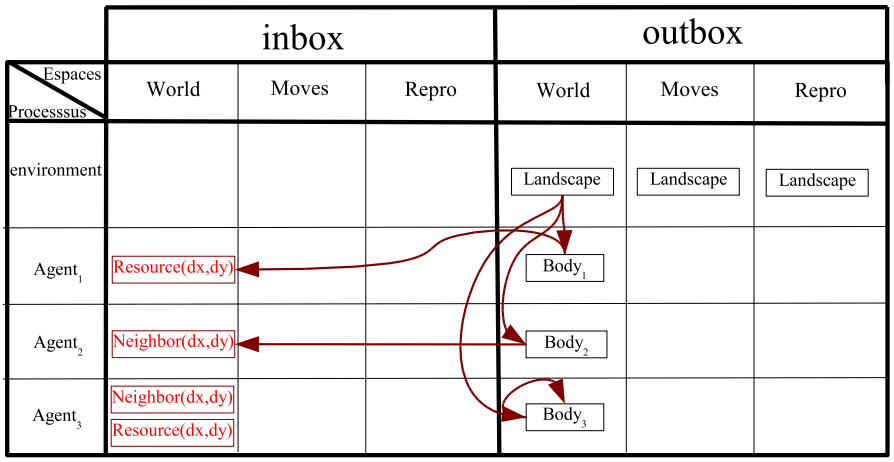

FIG. 9.2 – Evolution du terme MIC* après l'application de la loi d'interaction pour la perception des agents.

qu'il est effectivement tout à fait possible d'utiliser les espaces interaction pour classer les perceptions des agents par type. On aurait par exemple pu créer des espaces d'interaction *neighbors* et *resources* pour effectuer cette classification. Il est clair que cette possibilité sera très appréciable dans des systèmes à forte complexité où les agents peuvent avoir de multiples perceptions différentes de leur environnement.

9.2.4 Module *Comportements*

Nous allons maintenant nous attaquer à la modélisation du système conatif des agents, c'est-à-dire leur comportement proprement dit. Il s'agit ici pour nous de spécifier la manière dont un agent, à partir d'une perception p_a, produit une influence γ_a sur l'environnement. Du point de vue du modèle Influence/Réaction, nous sommes ici dans le deuxième temps de la phase $influence$ $((Memorization_a \circ Decision_a) : P_a \times S_a \mapsto \Gamma')$.

Description des influences produites par les agents

Avant toute chose, il nous faut définir l'ensemble Γ_a des influences que les agents sont capables de produire. Soit $\Gamma_a = \{Move_a(dx, dy), Repro_a(b)\}$, nous définissons les deux influences suivantes :

- $Move_a(dx, dy)$: une influence de mouvement qui correspond à la volonté de l'agent a de se déplacer.
- $Repro_a(b)$: une influence de reproduction qui correspond à la volonté de l'agent a de se reproduire avec l'agent b.

De la même manière que nous avions défini un ordre total sur chaque type de perception, les influences produites par les agents seront ordonnées en fonction du numéro de l'agent qui a produit l'influence.

Pour bien faire, il nous aurait fallu décrire une autre influence : la consommation de la ressource par les agents. Cependant, d'une part nous n'avons pas voulu alourdir la notation et d'autre part, cette influence sera systématiquement présente dans le système et elle ne nécessite ainsi aucun calcul délibératif en particulier. Autrement dit, cette influence sera invariablement prise en compte dans la fonction de transition d'état du système global, c'est-à-dire lors du

calcul de la réaction de l'environnement. Nous considérons ainsi que les agents n'exhibent que deux comportements : le mouvement et la reproduction.

Description du mécanisme de délibération des agents

Le système décisionnel d'un agent sera donc considéré comme la composition de deux fonctions distinctes : $move_a$ et $reproduction_a$. Sachant que nous avons en grande partie mâché le travail de délibération des agents lors de la construction de leurs perceptions, les algorithmes qui correspondent à ces deux fonctions sont extrêmement simples. Par exemple, pour se mouvoir un agent n'a plus qu'à choisir aléatoirement l'une des cellules qu'il perçoit étant donné qu'elles ont déjà été triées par valeurs maximales par le processus de perception. Pour se reproduire, un agent aura un peu plus de travail. Il lui faudra sélectionner le partenaire potentiel qui a le plus d'énergie. Finalement, on obtient respectivement les deux algorithmes suivants :

Algorithme $move_n$
($*$ le comportement de mouvement d'un agent n $*$)
1. Soit *perceptions* la liste des perceptions.
2. Sélectionner puis classer[7] les perceptions $Resource(dx, dy)$ présentes dans *perceptions*
3. Sélectionner aléatoirement une destination.
4. **return** l'influence $Move_a(dx, dy)$ correspondante

Algorithme $reproduction_n$
($*$ le comportement de reproduction d'un agent n $*$)
1. Soit *perceptions* la liste des perceptions.
2. Sélectionner puis classer les perceptions $Neighbor(dx, dy)$ présentes dans *perceptions*
3. Soit $b \leftarrow$le partenaire potentiel $Neighbor(dx, dy)$ possédant l'énergie la plus grande. Choisir aléatoirement en cas d'égalité.
4. **return** l'influence $Repro_n(b)$ correspondante

Par ailleurs, étant donnés ces deux algorithmes, il est important de remarquer que l'influence produite par un agent peut être nulle. Ce qui arrivera en fait assez fréquemment pour le comportement de mouvement lorsque la population sera dense par exemple.

Modélisation MIC* des comportements

Objets d'interaction. C'est maintenant devenu un classique, les influences que nous avons décrites correspondent exactement à deux nouveaux objets d'interaction. De plus, pour signifier la production d'une influence nulle, on utilise l'objet d'interaction zéro $IO_\emptyset$. On a donc ici un nouvel ensemble d'objets d'interaction $\mathcal{O}_{influences}$ tel que $\mathcal{O}_{influences} = \{Move_n(dx, dy) \cup Repro_a(b) \cup IO_\emptyset\}$. Du point de vue de la notation Influence/Réaction, on a donc naturellement $\Gamma_a = \mathcal{O}_{influences}$.

Premières lois de computation : $move_a$ et $reproduction_a$. Pour décrire la dynamique comportementale des agents en MIC*, il nous faut maintenant définir les lois de computation qui lui correspondent. En fait, on définit ici tout simplement deux lois de computation pour chaque agent, $move_a$ et $reproduction_a$. L'application de l'une de ces lois produira un nouvel objet d'interaction, l'influence, qui sera placé dans la outbox de l'agent sur l'espace

[7] Préalablement à l'utilisation de l'aléatoire, les perceptions doivent impérativement avoir été classées suivant l'ordre que nous avons défini sur chaque type de perception. Sans quoi la réplication exacte de notre modèle sera dépendante de l'implémentation.

d'interaction correspondant au type de l'influence générée. De plus, cet objet d'interaction remplacera le précédent (que l'influence soit nulle ou non). La figure 9.3 illustre un exemple d'application de ces lois sur le terme MIC* de la figure 9.2. On remarquera au passage que ces lois de computation ont aussi pour conséquence la réinitialisation de la inbox d'un agent[8].

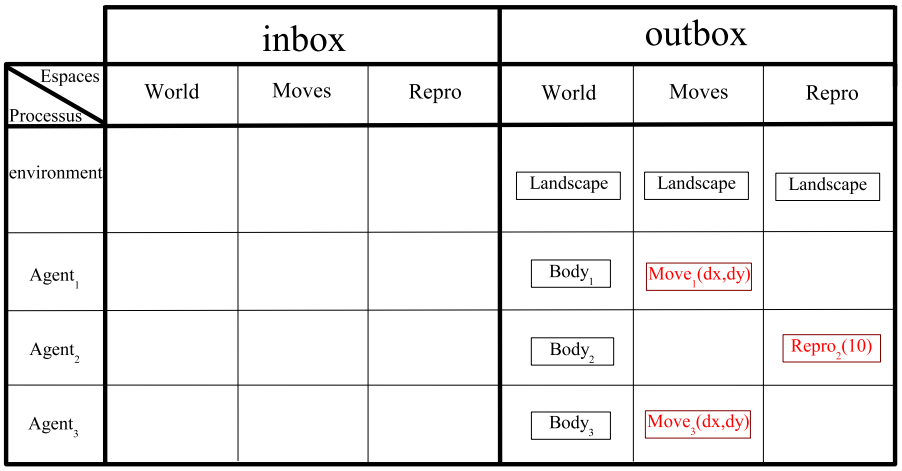

FIG. 9.3 – Evolution du terme MIC* après l'application des lois de computation correspondant à la production d'influences par les agents.

9.2.5 Module *Interaction*, deuxième partie : calcul de la réaction

Nous arrivons maintenant à la partie la plus délicate de la modélisation : le calcul de la réaction à un instant t. Rappelons ici qu'il s'agit de calculer le nouvel état $\delta(t+dt) \in \Delta$ en appliquant une fonction $Reaction : \Sigma \times \Gamma' \mapsto \Delta$ telle que $\delta(t+dt) = Reaction(\gamma'(t), \delta(t))$. Ce calcul étant effectué par l'environnement, du fait de la modélisation en MIC*, celui-ci sera décomposé en deux phases : la récupération des informations nécessaires (influences et variables d'état du système) puis le calcul de la réaction proprement dit.

Modélisation MIC* : récupération des influences par l'environnement

Pour que le processus de calcul représentant l'environnement récupère l'ensemble des informations nécessaires au calcul de la réaction, nous allons définir trois lois d'interaction, $envPerception_{moves}$, $envPerception_{repro}$ et $envPerception_{world}$. Les deux premières sont définies sur les deux espaces d'interaction où se trouvent les objets d'interaction qui correspondent aux influences tandis que la troisième permettra de récupérer l'état courant du système. Ces différentes lois d'interaction permettront au processus de calcul représentant l'environnement de recevoir ces informations dans sa inbox. A titre d'exemple, considérons l'application de la loi $envPerception_{moves}$ restreinte à un agent a. Le résultat obtenu sera tout simplement l'influence de mouvement produite par cet agent. Ce qui donne par exemple le résultat suivant lorsque cet agent a produit une influence de mouvement non nulle :

[8] Cette modélisation nous permet aussi de faire une première remarque en ce qui concerne la nature des agents que nous considérons : ils n'ont aucune mémoire. En effet, aucune variable n'est nécessaire pour définir leur comportement. Ainsi, l'application de la distinction esprit/corps et sa modélisation en MIC* nous ont permis de mettre en évidence le fait que le système que nous sommes en train de modéliser contient exclusivement des agents tropiques.

$$\begin{aligned} \gamma_a &= in[landscape, moves] \\ &= envPerception_{moves}(out[landscape, moves], out[a, moves]) \\ &= envPerception_{moves}(Landscape, Move_a(dx, dy)) \\ &= Move_a(dx, dy) \end{aligned} \tag{9.2}$$

Par ailleurs, la loi $envPerception_{moves}$ est définie de telle sorte qu'elle a un effet de bord supplémentaire : elle réinitialise la outbox de l'agent sur l'espace d'interaction *moves*. Du point de vue de la sémantique du modèle MIC*, on dit que l'objet d'interaction a été *consommé* par l'interaction. Du point de vue de la dynamique du modèle, cela veut tout simplement dire qu'une influence de mouvement sera systématiquement traitée. Les deux autres lois d'interaction n'ont pas cet effet sur la outbox des agents. Comme nous allons le voir, une influence de reproduction n'a pas forcément de répercussions et elle ne doit donc pas être effacée du système. La loi d'interaction $envPerception_{world}$ représente quant à elle uniquement le moyen de récupérer les informations sur l'état courant du système. Il est donc naturel que la représentation des agents (les objets d'interaction de type *Body*) ne disparaissent pas à la suite de son application.

Afin de ne pas alourdir la notation, il est ici intéressant de définir une loi d'interaction $envPerception_{\mathcal{J}}$ qui encapsule les trois lois précédentes. Ceci est possible car elles seront en fait déclenchées systématiquement en même temps. Rappelons ici qu'il s'agit uniquement de récupérer les informations nécessaires au calcul de la réaction. L'application de ces lois d'interaction n'a donc pas d'implication sur le résultat de la dynamique du système.

La figure 9.4 illustre l'évolution du terme MIC* précédent une fois que la loi d'interaction $envPerception_{\mathcal{J}}$ a été appliquée.

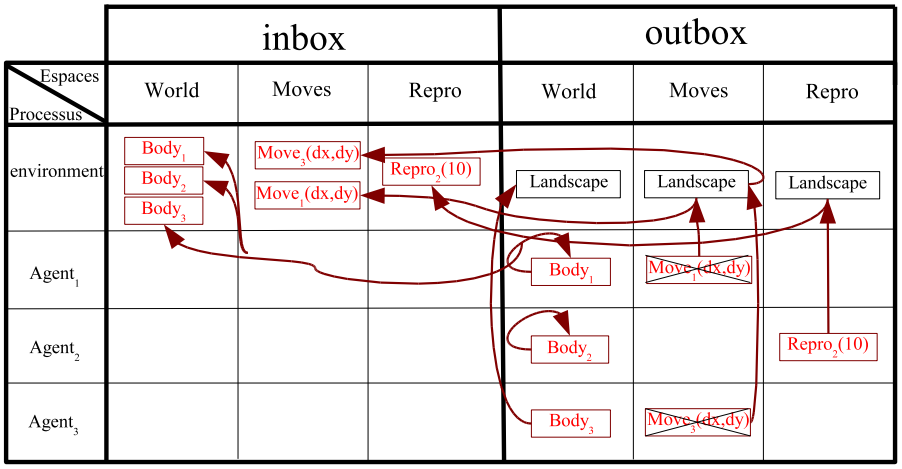

FIG. 9.4 – Evolution du terme MIC* après l'application de $envPerception_{\mathcal{J}}$.

Une fois cette évolution effectuée, l'ensemble des informations nécessaires au calcul de la réaction se trouvent dans la inbox du processus de calcul *environment*. On peut maintenant calculer la transition d'état du système en fonction des différentes influences, de l'état courant du système et de son évolution endogène. Nous allons maintenant présenter les différentes étapes de la réaction dans l'ordre où elles sont effectivement traitées dans notre modèle[9].

[9] Il peut en effet bien sûr exister une différence suivant l'ordre dans lequel les différentes étapes de la réaction sont considérées.

Gestion des influences de mouvement

La gestion des influences de mouvement consiste à prendre en compte les différentes influences $\gamma'_{moves}(t)$ et à modifier la position des agents en conséquence. Etant donné qu'il est possible d'avoir plusieurs agents qui souhaitent se déplacer sur une même cellule, le traitement des influences de mouvement consiste à considérer les influences dans un ordre aléatoire et à résoudre les conflits potentiels selon la règle suivante[10]. Dans le cas où un agent souhaite se déplacer sur une cellule qui se retrouve occupée, l'environnement calcule alors la cellule non occupée la plus proche de l'objectif initial et y place l'agent. Dans le cas où cette cellule n'existerait pas, le mouvement de l'agent est annulé. Une fois l'ensemble des influences de mouvement ainsi traité, on aura $\gamma_{moves}(t + dt) = \emptyset$. Soit l'algorithme suivant :

Algorithme *movesReaction*
(∗ gestion des influences de mouvement ∗)
1. Soit $moveInfluences$ la liste classée des influences de mouvement
2. **repeat**
3. Sélectionner aléatoirement une influence $Move_n(dx, dy)$ de la liste $moveInfluences$
4. Calculer la cellule de destination adéquate et mettre à jour la variable L_n de l'agent A_n
5. Retirer cette influence de la liste $moveInfluences$
6. **until** $moveInfluences$ n'est pas vide

Modélisation MIC* de la gestion des mouvements

La réaction aux influences de mouvement de l'environnement que nous venons de décrire correspond en MIC* à une loi de computation *environmentMovesReaction* qui consiste simplement à faire effectuer l'algorithme *movesReaction* par le processus de calcul *environment* sur les différents objets d'interaction concernés. Notamment les objets qui représentent les variables d'état de l'environnement (*Body* et *Landscape*).

Gestion des influences de reproduction

La gestion des influences de reproduction est tout à fait différente. Il s'agit en effet de traiter une interaction forte. Autrement dit, il est nécessaire de traiter simultanément l'ensemble des influences de reproduction de manière à respecter le principe d'autonomie, et donc le principe de cohérence paradigmatique. Pour cela, nous allons maintenant définir une nouvelle influence $ReproSuccess(a, b)$ qui correspond à la combinaison réussie de deux influences de reproduction telle que $ReproSuccess(a, b) = Repro_a(b) + Repro_b(a)$. En d'autres termes, cette influence modélise la naissance potentielle d'un nouvel agent. Grâce à cette modélisation, chaque agent garde la maîtrise de son processus décisionnel. Par ailleurs, pour définir un ordre total entre les influences de ce nouveau type, on considère l'agent du couple qui possède le plus petit numéro.

Pour appliquer ce principe de modélisation dans le calcul de la réaction, on va appliquer une règle de réduction à l'ensemble des influences de reproduction γ'_{repro} telle que $Repro_a(b) + Repro_b(a) \mapsto ReproSuccess(a, b)$. Par exemple, si l'ensemble des influences de reproduction est $\gamma'_{repro} = \{Repro_a(b), Repro_c(b), Repro_b(a), Repro_d(c)\}$, la règle de réduction donne le nouvel ensemble $\gamma'_{repro} = \{ReproSuccess(a, b), Repro_c(b), Repro_d(c)\}$.

[10] Deux agents ne pouvant pas se trouver sur la même cellule dans ce modèle.

Une fois ce nouvel ensemble calculé, l'environnement calcule les naissances qui doivent avoir lieu étant donné l'ensemble des objets d'interaction de type *ReproSuccess*[11]. Cet ensemble est traité dans un ordre aléatoire étant donné qu'une naissance peut entraîner l'occupation d'une cellule et potentiellement l'impossibilité d'une autre naissance.

Lorsqu'une naissance réussie (ou non) intervient, l'objet d'interaction *ReproSuccess* correspondant est éliminé et l'énergie des parents est mise à jour de telle sorte que chaque géniteur perd la moitié de son énergie initiale $w_a(\beta)$ au profit du nouveau-né. Les caractéristiques de ce dernier sont héritées de l'un des deux parents suivant un tirage aléatoire, sauf pour le sexe et par conséquent pour la caractéristique de l'âge de fin de reproduction de l'agent. Par ailleurs, le nouvel agent sera placé sur la cellule où la quantité de ressource est maximale et qui se trouve dans le voisinage de Von Neumann d'un des deux agents. Nous détaillerons cette procédure un peu plus loin.

Finalement, pour résumer l'ensemble de cette gestion, on a l'algorithme suivant :

Algorithme *reproReaction*
(∗ gestion des influences de reproduction ∗)
1. Soit $reproInfluences$ la liste des influences de reproduction et de naissances
2. Appliquer la règle de réduction sur la liste $reproInfluences$
3. Soit *births* la liste classée des influences de type *ReproSuccess* dans $reproInfluences$ et qui doivent intervenir à l'instant t
4. **repeat**
5. Sélectionner aléatoirement une influence $ReproSuccess(a, b)$ dans la liste *births*
6. Effectuer les calculs correspondants : mise à jour de A_a, A_b et nouvel agent si possible
7. Retirer cette influence de la liste *births*
8. **until** *births* n'est pas vide

Modélisation MIC* de la gestion des influences de reproduction

De la même manière que pour la réaction précédente, la gestion des influences de reproduction correspond en MIC* à une loi de computation *environmentReproReaction* qui consiste dans l'application de l'algorithme *reproReaction*. Il s'agit encore une fois de manipuler les objets d'interaction concernés.

La modélisation de la règle de réduction des influences de reproduction est quant à elle très intuitive en MIC*. En fait, elle peut être appliquée telle qu'elle a été définie car il est effectivement possible de donner une sémantique à l'opérateur + entre les objets d'interaction. Autrement dit, la règle de réduction est directement modélisée à l'intérieur des objets d'interaction une fois l'opération + redéfinie de la manière suivante :

$$\begin{aligned} Repro_a(b) + Repro_c(d) &= ReproSuccess(a, b) \ si \ a = d \ et \ b = c \\ &= Repro_a(b) + Repro_c(d) \quad sinon \end{aligned} \tag{9.3}$$

Comme on peut le voir dans l'équation précédente, cette opération définit naturellement le **nouvel objet d'interaction** *ReproSuccess*. Cette création d'un objet supplémentaire concrétise le fait que nous modélisons la reproduction comme une interaction forte. Interaction qui ne se réduit donc pas à la somme de ses parties.

[11] Ces objets contiennent l'information de la date de la naissance

Evolution endogène de l'environnement

Nous allons maintenant finir le calcul de la réaction en y incluant l'évolution endogène de l'environnement. Cette évolution concerne d'abord la condition physique des agents : ils récoltent la ressource, brûlent leur énergie et vieillissent. En deuxième lieu vient la régénération de la ressource présente dans une cellule.

Pour calculer la mise à jour de l'environnement correspondant à cette évolution, nous allons tout d'abord régénérer l'ensemble des cellules et à mettre à jour l'énergie des agents de telle sorte que :

$$R_{(x,y)}(t+dt) = \min(C_{(x,y)}, R_{(x,y)}(t) + dt * \rho_{(x,y)}) \text{si la est cellule libre, 0 sinon} \tag{9.4}$$

$$w_a(t+dt) = w_a(t) + dt * (\rho_{(L_a(t+dt))} - \mu_a) + R_{L_a(t+dt)}(t)) \tag{9.5}$$

Le détail de l'équation 9.4 montre que nous avons considéré que la quantité de ressource régénérée sur une cellule est immédiatement récupérée par l'agent qui l'occupe. Par exemple, si un agent tel que $w_a(1) = 5$ se retrouve après un mouvement sur une cellule telle que $R_{(x,y)}(1) = 2$ et $\rho(x,y) = 1$, on on obtient par exemple $w(2) = 1 + 2 - \mu_a$. Nous avons ainsi considéré que la régénération de la ressource est concomitante avec la consommation qui en est faite.

Une fois ces équations appliquées[12], tous les agents dont l'énergie est inférieure ou égale à 0 sont éliminés. Vient ensuite l'élimination des agents morts de mort naturelle. Autrement dit, avant de valider définitivement le nouvel état du système, l'environnement élimine tous les agents qui ont un âge tel que $t + dt \leq \beta_a + \lambda_a$. Une fois ces calculs effectués, la transition d'état peut avoir lieu et le système se trouve maintenant défini pour l'instant $t + dt$ et un nouveau cycle Influence/Réaction peut commencer.

Modélisation MIC* de l'évolution endogène de l'environnement

Du point de vue de la modélisation MIC*, l'évolution endogène de l'environnement consiste encore une fois à manipuler les objets d'interaction concernés de manière à mettre à jour leurs variables d'état. Il s'agit ici d'une troisième loi de computation que l'on peut appeler *environmentEndogenousEvolution* et qui aura pour conséquence la mise à jour des représentations de l'environnement dans la outbox du processus de calcul *environment*.

Par ailleurs, les trois étapes que nous venons de détailler peuvent être encapsulées dans une seule loi de computation *environmentReaction* qui correspond à l'application de ces différentes étapes dans l'ordre où nous les avons définies.

La figure 9.5 illustre un exemple du résultat qui pourrait être obtenu par l'application de la loi de computation *environmentReaction* sur le terme MIC* de la figure 9.4.

9.2.6 Module *Ordonnancement* : modélisation de l'évolution temporelle

Nous pouvons à présent définir un modèle d'évolution temporelle en définissant la manière dont le comportement des agents et la réaction seront activés. Pour cela, nous allons définir

[12] L'ordre d'application de ces deux équations n'a ici aucune importance.

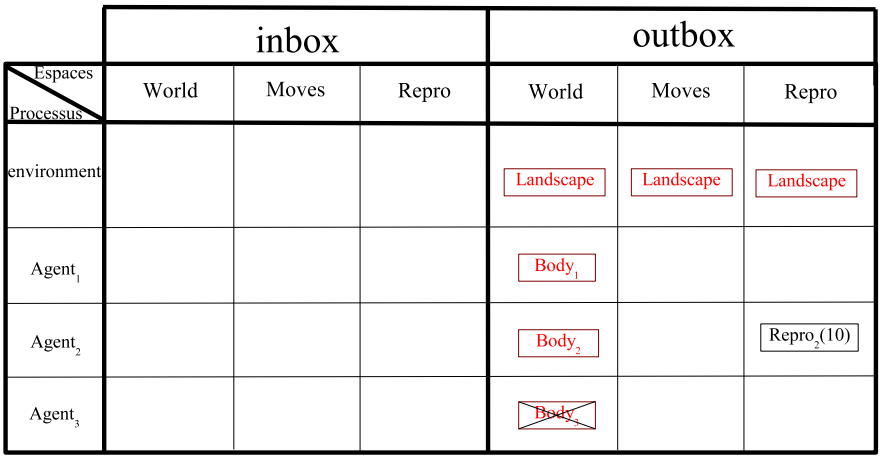

FIG. 9.5 – Evolution du terme MIC* après l'application de la loi de computation correspondant à la réaction de l'environnement.

deux modes d'évolution du temps : le premier sera basé sur une discrétisation régulière du temps (à pas de temps constant) et le deuxième consistera dans un principe de simulation par événement. Dans les deux cas, il s'agit de définir de quelle manière les différentes lois d'interaction et de computation que nous avons définies seront activées.

Simulation à pas de temps constant

Dans ce premier modèle de l'évolution temporelle, le temps sera discrétisé en intervalles réguliers et tous les agents seront activés pour un même instant t de la simulation, c'est-à-dire de manière synchrone. Suivant ce principe, de nombreux modes de fonctionnement sont possibles suivant les comportements qui sont activés et les instants auxquels la réaction est déclenchée.

Par exemple, dans la modélisation de L-P, le mouvement de chaque agent est immédiatement validé et il n'existe ainsi aucun conflit. C'est pourquoi, pour ne pas trop nous éloigner de cette version, notre premier choix de modélisation a été de considérer que lorsqu'un agent effectue un mouvement, la réaction de l'environnement est immédiate (sans évolution de la variable temporelle). On n'aura ainsi aucun conflit.

Etant donnée cette gestion des mouvements, que l'on peut qualifier de classique, il sera important de traiter le mouvement des agents dans un ordre aléatoire. Dans un deuxième temps, le comportement de reproduction de tous les agents sera activé puis viendra le temps de la réaction à ces influences. Dans la gestion de ce deuxième comportement, il ne sera pas nécessaire de considérer les agents aléatoirement sachant qu'il s'agit d'une application stricte du principe Influence/Réaction et que l'ordre dans lequel les agents seront activés n'a aucune importance.

De plus, dans L-P les agents effectuent ces deux comportements dans une seule unité de temps, c'est pourquoi nous avons discrétisé le temps en intervalles d'une demi unité temporelle afin d'avoir une échelle temporelle équivalente. On aura donc toujours $dt = 0.5$.

L'algorithme résume la manière dont un pas de temps sera réalisé étant donné ce premier modèle d'exécution :

Algorithme *Discret : modèle d'exécution en mode discret*
(∗ application directe des mouvements ∗)
1. $dt \leftarrow 0.5$
2. Soit *agentsList* la liste classée de tous les agents.
3. **for** tous les agents A_i de *agentsList* pris dans un ordre aléatoire
4. **do** activer la loi d'interaction $agentPerception_{world}$ de A_i
5. activer la loi de computation $move_{A_i}$ de A_i
6. activer la loi d'interaction $envPerception_{\mathcal{J}}$ de l'environnement
7. activer la loi de computation *environmentMovesReaction*
8. activer la loi de computation *environmentReaction*
9. $T \leftarrow$T + dt
10. **for** tous les agents A_i de *agentsList*
11. **do** activer la loi d'interaction $agentPerception_{world}$ de A_i
12. activer la loi de computation $reproduction_{A_i}$ de A_i
13. activer la loi d'interaction $envPerception_{\mathcal{J}}$ de l'environnement
14. activer la loi de computation *environmentReaction*
15. $T \leftarrow$T + dt

Simulation par événements

Dans ce mode de fonctionnement, les agents seront cette fois activés étant donné un rythme qui leur sera propre. Il s'agit ici du mode de fonctionnement que l'on appelle asynchrone. Au contraire du principe de simulation précédent, un seul agent sera activé pour un instant t de la simulation (hormis dans le cas où deux événements ont la même date). Pour cela, lorsqu'un agent sera activé, l'échéancier calculera la date de sa prochaine activation en considérant que le comportement d'un agent est un processus de renouvellement de type poissonnien. Pour garder la même échelle temporelle que dans le mode de simulation synchrone, c'est-à-dire pour faire en sorte que les agents agissent deux fois par unité de temps en moyenne (un mouvement et une tentative de reproduction de manière alternée), le comportement d'un agent sera un processus de Poisson $P(\lambda = 2)$[13].

En plus des événements correspondants à l'activation du comportement des agents, deux événements supplémentaires seront intégrés à la liste des événements suivant les besoins : la mort naturelle d'un agent et la naissance d'un nouvel agent. Par ailleurs, lorsque deux agents sont en train de se reproduire, les événements qui correspondaient à leurs futures actions sont enlevés de la liste des événements. Lorsqu'une naissance a lieu, les dates d'exécution des futurs comportements (un mouvement) du nouvel agent et de ses parents sont réactualisées.

A partir de là, l'algorithme de simulation par événements est des plus classiques. Il consiste simplement à traiter les événements dans l'ordre où ils doivent intervenir et à intégrer les nouveaux événements générés par les agents (futurs comportements) et la réaction de l'environnement (naissances). Il comporte cependant une petite difficulté. Contrairement au mode précédent où l'évolution du système est calculée étant donnée un dt invariant, la mise à jour de l'environnement ne peut être effectuée avant de connaître la date d'activation du prochain agent, cela afin d'avoir la donnée dt. Par exemple, si le premier agent agit à la date 0.36, il est d'abord nécessaire de calculer l'évolution qui a eu lieu entre le temps $t = 0$ et cette nouvelle date. Autrement dit, soit $event(date)$ un événement dont la date de réalisation est $date$:

[13] Pour simuler un processus poissonnien de paramètre λ, on sait que le temps entre deux événements Δt suit une loi exponentielle de moyenne $T = 1/\lambda$ telle que $\Delta t = -T * \log(U)$ avec U de loi uniforme sur $[0, 1]$.

Algorithme *Event : mode événementiel*
(∗ simulation par événements ∗)
1. **while** la liste des événements n'est pas vide
2. Retirer le prochain événement $event(date)$ à exécuter de la liste des événements
3. $dt \leftarrow date - T$
4. activer la loi de computation $environmentEndogenousEvolution$
5. $T \leftarrow$date
6. $dt \leftarrow 0$
7. **if** event(date) est l'activation d'un comportement $behaviour$ d'un agent A_i
8. **then** activer la loi d'interaction $agentPerception_{world}$ de A_i
9. activer la loi de computation $behaviour_{A_i}$
10. activer la loi d'interaction $envPerception_{\mathcal{J}}$
11. activer la loi de computation $environmentReaction$
12. suivant les besoins : ajouter ou retirer les événements de la liste des événements

Au contraire du mode de simulation précédent et étant donné le modèle de transition d'état que nous avons présenté dans la section 9.2.5, les variations qu'il est possible de faire sur l'ordre dans lequel la réaction de l'environnement traite les différentes influences n'aura quasiment aucun impact sur une transition d'état lorsque la variable qui représente le temps est d'une précision suffisante. En fait, lorsque la variable temporelle est représentée par une bonne approximation d'un réel, il n'y a pour ainsi dire jamais deux événements ayant la même date de réalisation (du fait de la modélisation suivant un processus poissonnien).

9.2.7 Génération de l'état initial et gestion des nouveaux agents créés par reproduction

Nous avons à présent défini l'ensemble des spécifications qui correspondent à la dynamique du modèle. Pour que l'implémentation que nous allons effectuer puisse être répliquée, il nous reste cependant à préciser certains paramètres du modèle. Il nous faut notamment définir en quoi consiste l'état initial du système et la manière dont celui-ci est généré. Par ailleurs, nous allons aussi préciser la façon dont un nouvel agent créé par reproduction est initialisé.

Commençons tout d'abord par préciser le processus utilisé pour initialiser la grille de cellules. Pour toutes tailles (X, Y) de l'environnement, la capacité d'une cellule sera calculée en utilisant l'équation proposée par L-P :

$$\begin{aligned} C_{(x,y)} &= f(x - X/4, y - Y/4) + f(x - 3X/4, y - 3Y/4) \\ avec \quad f|f(a,b) &= \Psi \exp(-(a/0.3X)^2 - (b/0.3Y)^2) \end{aligned} \tag{9.6}$$

Dans toutes nos expériences, nous prendrons pour Ψ la même valeur que celle qui a été utilisée par L-P, à savoir $\Psi = 4$[14]. Une fois la grille initialisée suivant cette formule, on obtient un environnement où la ressource est répartie de manière non uniforme telle qu'illustrée par la figure 9.6 pour un environnement où $X = Y = 50$. On voit sur cette figure que la formule précédente définit deux zones très attractives (en bas à gauche et en haut à droite) où la capacité des cellules est élevée et autour desquelles la capacité décroît régulièrement.

[14]Cette valeur correspondra en fait à $C_{(x,y)}$ pour les cellules possédant la capacité la plus forte.

FIG. 9.6 – Répartition initiale de la ressource (ici $X = Y = 50$).

Passons maintenant à l'initialisation des agents. Soit N le nombre d'agent initial pour l'instant 0, les agents initialement présents dans le système sont initialisés à l'aide de l'algorithme suivant :

Algorithme *Initialisation d'une population initiale de n agents*
1. **for** $i \leftarrow 1$ **to** n
2. **do** générer S_i aléatoirement
3. générer Φ_i aléatoirement
4. générer μ_i aléatoirement
5. générer λ_i aléatoirement
6. générer α_i aléatoirement
7. générer ω_i aléatoirement étant donné S_i
8. $w_i \leftarrow 0$
9. $\beta_i \leftarrow 0$
10. **repeat**
11. générer un entier x aléatoirement tel que $x \in [0, X-1]$
12. générer un entier y aléatoirement tel que $y \in [0, Y-1]$
13. **until** la cellule de coordonnées (x, y) n'est pas occupée
14. $L_i(0) \leftarrow (x, y)$
15. **if** (simulation par événements)
16. **then** générer la date de la première activation du comportement de A_n (toujours un mouvement)

A chaque fois qu'un nouvel agent sera créé par reproduction à un instant t, il sera initialisé en héritant des caractéristiques (sauf le sexe) de l'un de ses parents, A_a et A_b, à l'aide de l'algorithme suivant et sachant que $a < b$:

Algorithme *Initialisation d'un nouvel agent de numéro n à l'instant t*
1. Soit $b1$ une valeur booléenne tirée aléatoirement
2. $S_n \leftarrow b1$
3. **if** $S_n = vraie$
4. **then if** $S_a = vraie$

5. **then** $\omega_n \leftarrow \omega_a$
6. **else** $\omega_n \leftarrow \omega_b$
7. **else if** $S_a = vraie$
8. **then** $\omega_n \leftarrow \omega_b$
9. **else** $\omega_n \leftarrow \omega_a$
10. Soit $b2$ une valeur booléenne tirée aléatoirement
11. **if** $b2 = vraie$
12. **then** $\mu_n \leftarrow \mu_a$
13. $\lambda_n \leftarrow \lambda_a$
14. $\alpha_n \leftarrow \alpha_a$
15. $\Phi_n \leftarrow \Phi_a$
16. **else** $\mu_n \leftarrow \mu_b$
17. $\lambda_n \leftarrow \lambda_b$
18. $\alpha_n \leftarrow \alpha_b$
19. $\Phi_n \leftarrow \Phi_b$
20. $\beta_n \leftarrow$t
21. $L_n(\beta_n) \leftarrow (x, y)$ tel que (x, y) est la cellule de ressource maximale prise dans le voisinage de Von Neumann de l'un des deux parents. Choisir aléatoirement en cas d'égalité[15]
22. **if** (simulation par événements)
23. **then** générer la date de la première activation du comportement de A_n
24. générer la date de l'activation du comportement de A_a
25. générer la date de l'activation du comportement de A_b

Nous en avons à présent fini avec l'étape de modélisation. Nous allons maintenant présenter l'implémentation du simulateur qui a été élaboré pour simuler ce modèle.

9.3 Implémentation du simulateur

Pour implémenter le simulateur correspondant au modèle que nous venons de présenter, nous avons utilisé plusieurs librairies logicielles. En premier lieu, nous avons utilisé les outils de conception que nous avons présentés dans le chapitre 5. Le simulateur a donc été réalisé dans le cadre de la plate-forme MadKit à l'aide du langage Java. Par ailleurs, nous avons aussi utilisé le prototype écrit en Java du modèle MIC* (cf. section 8.3.3 page 177). Ce qui a permis une intégration relativement simple de celui-ci dans le contexte des outils proposés par MadKit. Le simulateur a donc été réalisé en deux temps, l'implémentation des structures MIC* présentes dans la modélisation puis l'intégration de celles-ci dans un principe de simulation basé sur le pattern organisationnel que nous avons proposé dans la section 5.4 (page 107).

9.3.1 Implémentation des structures MIC*

Du point de vue de l'implémentation, l'avantage de la modélisation MIC* que nous avons effectuée repose précisément sur la correspondance qui existe entre la description des fonctions (computation, interaction, etc.), des objets d'interaction ($\mathcal{O}_{influences}$, $\mathcal{O}_{perceptions}$ et $\mathcal{O}_{physical}$) et des processus de calcul utilisés avec des structures informatiques concrètes (des instances

[15] Pour faire ce choix aléatoire, il est encore une fois nécessaire d'établir un ordre total entre les différentes cellules cibles dans le cas où il existe une égalité sur les quantités de ressource. Pour cela, le classement effectué consiste à classer les cellules suivant le sens trigonométrique en considérant en priorité les cellules du voisinage de A_a puis les cellules du voisinage de A_b.

ou des sous-classes du noyau MIC*). En effet, le noyau MIC* a été implémenté de manière à fournir une librairie logicielle générique de l'ensemble des concepts qui sont proposés par le modèle théorique. On a ainsi une équivalence directe entre le formalisme de la modélisation et l'implémentation. Un peu à la manière de ce qui existe pour le formalisme DEVS où les simulateurs correspondants implémentent directement les concepts manipulés dans le modèle. Par exemple, l'écriture utilisée dans l'équation 9.1 a été directement transcrite en utilisant les appels correspondants sur le noyau MIC*. Le code de cette équation (l'application d'une loi d'interaction) se trouve par exemple dans une classe appelée `InteractionLawForPerception` et a la forme suivante :

```
public void computeAgentPerceptions(AgentPlugin a)
{
        MICkernel.getScheduler().applyInteractionLaw(new InteractionLaw("world",a));
}
```

Ce qui correspond pour le noyau à faire interagir la représentation de l'agent a ($Body_a$) dans l'espace d'interaction *world* avec les autres objets d'interaction qui se trouvent dans ce même espace. Ce qui va notamment déclencher l'appel de méthode $Body_a.interaction(Landscape)$ dont le résultat sera placé dans la inbox de l'agent concerné.

L'interaction elle-même est encodée à l'intérieur de l'objet $Body_a$. Pour cet objet, nous avons par exemple implémenté une classe appelée `AgentPhysicalBody` dans lequel se trouve le code de l'interaction avec l'environnement :

```
public InteractionObject interaction(InteractionObject io)
{
        InteractionObject result = null;
        if(io instanceof Landscape)
        {
                ...
                //traitement de l'interaction avec l'environnement
                //Landscape : seul objet avec lequel l'agent interagit
                ...
        }
        return result;
}
```

La figure 9.7 résume les différentes instances ou sous-classes que nous avons créées pour l'implémentation des structures MIC* utilisées dans le simulateur. Dans l'annexe B, nous donnons une version simplifiée de quelques-unes des classes qui ont été utilisées pour réaliser le simulateur. Les objets d'interaction (section B.1 page 231), les processus de calcul (section B.2 page 240) et une partie du moteur de simulation (section B.3 page 245).

9.3.2 Intégration des structures MIC* dans un simulateur basé sur MADKIT

Une fois les structures MIC* implémentées, on dispose d'une transcription fidèle du modèle. Cependant ces structures ne définissent pas les mécanismes qui permettent de les mettre en œuvre et de les exploiter. C'est précisément à ce stade de l'implémentation qu'interviennent les outils de conception que nous avons proposés car ils ne font aucune supposition sur les modèles qu'ils permettent de mettre en œuvre. Nous avons pu ainsi intégrer très naturellement les structures MIC* dans une architecture de simulation basée sur le pattern organisationnel que nous avons proposé dans la section 5.4 (page 107) comme l'illustre la figure 9.8.

Par exemple, les processus de calcul ont été convertis en agents MADKIT, ce qui nous a permis d'ordonnancer les lois de computation en utilisant le principe des activateurs grâce à la structure organisationnelle sous-jacente. De la même manière, les lois d'interaction ont été

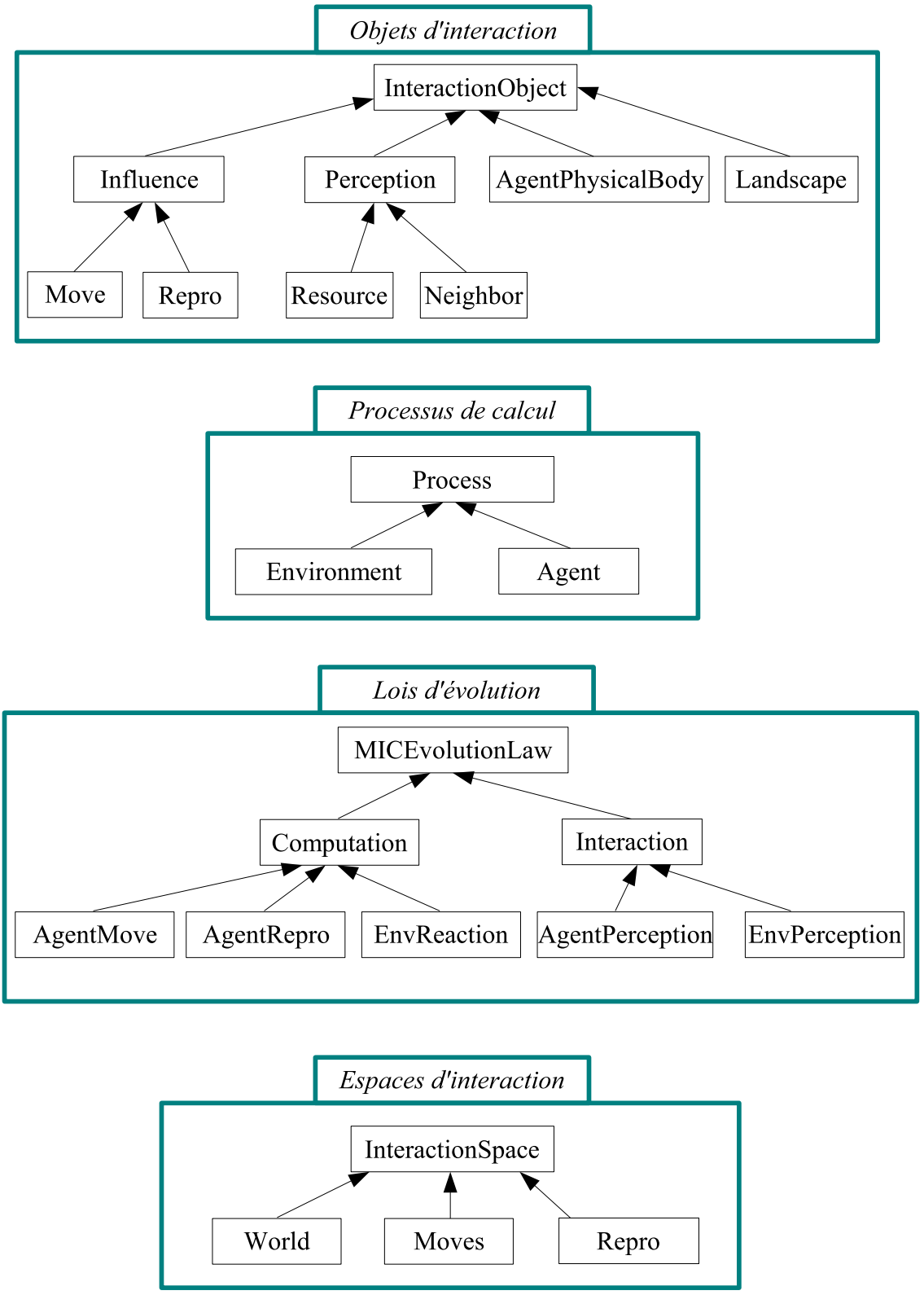

FIG. 9.7 – Structures MIC* utilisées dans le simulateur.

encapsulées dans des activateurs qui réalisent les appels correspondants sur le noyau MIC*. A partir de là, il nous a suffi de définir un ordre d'activation entre activateurs pour définir les différentes dynamiques souhaitées.

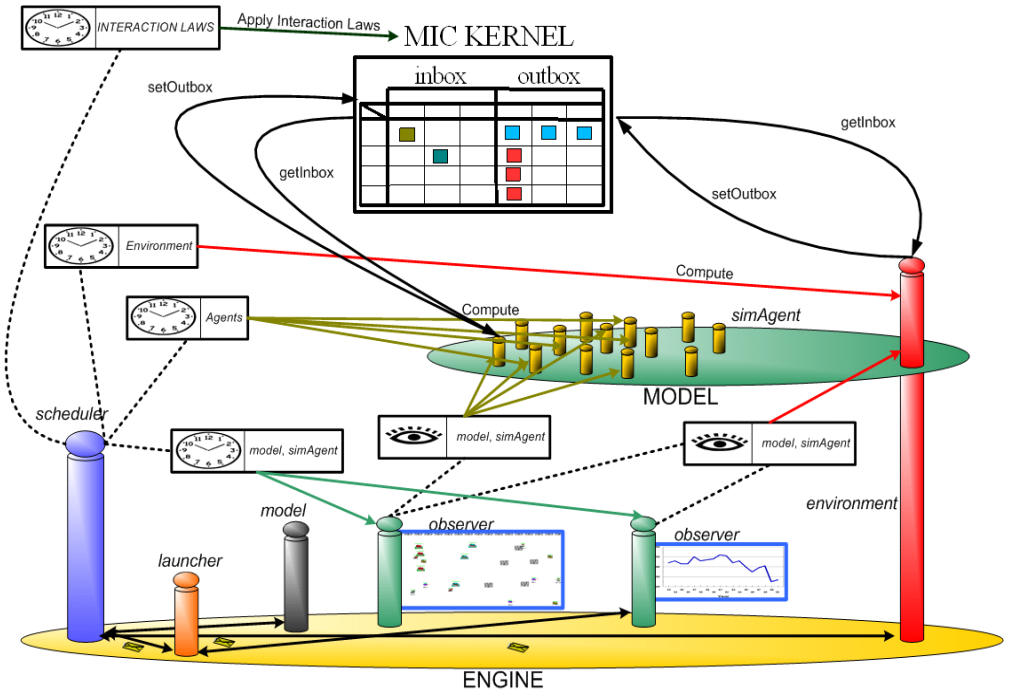

FIG. 9.8 – Utilisation des structures MIC* dans un simulateur basé sur MADKIT.

9.3.3 Interfaces graphiques et récupération des résultats

Pour contrôler l'exécution du simulateur, nous avons implémenté l'interface graphique de l'agent ayant le rôle de *launcher* de manière à pouvoir manipuler les différents paramètres du modèle (nombre d'agents initial, temps de simulation, etc.). En ce qui concerne la visualisation de l'état du système, celle-ci a été réalisée grâce à un agent ayant le rôle d'*observer* dans l'organisation. Le principe est de placer une sonde sur l'agent *environnement* de façon à récupérer les informations nécessaires à l'affichage. Toujours sur ce même principe, un autre agent ayant le même rôle est chargé de récupérer les informations qui concernent la population et de les stocker dans un fichier. La figure 9.9 montre une copie d'écran des agents ayant une interface graphique.

9.3.4 Génération et utilisation des nombres aléatoires

Dans la section précédente, nous avons détaillé les différentes utilisations de l'aléatoire de manière à ce que le modèle puisse être répliqué au plus près. Dans l'implémentation, nous avons fait en sorte qu'il soit possible d'utiliser plusieurs classes de générateurs aléatoires (Pseudo Random Number Generator PRNG) différentes pour peu que celles-ci possèdent une interface identique à celle de la classe `Random` fournie par le langage Java. La *graine* (seed) utilisée pour initialiser le générateur aléatoire nous a par ailleurs servi de numérotation pour les expériences. Ce qui nous a permis de vérifier qu'une même simulation donne invariablement le même résultat. Par ailleurs, le générateur est partagé par l'ensemble des entités de la simulation.

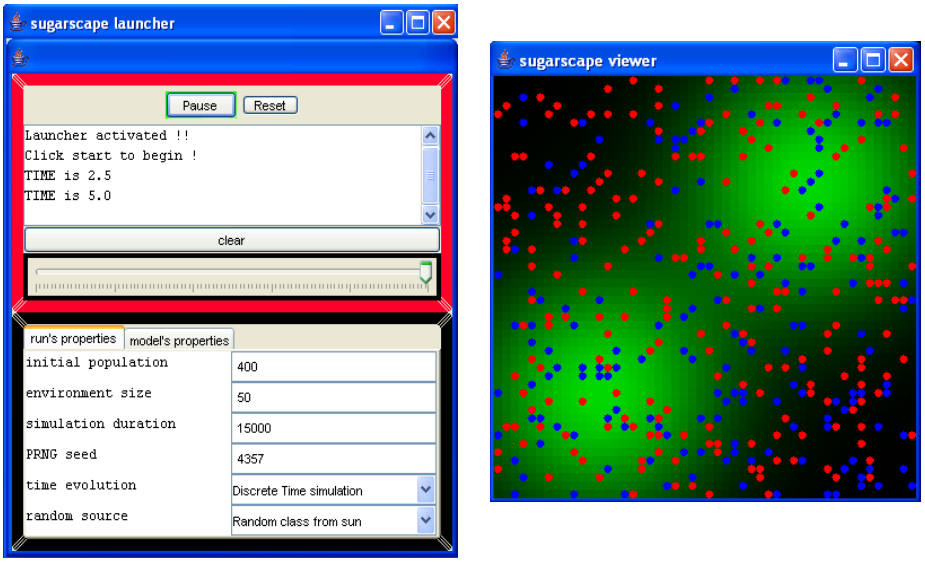

FIG. 9.9 – Contrôle, paramétrisation et visualisation via des agents MADKIT.

Les nombres aléatoires sont donc générés en fonction des besoins en utilisant les méthodes définies par l'interface du générateur. Par exemple, le code correspondant à l'initialisation des variables d'un agent appartenant à la population initiale est le suivant :

```
boolean sex=getRandomSource().nextBoolean(); //getRandomSource() is the selected PRNG
int fov = getRandomSource().nextInt(5)+1;
double mu = getRandomSource().nextDouble()*(maxMU-minMU)+minMU;
double lambda = getRandomSource().nextDouble()*(maxLAMBDA-minLAMBDA)+minLAMBDA;
double alpha = getRandomSource().nextDouble()*(maxALPHA-minALPHA)+minALPHA;
double omega;
if(sex)
        omega = getRandomSource().nextDouble()*(maxMaleOMEGA-minMaleOMEGA)+minMaleOMEGA
            ;
else
        omega = getRandomSource().nextDouble()*(maxFemaleOMEGA-minFemaleOMEGA)+
            minFemaleOMEGA;
int Xlocation;
int Ylocation;
do
{
        Xlocation = getRandomSource().nextInt(squareSize);
        Ylocation = getRandomSource().nextInt(squareSize);
}
while(grid[Xlocation][Ylocation].getOccupant()!=null);
```

De plus, nous avons utilisé les facilités proposées par le langage Java pour la randomisation des listes. Comme nous l'avons expliqué, tous les objets qui sont manipulés dans la simulation peuvent être classés suivant un ordre total. A partir de là, pour effectuer un tirage aléatoire sur une liste d'agents, d'influences ou de perception, nous avons utilisé les méthodes proposées par la classe `Collections` qui permettent très simplement de classer une liste, en utilisant `Collections.sort(laListe)` et de la randomiser, avec `Collections.shuffle(laListe,getRandomSource())`. L'algorithme de cette méthode, disponible dans les sources du langage Java, correspond à la réalisation d'une permutation aléatoire sur tous les index de la liste, du dernier au premier. Nous donnons ici ces précisions car il est évident que la manière dont une liste est randomisée peut influencer la dynamique du système et c'est un point du modèle dont nous n'avions pas encore donné la spécification.

9.3.5 A propos de l'implémentation du mode événementiel

Si l'implémentation à pas de temps constant n'a pas posé de problème particulier, il n'en a pas été de même en ce qui concerne le mode événementiel du simulateur. En fait, il n'a tout simplement pas été possible d'appliquer à la lettre l'algorithme que nous avons proposé pour des raisons de temps d'exécution prohibitif. En effet, le principe de cet algorithme consiste à effectuer le calcul de l'évolution endogène de l'environnement (régénération, consommation, mort naturelle, etc.) avant chaque événement qui doit intervenir. De fait, lorsque 1000 agents sont présents dans le système, appliquée telle quelle, cette mise à jour rend la simulation environ 1000 fois plus lente que le mode de simulation synchrone. Sachant que la puissance de calcul dont nous disposions nous permettait de faire environ 15 000 itérations par heure en mode synchrone, la simulation événementielle était tout simplement inexploitable en l'état. C'est pourquoi il était nécessaire de réaliser une optimisation du principe simulation asynchrone. Nous avons notamment utilisé une optimisation assez courante dans ce type de simulation qui consiste à effectuer la mise à jour des différentes entités du système uniquement en cas de besoin. Par exemple, la quantité de ressources présente sur une cellule est mise à jour uniquement lorsque celle-ci est accédée, pour la perception d'un agent par exemple[16]. Idem pour l'état physique d'un agent. Il est alors important de bien appliquer les équations temporelles du modèle en gardant trace de la dernière mise à jour. A ce propos, il nous semble qu'il existe une erreur dans les équations temporelles proposées par L-P. En effet, avec les notations que nous avons utilisées, L-P donne les spécifications suivantes pour la régénération de la ressource et le calcul de la mise à jour de l'énergie des agents :

$$R_{(x,y)}(t+dt) = \min(C_{(x,y)}, R_{(x,y)}(t) + dt * \rho_{(x,y)}) \tag{9.7}$$

$$w_a(t+dt) = w_a(t) + R_{(x,y)}(t+dt) + dt * \mu_a \tag{9.8}$$

L-P fait par ailleurs la supposition que la ressource est immédiatement collectée par un agent qui se trouve sur une cellule, comme nous l'avons fait nous aussi. Le problème vient de ce que ces équations sont corrélées. L'équation qui calcule l'énergie d'un agent utilise le résultat donné par l'équation qui met à jour la quantité de ressource présente dans une cellule. Il faut alors remarquer que ces équations ne rendent pas correctement compte de la dynamique supposée. En effet, le calcul de la ressource présente sur une cellule ne tient pas compte du fait qu'elle soit occupée ou libre. De fait, la quantité calculée est limitée par la capacité de celle-ci. Or la ressource régénérée est censée être récupérée au fur et à mesure par les agents, cette quantité ne doit donc pas être limitée par la capacité de la cellule. Prenons un exemple. Soit pour un instant $t = 0$ tel qu'une cellule est occupée par un agent avec $R_{(x,y)}(0) = 0$, $C_{(x,y)} = 2$, $\rho_{(x,y)} = 1$, $w_a(0) = 10$ et μ_a. Prenons $dt = 4$, les équations de L-P nous donnent $R_{(x,y)}(5) = 2$, donnée qui est immédiatement utilisée dans la seconde équation de sorte que $w_a(5) = 10+2-4 = 8$. Ce qui est incorrect si l'on considère effectivement que l'agent récupère la ressource au fur et à mesure de sa régénération. Il est donc nécessaire d'effectuer un calcul différent suivant que la cellule soit occupée ou non, comme nous l'avons fait dans les équations 9.4 et 9.5. Equations qui donnent $w_a(5) = 10 + 4 - 4 = 10$. μ_a et $\rho_{(x,y)}$ étant identiques, il est normal que l'énergie de l'agent reste stable.

Un autre exemple de la complexité de l'implémentation événementielle concerne le calcul de la mort naturelle d'un agent. Après chaque activation d'un agent, il est nécessaire de vérifier que celui-ci est capable de survivre jusqu'à la date de sa prochaine exécution. Il est en effet possible que son métabolisme ne lui permette pas d'atteindre cette date. Il est alors

[16] Ce qui est d'ailleurs visible dans l'affichage de la simulation.

important de calculer la date à laquelle la mort naturelle doit intervenir de manière à ne pas introduire de biais sur le reste de la dynamique. Rappelons ici qu'une cellule occupée peut empêcher la naissance d'un agent. Tout cela pour dire que l'implémentation du mode événementiel recèle de nombreux pièges et que, malgré toute l'attention que nous lui avons portée, nous n'excluons pas d'avoir fait nous-même des erreurs dans son application. Ces différents éléments ne nous étant apparus qu'au moment de l'implémentation, cela montre aussi la limite des spécifications que nous avons données à ce propos. En effet, même si le modèle que nous avons donné contient implicitement la gestion des différents événements générés par le système, la complexité associée aux différents traitements est sans aucun doute une source de biais potentielle.

9.3.6 Consultation et téléchargement du simulateur

L'ensemble du simulateur et de son code source sont disponibles sous plusieurs formes. En premier lieu, ils sont distribués sous la forme d'un plugin de la plate-forme MadKit dans sa version 4[17]. Par ailleurs, une version fonctionnant sous la forme d'une applet Java peut être consultée et utilisée à l'adresse suivante www.lirmm.fr/∼fmichel/sugarscape. A la même adresse, on peut également télécharger une distribution légère et fonctionnelle du simulateur ne nécessitant aucune installation[18].

9.4 Quelques résultats expérimentaux

Nous allons maintenant présenter les résultats que nous avons obtenus avec le simulateur selon les deux principes d'évolution temporelle que nous avons modélisés : par discrétisation régulière du temps (mode synchrone) et par événements (mode asynchrone). Ceci nous permettra de faire une comparaison qualitative entre nos résultats et ceux qui ont été obtenus par L-P où la même démarche a été réalisée. La figure 9.10 est un extrait des résultats obtenus par L-P. Rappelons ici que les résultats publiés dans L-P montrent un comportement oscillatoire de la population pour quasiment toutes les simulations en mode synchrone et à l'inverse une population stable pour presque tous les cas en mode asynchrone. Pour commencer, nous utiliserons les mêmes paramètres initiaux que L-P dans nos expériences, c'est-à-dire une population initiale $Pop(0) = 400$ et une grille de 50 de côté[19]. Rappelons ici tout de même qu'il ne s'agit pas de notre part d'une réplication à l'identique de l'expérience de L-P.

9.4.1 Premiers résultats : utilisation de la classe `Random` pour PRNG

Pour cette première série d'expériences, nous avons utilisé un PRNG correspondant à l'utilisation de la classe `Random` fournie par le langage Java. Comme on peut le voir sur la figure 9.11, il aurait été difficile d'obtenir des résultats plus hétérogènes, même si d'un point de vue global nous obtenons des résultats assez similaires à ceux de L-P en ce qui concerne les simulations par événements, à savoir une population très stable dans le temps. Ceci doit cependant être fortement nuancé car nous obtenons une simulation ($seed = 4357$)

[17] sourceforge.net/projects/madkit/.

[18] Cette version ne nécessite pas l'installation de la plate-forme MadKit et elle contient tous les éléments nécessaires à l'exécution. Un environnement d'exécution Java (JRE) installé sur la machine hôte est cependant indispensable.

[19] Nous garderons une grille de 50 dans nos expériences. La période de gestation est par ailleurs égale à 0.

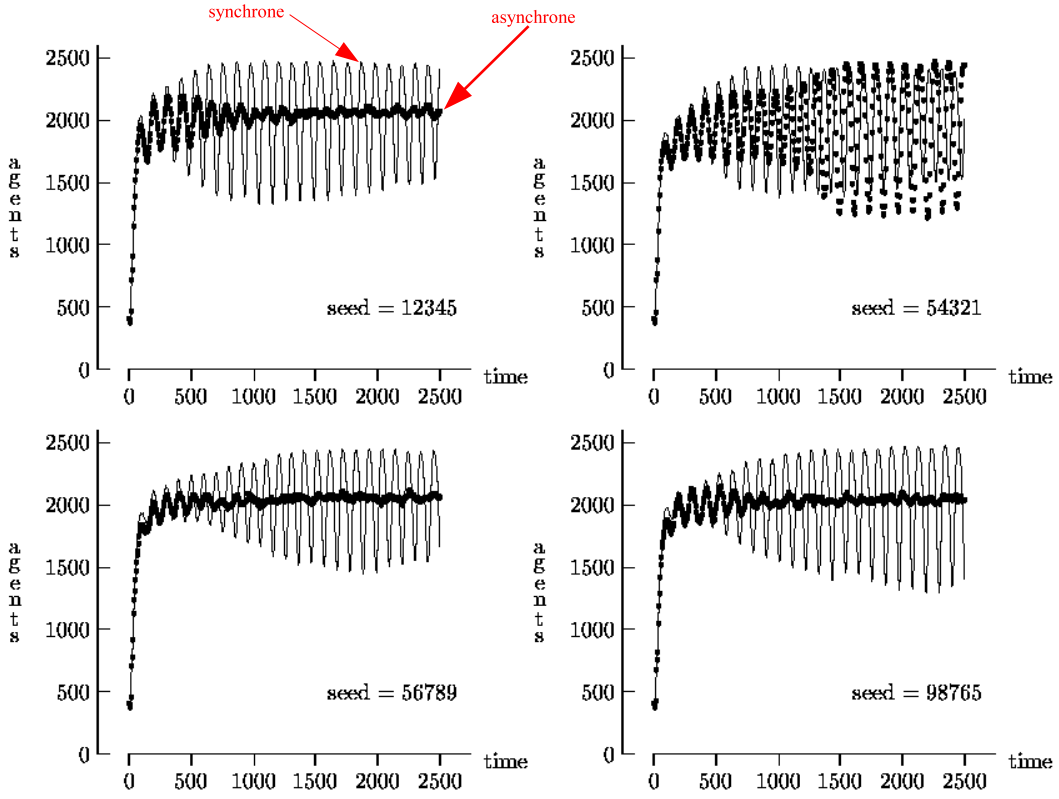

FIG. 9.10 – Résultats obtenus par L-P avec $Pop(0) = 400$ [Lawson & Park, 2000].

où la population oscille dans ce mode[20]. Ce qui est d'ailleurs le cas dans quelques-unes des simulations publiées par L-P et dont les auteurs n'ont pas fait beaucoup de cas dans leurs conclusions. Même si nous n'avons trouvé que rarement des oscillations en mode asynchrone, ce résultat reste important en proportion.

Par contre, le moins que l'on puisse dire est que la simulation synchrone donne des résultats très disparates et bien différents de ceux de L-P. Dans certains cas nous obtenons de fortes oscillations alors que dans d'autres la population reste stable, cela dans des proportions équivalentes. Il semble ainsi que le système soit extrêmement sensible aux conditions initiales avec ce principe de simulation et nous essaierons de comprendre pourquoi un peu plus loin. Dans un premier temps, nous tenions tout d'abord à faire une comparaison entre les résultats obtenus par différents générateurs aléatoires. Nous allons maintenant présenter une autre série de résultats qui utilisent un algorithme différent pour cette partie de l'implémentation.

9.4.2 Changement du générateur de nombres pseudo aléatoires

Sachant que la seule différence qu'il y avait entre deux expériences était la graine utilisée pour le générateur aléatoire, il était intéressant de tester la sensibilité de la simulation à ce paramètre[21]. Il s'avère en fait que le PRNG implémenté par la classe `Random` du langage Java

[20] Ce qui est assez conséquent étant donnée la faible taille de l'échantillon ici présenté.

[21] En fait, après une analyse poussée de l'exécution d'une simulation, il s'avère que nous faisons bien sûr un usage assez intensif du PRNG. Par exemple, durant les 500 premières itérations, pas moins de 1 500 000 d'appels sur la méthode `Random.nextInt(int)` sont réalisés en mode synchrone, ce qui est beaucoup sachant que cela inclut les itérations initiales où le nombre d'agents est relativement faible.

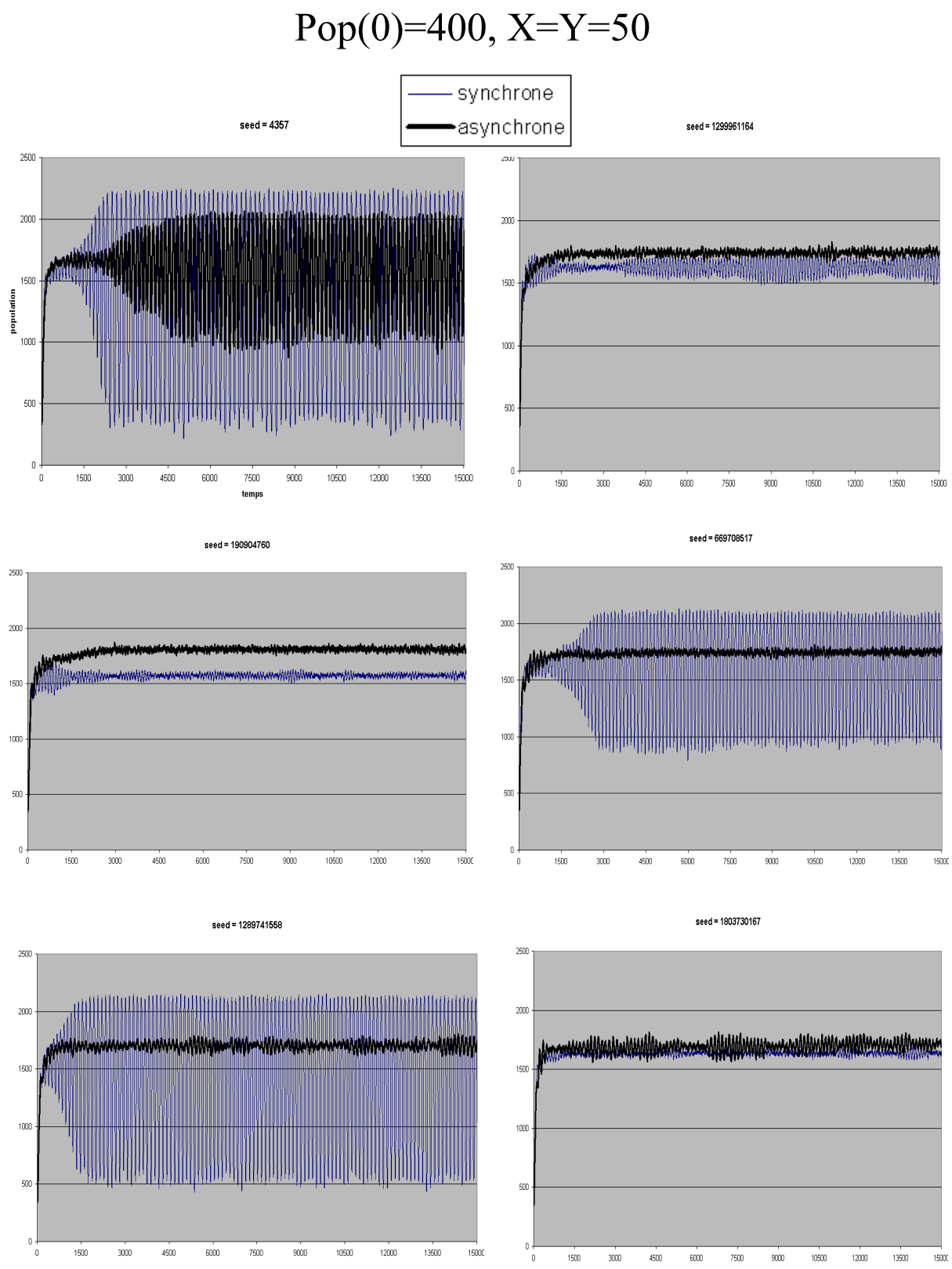

FIG. 9.11 – Résultats obtenus avec `Random` en tant que PRNG.

a une très mauvaise réputation[22]. Cela nous a conduit à utiliser un nouveau PRNG basé sur un algorithme relativement récent appelé *Mersenne Twister* [Matsumoto & Nishimura, 1998]. Nous avons ainsi utilisé une adaptation Java[23] de cet algorithme en lieu et place de la classe `Random`. Dans la suite, tous les résultats que nous présenterons ont été générés à l'aide de ce PRNG. La figure 9.12 montre les résultats que nous avons obtenus avec ce nouveau PRNG.

Bien que ces nouveaux résultats soient générés en utilisant les mêmes graines pour le PRNG, il serait faux de comparer les résultats deux à deux. En effet, le PRNG étant utilisé pour générer les premiers agents, les états initiaux sont tout à fait différents. Cependant, les résultats peuvent être comparés dans la globalité des dynamiques obtenues. Et à ce niveau-là, bien que le système soit encore un peu plus stable en ce qui concerne le mode asynchrone, on constate encore une fois des oscillations de la population pour certains cas du mode synchrone. Cette fois encore, l'état initial du système semble toujours jouer un rôle très important dans la dynamique obtenue.

9.4.3 Modification de la génération de l'état initial

Essayons maintenant de donner une explication à ces fameuses oscillations. Que signifient-elles ? En fait, une oscillation correspond empiriquement, dans le sens descendant, à la mort quasi simultanée (sur quelques périodes seulement) d'un très grand nombre d'agents. De façon duale, en phase montante, l'oscillation correspond à un très grand nombre de naissances. A y regarder de plus près, il est facile d'établir une corrélation entre ces deux phénomènes. Il suffit de remarquer que, étant donné que les paramètres initiaux utilisés pour la régénération de la ressource créent une situation d'abondance de celle-ci, la véritable ressource critique du système est l'espace. Lorsqu'un très grand nombre d'agents meurt (phase descendante), beaucoup d'espace est par conséquent libéré. Ce qui engendre naturellement un très grand nombre de naissances du fait de la place disponible et ainsi un nombre important de nouveaux agents (phase montante). Ces nouveaux agents vont occuper l'espace et réduire de fait le nombre de naissances. Mais ils vont aussi tous mourir approximativement au même moment, c'est-à-dire en quelques itérations. Ce qui va créer une phase descendante et ainsi produire un cycle dans la dynamique de la population. C'est pourquoi, une première oscillation suffisamment forte à toutes les chances de rendre le système définitivement instable.

A partir de cette hypothèse, il faut remarquer que la première oscillation d'importance est en fait générée par l'état initial lui-même. En effet, tel quel, l'état initial consiste dans une sorte de génération spontanée plus ou moins uniforme. Par conséquent, il est naturel de considérer qu'il s'agit là d'un paramètre susceptible d'influencer très fortement la dynamique du système dans son ensemble. Cela alors que le cadre expérimental suggère que l'expérimentation vise à étudier l'évolution *normale* d'une société artificielle au cours du temps. Ce qui nous semble en contradiction avec un état initial qui modélise une génération spontanée créée ex nihilo. C'est pourquoi nous avons modifié l'algorithme qui génère l'état initial de telle sorte que la

[22] Le lecteur pourra à ce sujet se référer à la page web www.alife.co.uk/nonrandom qui met en évidence le problème.

[23] Adaptation que nous avons utilisée a été réalisée par Sean Luck, `MersenneTwisterFaste.java`, à partir d'une première implémentation de Michael Lecuyer. Il existe aussi une implémentation du *Mersenne Twister* distribuée dans la librairie *Colt* (Open Source Libraries for High Performance Scientific and Technical Computing in Java hoschek.home.cern.ch/hoschek/colt/) qui a été développée par le CERN (Centre Européen de Recherche Nucléaire). Nous avons aussi essayé d'utiliser cette version, mais cela a nécessité une extension de la classe originale de manière à ce qu'elle puisse être utilisée dans nos algorithmes. Ce qui nous a fait dire que nous avions potentiellement biaisé le PRNG. C'est pourquoi nous n'avons pas retenu cette version.

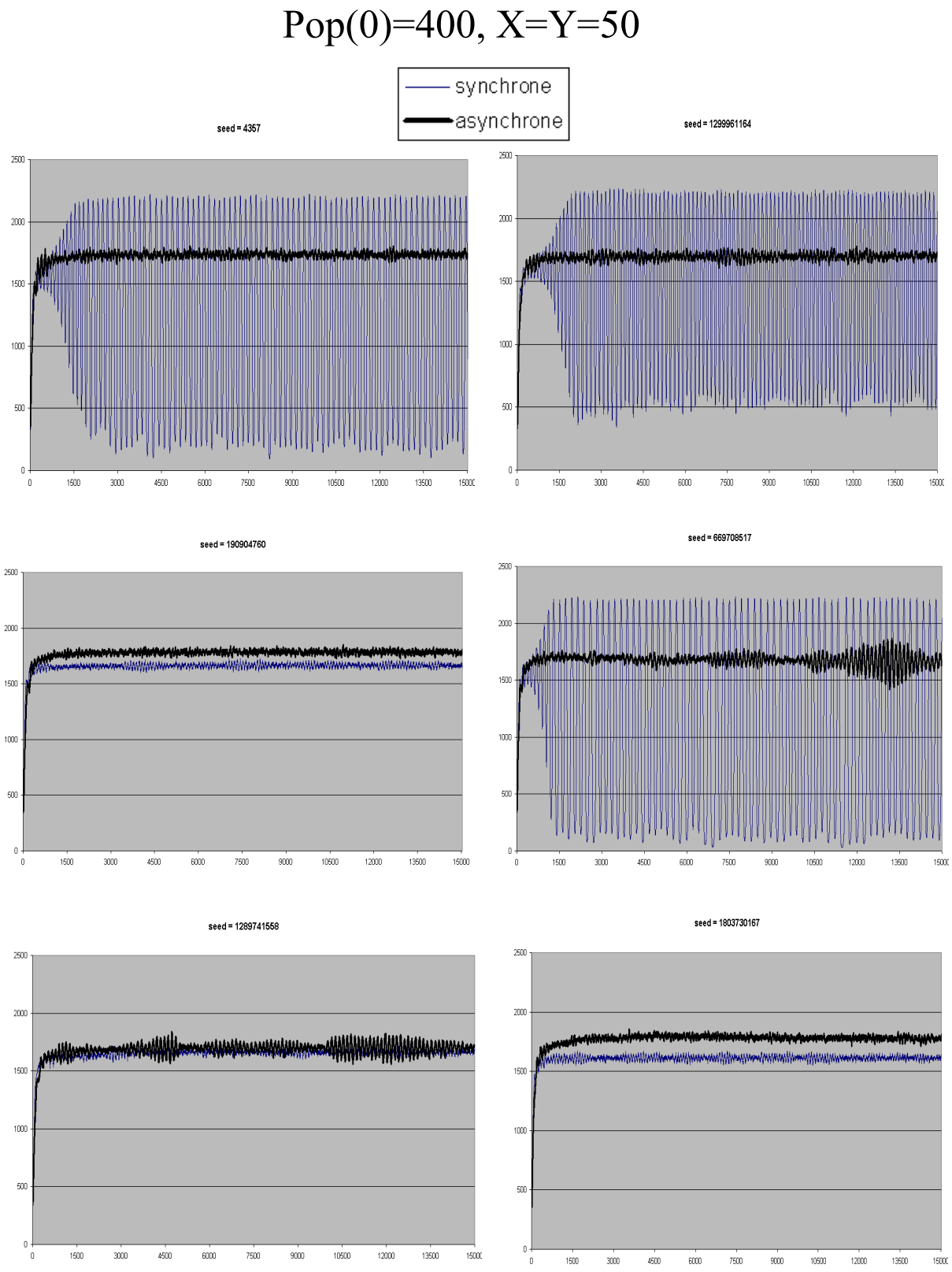

FIG. 9.12 – Résultats obtenus avec `MersenneTwisterFast` en tant que PRNG.

population créée comporte des agents de tous âges et de condition physique $w_n(0)$ différentes. La figure 9.13 montre les résultats que nous avons obtenus dans ces conditions.

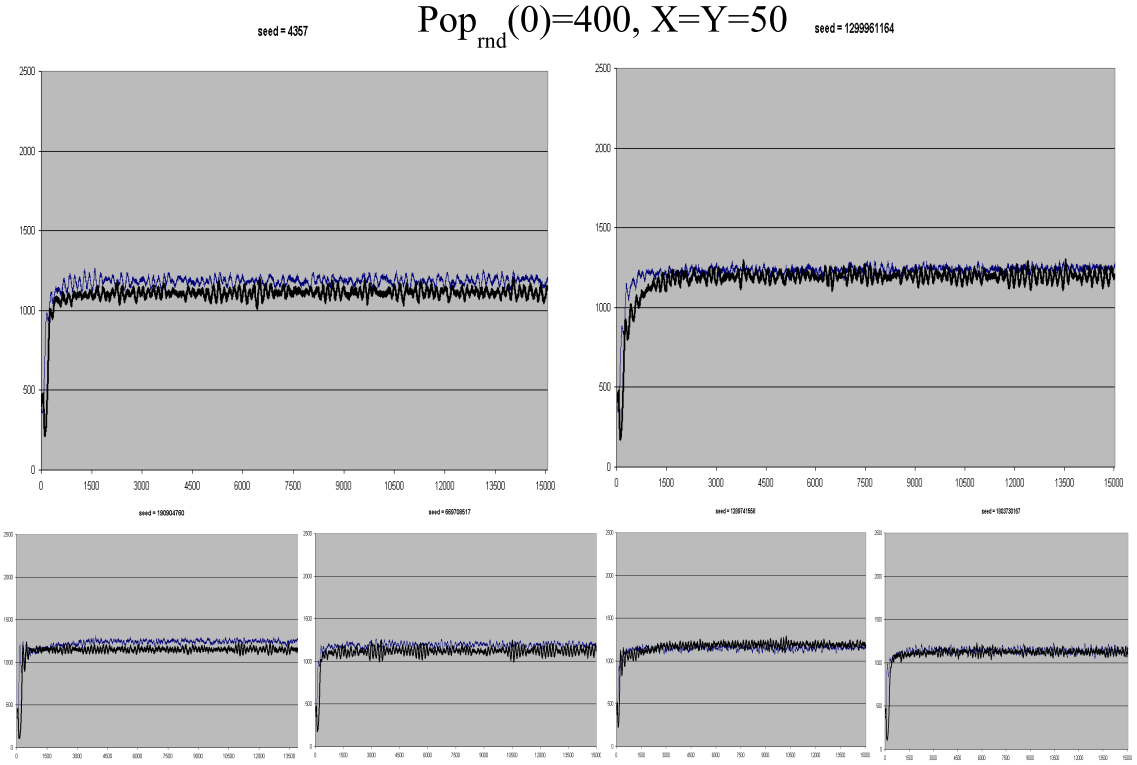

FIG. 9.13 – Résultats obtenus pour $Pop(0) = 400$ initialement randomisée.

Comme on peut le voir sur la figure 9.13, nous avons cette fois obtenu une population stable pour toutes les simulations que nous avons effectuées. Ce qui tend à prouver notre hypothèse précédente sur le rôle que joue l'état initial sur la dynamique globale de la population. Cependant, à y regarder de plus près, ces simulations ne sont pas suffisantes pour pouvoir conclure si rapidement. En effet, il faut remarquer que dans toutes ces simulations, la population n'atteint jamais le niveau critique d'occupation de l'espace qui se situe aux alentours de 1500 agents. Ce qui correspond à l'occupation de l'ensemble des cellules dignes d'intérêt, c'est-à-dire possédant une capacité environ supérieure à 0.5.

C'est pourquoi nous avons réalisé quelques expériences supplémentaires dans lesquelles nous avons placé un nombre initial d'agents très élevé, 1600, de manière à provoquer une première phase descendante du fait du manque d'espace. On aura donc en quelque sorte une première oscillation forcée. Pour affiner cette étude de l'influence de l'état initial, nous ferons également des simulations basées sur une population initiale uniforme avec ce même paramètre pour $Pop(0)$ de manière à confirmer ou infirmer notre hypothèse.

Comme le montrent les figures 9.14 et 9.15, les deux séries de simulations exhibent des différences très importantes. Les simulations où la population initiale est uniforme présentent assez souvent une grande instabilité de la population. Au contraire, dans toutes les simulations que nous avons réalisées, lorsque la population initiale n'est pas homogène, le système finit par atteindre un équilibre, et ce quel que soit le mode de simulation. Sur la base de ces quelques expériences, il semble clair que l'uniformité de l'état initial joue un rôle extrêmement important dans la dynamique globale qui est obtenue. Nous avons par ailleurs réalisé d'autres expériences

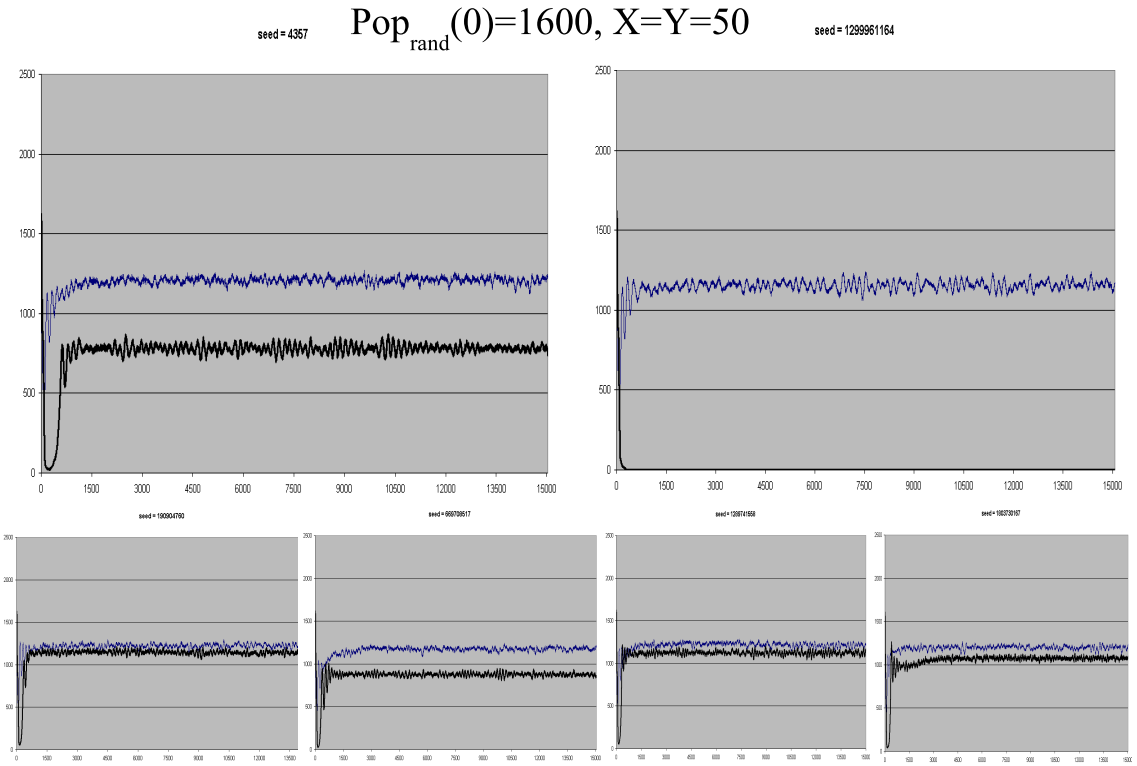

FIG. 9.14 – Résultats obtenus pour $Pop(0) = 1600$ initialement randomisée.

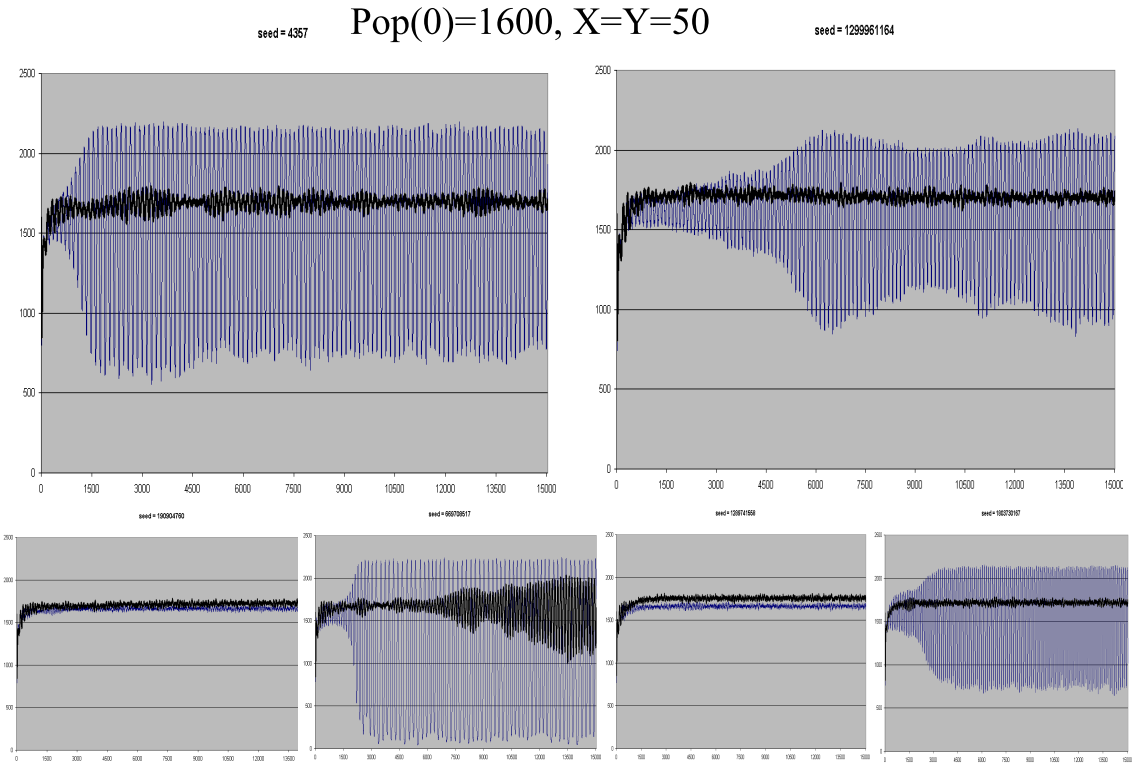

FIG. 9.15 – Résultats obtenus pour $Pop(0) = 1600$ initialement uniforme.

avec différentes populations initiales dans lesquelles nous avons retrouvé ce phénomène. Dans toutes nos expériences, le système s'est comporté très différemment suivant que la population initiale était uniforme ou randomisée.

9.4.4 Analyses et hypothèses

Aux vues des différentes expériences que nous avons pu mener, nous allons ici nous risquer à quelques conclusions et hypothèses. Tout d'abord, nous ne pouvons pas faire le même constat que celui qui est proposé dans L-P à propos de l'instabilité de la population en mode synchrone, et de sa stabilité en mode asynchrone. En effet, nous avons obtenu de nombreux résultats qui nous disent le contraire. Par ailleurs, nous nous sommes essayé à mettre en évidence l'importance de la nature de la population initiale sur le comportement global du système. Et à ce propos, il nous semble évident que la façon dont la population initiale est générée constitue un facteur très important du modèle et nous pensons que l'influence de l'état initial sur les résultats ne peut sérieusement pas être négligée.

De là à dire que les conclusions tirées par L-P, et par extension [Epstein & Axtell, 1996], sont fortement biaisées par un état initial trop prééminent dans les résultats, il n'y a qu'un pas. Cependant, nous ne le franchirons pas et il y a plusieurs raisons à cela. Tout d'abord, notre modèle n'a fait l'objet d'aucune des méthodes de vérification/validation que nous avons évoquées tout au long de ce document, notamment le principe vérification/validation indépendante dont nous avons beaucoup vanté les mérites (IV&V cf. section 4.4.2 page 89). Par ailleurs, pour pouvoir définitivement conclure à propos de l'importance de l'état initial, il nous aurait fallu un échantillon d'expériences beaucoup plus large. Cependant nous avons été limité par les puissances de calcul dont nous disposions et même en utilisant plusieurs ordinateurs simultanément, il a été assez fastidieux de réaliser la centaine de simulations qui nous a permis de traiter les quelques hypothèses que nous avons présentées. Par voie de conséquence, nous avons préféré effectuer beaucoup de simulations pour un nombre restreint de configurations initiales plutôt que le contraire. Il serait ainsi important d'aller plus loin dans l'exploration des configurations initiales.

Nous avons cependant maintenant une forte intuition en ce qui concerne la stabilité de la dynamique du système dans son ensemble, quel que soit le mode de simulation. Et il s'agit là d'une perspective de travail très intéressante car elle apporterait un point de vue radicalement différent de celui proposé dans L-P qui, lui, repose sur l'idée que le mode asynchrone convient mieux à la simulation de sociétés artificielles. Etant donnée la souplesse de modélisation qu'offre le principe de simulations par événement, il n'est bien sûr pas question de démontrer le contraire. Nous pensons plutôt que la véritable différence qu'il y a entre les deux modes d'application vient de ce que le mode synchrone exacerbe les biais de simulations, quelle que soient leurs origines (conditions initiales, erreurs de programmation, etc.). Pourquoi ? Ces différentes expériences soulèvent en effet une question intéressante : pourquoi le système est-il beaucoup moins sensible aux paramètres initiaux dans les simulations événementielles ?

Notre hypothèse est la suivante. En mode asynchrone, il n'est pas rare qu'un agent agisse deux fois plus souvent qu'un autre sur l'ensemble de sa durée de vie. En effet, le processus utilisé pour générer les dates correspondant aux comportements des agents étant poissonnien, certains agents peuvent tout à fait être activés plusieurs fois par unité de temps. De la même manière, d'autres ne seront activés que trois fois pour dix unités de temps par exemple. Bien sûr, le nombre d'actions par agent est statistiquement le même, mais la durée de vie d'un agent peut parfois être assez courte. C'est pourquoi nous pensons qu'il est tout à fait possible que

le mode événementiel ait pour conséquence d'atténuer le caractère uniforme de la population initiale. En effet, si certains agents agissent deux fois plus que d'autres, cela revient en quelque sorte à ce qu'ils vivent deux fois plus longtemps, au sens du mode synchrone. Ce qui rend en quelque sorte l'état initial moins homogène. Nous n'en dirons ici pas plus car il ne s'agit là que de suppositions que nous n'avons pas le temps de vérifier expérimentalement. De plus, dans cette section nous avons largement dépassé les frontières du cadre expérimental que nous nous étions initialement fixé.

9.5 Conclusion du chapitre

Dans ce chapitre, nous avons présenté la modélisation et la simulation d'une société artificielle d'agents, de la définition du cadre expérimental jusqu'à l'exploitation des résultats. Cette expérience nous a permis, d'une part de mettre en œuvre les différents principes de modélisation que nous avons précédemment identifiés, et d'autre part d'illustrer plus en détail ce que peut-être l'utilisation du modèle MIC* pour la définition d'un modèle de simulation multi-agents basé sur le principe Influence/Réaction. Par ailleurs, cela a été aussi pour nous l'occasion d'illustrer notre approche de modélisation en quatre modules : *environnement*, *comportement*, *interaction* et *ordonnancement*. Cette division a facilité une description claire des différents points du modèle. Cette expérience constitue ainsi en quelque sorte une première validation expérimentale de notre approche.

La modélisation que nous avons effectuée nous a aussi permis de mettre en évidence l'intérêt de l'utilisation du modèle MIC* pour la simulation multi-agents. Premièrement il permet de se concentrer sur la modélisation de l'interaction grâce à l'utilisation de concepts explicites comme les objets d'interaction et les lois d'interaction. Deuxièmement, il permet une application sans ambiguïté de la distinction esprit/corps en séparant clairement les processus de calcul, les agents, de leurs représentations dans l'environnement situé qui, elles, sont modélisées par des objets d'interaction. Par ailleurs, comme nous l'avons expliqué, l'application du principe Influence/Réaction est une conséquence naturelle de l'utilisation du modèle MIC* et de la distinction esprit/corps qu'elle engendre. A ce propos, rappelons que l'apport majeur du modèle MIC* repose sur une **séparation claire entre le temps du calcul et le temps de l'interaction**. Ce qui permet effectivement d'abandonner la représentation de l'action comme une modification d'un état global. Ce qui constitue pour nous le point fondamental de notre approche.

Chapitre 10

Conclusions et Perspectives

En considérant que la dynamique globale d'un système complexe peut-être modélisée par un ensemble d'entités autonomes qui interagissent dans un environnement commun, la simulation multi-agents offre aujourd'hui une alternative très séduisante aux représentations classiques des systèmes complexes. De plus, l'approche multi-agents n'est pas restreinte à un domaine particulier et le paradigme qu'elle définit peut être utilisé pour modéliser toutes sortes de systèmes. C'est pourquoi, de par son aspect novateur et interdisciplinaire, la simulation multi-agents représente à n'en pas douter un paradigme de modélisation qui possède un énorme potentiel dans le cadre de l'étude des systèmes complexes.

10.1 Synthèse

10.1.1 Sur la problématique

Dans cette thèse, nous avons vu que la capacité d'adaptation de l'approche multi-agents possède à l'heure actuelle une contrepartie négative. Du fait de l'hétérogénéité des modèles considérés, il est en effet très difficile d'en extraire des méthodologies et des formalismes qui puissent être appliqués quel que soit le contexte de l'expérience. La conséquence directe de cette difficulté est qu'il n'existe aujourd'hui pas de correspondance claire entre les modèles destinés à être simulés et des structures informatiques clairement établies. Les modèles multi-agents sont ainsi le plus souvent élaborés sans référence à un système informatique cible et les spécifications proposées sont généralement insuffisantes pour ne pas prêter à interprétation lors de l'implémentation. Autrement dit elles sont ambiguës. Par conséquent, un même modèle multi-agents peut être implémenté de plusieurs façons différentes et potentiellement donner des résultats très éloignés ; c'est ce que nous avons appelé le *phénomène de divergence implémentatoire*.

Nous avons vu que l'existence de ce phénomène est mise en évidence par la quasi impossibilité de répliquer les modèles qui sont proposés dans la littérature. Outre le fait que cela diminue largement l'intérêt des expériences qui sont réalisées, nous avons vu que cela pose un problème majeur car la *relation de simulation*, fondamentale pour toute expérience de simulation, n'est généralement pas vérifiée. Cette relation stipule entre autres que l'implémentation du simulateur doit être neutre vis-à-vis du modèle. Ce qui est contradictoire avec la possibilité d'avoir des phénomènes de divergence implémentatoire, eux-mêmes engendrés par le manque de spécifications non ambiguës. La vérification de cette relation est ainsi le plus

souvent ignorée dans les expériences de simulation multi-agents. Ce qui pose le problème de la vérification et de la validation des expériences. Sans spécifications claires, la relation de simulation ne peut être étudiée et la simulation multi-agents reste un processus expérimental où les principes fondamentaux de la théorie de la M&S ne peuvent être appliqués. Résoudre ce problème constitue à n'en pas douter un enjeu majeur pour les années à venir.

C'est pourquoi nous avons considéré que le manque de formalisation dont souffrent les modèles multi-agents est sans aucun doute le point sur lequel les efforts doivent être concentrés. Toute la question est de mettre effectivement en relation les spécifications des modèles multi-agents avec des structures informatiques concrètes. Le but de cette thèse a été de contribuer à cet effort. Nous avons cependant insisté sur le fait que de très nombreux facteurs jouent un rôle dans cette problématique. La complexité associée à l'implémentation d'un système multi-agents et le manque de formalismes adéquats en sont les principaux exemples. Il n'existe ainsi pas de solution immédiate à cette problématique. Dans cette thèse, nous avons pris le parti de considérer le problème sous deux angles différents, mais cependant complémentaires.

10.1.2 Première approche

Dans un premier temps, nous avons soutenu qu'il était important de rendre explicite les principes de conception et d'implémentation des simulateurs multi-agents. Notre idée est que si les spécifications d'un modèle multi-agents doivent correspondre à des structures informatiques concrètes, il est alors pour le moins nécessaire de les rendre explicites et de clairement les définir. Si le fonctionnement des simulateurs multi-agents reste "ésotérique", il est bien sûr normal qu'on ne puisse pas en établir des spécifications pertinentes. A l'image de ce que propose l'approche VOYELLES dans le cadre de l'ingénierie des systèmes multi-agents, nous avons tout d'abord suggéré une approche méthodologique qui repose sur l'identification des aspects fondamentaux inhérents à tous modèles de simulation multi-agents. Nous avons ainsi proposé de faire une distinction explicite entre la modélisation/implémentation des *comportements*, de l'*environnement*, des *interactions* et de l'*ordonnancement*. Il s'agit respectivement de se concentrer sur l'architecture interne des agents, la modélisation du monde physique dans lequel ils évoluent, la représentation des interactions qu'ils engendrent et la manière dont l'évolution temporelle du système est mise en place.

Dans ce contexte, nous avons proposé des outils de conception qui n'encapsulent pas le fonctionnement du simulateur et qui tendent à le rendre explicite sans pour autant imposer un principe d'implémentation définitif pour chacun de ces différents aspects. Dans un premier temps, le caractère générique de cette approche a été motivé par l'idée que chaque expérience de simulation multi-agents est singulière et qu'elle définit des besoins particuliers qui ne doivent pas être remis en cause par un environnement de développement trop restrictif. Basés sur l'utilisation des couples *agent scheduler/activateur* et *agent watcher/sonde*, ainsi que sur l'élaboration d'une structure organisationnelle explicite, ces outils ont prouvé leur réutilisabilité et leur généricité à travers les nombreuses expériences de simulation qu'ils ont permis de réaliser dans des domaines aussi divers que la robotique mobile collective, la vie artificielle ou le jeu vidéo. Dans un deuxième temps, nous avons vu que l'utilisation d'une structure organisationnelle basée sur les principes du modèle AGR permet d'exploiter pleinement le pouvoir de réflexivité des langages objets. Le deuxième objectif de ces outils est ainsi de faciliter l'analyse et l'exploration des modèles et de leurs implémentations. L'agentification des différentes composantes du simulateur permet de s'affranchir des problèmes liés à l'hétérogénéité des architectures internes des différentes entités qui composent l'ensemble du système, qu'il s'agisse du moteur de simulation ou du modèle à simuler lui-même. Nous avons

à ce sujet proposé un *pattern organisationnel* qui peut être utilisé quel que soit le contexte de l'expérience.

Rendre plus explicite les structures informatiques utilisées pour simuler un système multi-agents est un premier pas vers la mise en relation entre les spécifications des modèles et leurs implémentations. Cependant, nous avons vu que l'aspect informel des modèles multi-agents tient aussi au manque de formalismes adéquats. C'est pourquoi il est aussi nécessaire d'agir en amont et de fournir un effort sur la nature des spécifications qui doivent être associées au paradigme agent. Si l'on souhaite que les structures informatiques qui sont mises en jeu dans la simulation d'un système multi-agents puissent être clairement identifiées, il est primordial d'essayer de cerner quels sont les besoins sous-tendus par cette approche de modélisation.

10.1.3 Deuxième approche

La deuxième approche de notre thèse a donc consisté dans une réflexion plus globale sur le paradigme agent. Il nous semble évident que le manque de formalismes dont souffrent les modèles multi-agents est entretenu par le fait que le paradigme lui-même ne propose pas de définitions claires en ce qui concerne les différents concepts manipulés. C'est pourquoi, tout au long de ce manuscrit, nous avons essayé d'extraire des contraintes méthodologiques, pratiques et conceptuelles qui soient inhérentes au paradigme agent et qui puissent donc être reprises quel que soit le domaine d'application considéré. Dans ce cadre, nous avons notamment essayé de démontrer qu'il est aujourd'hui fondamental de reconsidérer la manière dont l'action d'un agent est généralement modélisée. Héritée des applications issues de l'intelligence artificielle classique, la représentation de l'action comme une modification d'un état global doit être abandonnée au profit de la notion d'influence.

Les conséquences positives de ce changement de point de vue radical sont nombreuses. Tout d'abord, dans le cadre du principe Influence/Réaction, nous avons vu que cela permet de traiter le problème de la représentation des actions simultanées grâce à une distinction explicite entre le calcul de l'action délibérée par un agent, une influence, et le calcul de son résultat sur l'environnement lors de la phase de réaction. Problème essentiel dans le domaine des systèmes multi-agents où le résultat d'une action n'est pas uniquement lié à l'entité agissante. La notion d'influence concrétise par ailleurs la *contrainte d'intégrité environnementale* : un agent ne doit pas modifier directement les variables d'état de l'environnement.

Par ailleurs, nous avons à plusieurs reprises insisté sur l'intérêt pratique et conceptuel de considérer qu'un agent est fondamentalement composé de deux parties, *le corps et l'esprit*, et nous avons vu que cette séparation est aussi une conséquence naturelle du principe Influence/Réaction. Son application nécessite en effet de faire une distinction nette entre les variables du système conatif, sur lesquelles l'agent a un contrôle total, et les variables qui font partie du monde physique et qui ne sont pas sous son contrôle. C'est pourquoi nous avons consacré un chapitre entier à ce principe et que nous avons proposé une adaptation du modèle de [Ferber & Müller, 1996] mieux adaptée à la simulation multi-agents. Modèle que nous avons concrétisé dans l'application *Warbot*.

L'utilisation du principe Influence/Réaction nous a aussi amené à reconsidérer la notion d'interaction dans la simulation multi-agents. Nous avons vu que celle-ci est rarement identifiée en tant que tel et que le traitement qui lui correspond est souvent dilué dans la globalité du modèle. Par conséquent, il est naturel que la modélisation de l'interaction soit un point d'ambiguïté majeur dans les spécifications des modèles multi-agents. Au contraire, dans le principe Influence/Réaction, le calcul du résultat des interactions produites à un instant t est

clairement associé à celui de la réaction de l'environnement à l'ensemble des influences. Raison pour laquelle nous avons proposé une définition contextuelle de l'interaction qui est relative à ce calcul.

La réflexion que nous avons menée sur la modélisation de l'interaction nous a aussi conduit à étudier la deuxième relation fondamentale de la théorie de la M&S : la *relation de modélisation*. Nous avons vu que cet aspect de la validation, qui concerne l'évaluation de la qualité du modèle relativement au cadre expérimental considéré, est une question excessivement difficile dans le contexte de la modélisation multi-agents. Notamment car elle repose le plus souvent sur des critères subjectifs du fait de la complexité des dynamiques qui sont mises en jeu.

Dans ce cadre, nous avons insisté sur la nécessité d'intégrer de nouveaux aspects de validation qui soient purement basés sur des critères objectifs. Nous avons ainsi soutenu l'idée que le paradigme agent définit un cadre expérimental particulier dont les contraintes de modélisation doivent être prises en compte. De cette idée, nous avons proposé un niveau de cohérence supplémentaire dans l'étude de la relation de simulation : la *cohérence paradigmatique*. Celle-ci concerne l'analyse de l'adéquation du paradigme de modélisation utilisé et le modèle qui en est issu.

A ce propos, nous avons focalisé notre attention sur la notion d'*autonomie* et nous avons donné un moyen objectif de vérifier que le principe d'autonomie est effectivement présent dans la modélisation. Il s'agit de respecter l'*intégrité interne* d'un agent. Nous avons ainsi défendu l'idée que lorsque cette contrainte est violée, le modèle ne respecte pas la cohérence paradigmatique et la relation de modélisation n'est alors pas vérifiée. A partir de cela, nous avons argumenté sur l'intérêt d'identifier différents types d'interaction lors de l'élaboration du modèle. Nous avons vu que celles que nous avons désignées sous les noms d'*interactions fortes* et d'*interactions faibles* ne posent pas les mêmes problèmes vis-à-vis de la manière dont elles doivent être modélisées relativement à la cohérence paradigmatique.

De ces différentes réflexions, nous avons conclu que le principe Influence/Réaction n'est pas uniquement un moyen élégant de résoudre le problème de la simultanéité, il répond aussi à un véritable besoin pour une modélisation satisfaisante des systèmes multi-agents. Nous avons ainsi soutenu l'idée que les différents principes de modélisation que nous avons identifiés tout au long de ce document (distinction esprit/corps, respect de l'autonomie, contraintes d'intégrités, etc.) implique de considérer les deux temps définis par l'application du principe Influence/Réaction : le temps de la délibération, c'est-à-dire la production des influences, et le temps de l'interaction où le problème est de trouver une solution à la combinaison de toutes les influences.

Cette différence fondamentale se trouve au cœur de l'approche formelle proposée par le modèle MIC*. Comme nous l'avons vu, MIC* fait effectivement une différence claire entre le temps du calcul et le temps de l'interaction. En fait, le principe Influence/Réaction est inhérent à MIC*. Par ailleurs, nous avons vu que ce modèle permet aussi naturellement de mettre en application les différents principes de modélisation que nous avons identifiés. Tout d'abord, il réifie les concepts de capteur et d'effecteur à un haut niveau d'abstraction en définissant respectivement la *inbox* et la *outbox* d'un agent. Par ailleurs, nous avons vu que la distinction esprit/corps ainsi que le respect des différentes contraintes d'intégrité sont aussi des propriétés intrinsèques du modèle MIC*, notamment grâce à l'utilisation de la notion d'*objet d'interaction*. Ajoutée à cela, la possibilité de pouvoir définir contextuellement des *lois d'interaction* permet de donner une véritable dimension au problème de la modélisation de l'interaction. Clairement identifiée comme une problématique à part entière, la formalisation de la gestion des interactions constitue le point essentiel de cette approche.

10.2 Perspectives de recherche

Finalement, le modèle MIC* nous a permis de concrétiser les différents points de notre analyse puis de les mettre en œuvre dans le cadre de la modélisation et de la simulation d'une société artificielle d'agents inspirée du modèle *Sugarscape*. C'est de cette expérience que nous tirons le plus de perspectives de travail à court terme. En premier lieu, il est évident que le modèle que nous avons proposé est assez complexe du fait de l'application du principe Influence/Réaction, du modèle MIC* et des différentes contraintes liées à la cohérence paradigmatique (distinction esprit/corps, intégrité environnementale, etc.). C'est pourquoi il conviendra de clarifier le plus possible les différents concepts qui sont manipulés lors d'une modélisation basée sur ces outils formels. La formalisation d'une loi d'interaction, d'une loi de computation et des différents objets d'interaction n'est pas forcément une tâche triviale en l'état. Il est certain qu'un effort d'encapsulation est nécessaire pour qu'un tel processus de modélisation puisse être généralisé et, surtout, reproduit à l'identique.

Une perspective de recherche importante reposera donc sur l'élaboration d'un formalisme qui puisse englober l'ensemble des notions que nous avons manipulées de façon à en diminuer la complexité autant que possible. En quelque sorte, il s'agit là d'effectuer un travail d'unification. Sachant que le modèle MIC* intègre naturellement le principe Influence/Réaction et les différentes solutions méthodologiques associées aux contraintes de modélisation que nous avons identifiées, nous avons bon espoir en ce qui concerne ce travail. C'est pourquoi l'un de nos premiers objectifs futurs sera d'élaborer une extension du modèle MIC* spécialement conçue pour la modélisation et la simulation. Il serait par exemple très intéressant de disposer d'une bibliothèque MIC* où les concepts primordiaux tels que l'influence et la perception seraient déjà présents. Pour cela, nous pensons pouvoir nous inspirer de la hiérarchie que nous avons présentée dans le chapitre précédent. A partir de là, il sera crucial d'élaborer une méthodologie de conception associée à un formalisme unifié. Le but sera de normaliser le processus de modélisation sous-tendu par notre démarche. Une autre perspective de travail associée à cet effort de formalisation sera d'étudier dans quelle mesure les formalismes que nous avons utilisés peuvent être traduits dans des structures DEVS par exemple.

A moyen terme, il nous faudra continuer d'explorer les possibilités de modélisation offertes par le principe Influence/Réaction. La notion d'influence possède en effet de multiples potentialités que nous n'avons pas eu le temps d'aborder. Elle permet par exemple de modéliser un système où les agents perçoivent le comportement des autres entités sans pour autant violer la contrainte d'intégrité interne comme nous l'avons vu dans la section 7.5.3 (page 152) . Une influence est en effet un moyen d'extérioriser dans l'environnement le comportement d'un agent, c'est-à-dire de le rendre perceptible par les autres. Dans le cas de l'interaction de reproduction, il serait par exemple intéressant qu'un agent puisse percevoir qu'une autre entité souhaite se reproduire avec lui, sans pour autant accéder à ses variables personnelles et/ou remettre en cause son autonomie décisionnelle. Nous pensons qu'il y a là un énorme champ d'investigation et qu'il est ainsi possible d'augmenter le pouvoir d'expressivité des modèles multi-agents.

A plus long terme, nous pensons qu'il sera possible de développer des outils de conception qui permettront d'automatiser la modélisation et la simulation d'un système multi-agents basé sur le principe Influence/Réaction et sur le modèle MIC*. Dans ce cadre, la principale difficulté sera sans aucun doute de ne pas sacrifier les fondements du paradigme agent par trop de simplification. Au contraire, le but sera de faire en sorte que ses fondements soient incontournables. Les contraintes qu'il définit doivent être prises en compte dans la modélisation. Le respect de l'autonomie décisionnelle, la modélisation explicite de l'interaction, la distinction

esprit/corps, l'intégrité environnementale sont autant de notions auxquelles il est aujourd'hui important d'associer explicitement le paradigme agent.

Il y a fort à parier que de plus en plus de systèmes complexes seront modélisés à partir d'une vision décentralisée du monde. Considérer que la dynamique globale d'un système peut être représentée par un ensemble d'entités autonomes qui interagissent localement les unes avec les autres constitue une approche dont la philosophie est extrêmement prometteuse. C'est pourquoi la simulation multi-agents a déjà prouvé qu'elle constituait une approche novatrice et l'intérêt conceptuel des principes de modélisation qui lui sont associés sont évidents. Il lui faut maintenant cependant passer à un stade supérieur : elle doit devenir une démarche scientifique plus rigoureuse et reproductible. Celle-ci ne doit plus être uniquement une façon de penser un système, mais un véritable paradigme de modélisation qui suit des principes clairement définis et invariants.

Annexe A

Le Jeu de la Vie en MIC*

A.1 Les objets d'interaction

A.1.1 `CellOwner` : représentation d'un agent dans l'espace d'interaction représentant sa cellule

```
public class CellOwner extends InteractionObject implements Sensor{

        public CellOwner(boolean state){
                super(new Boolean(state));  // true: alive cell
        }
        public boolean getState(){
            return ((Boolean) getContent()).booleanValue();
        }
// this IO inteacts with neighbor IOs to return a NeighborComputation as perception
        public InteractionObject interaction(InteractionObject o){
                if(o instanceof Neighbor)
                        return new NeighborComputation(((Neighbor) o).getState());
                return InteractionObject.zero;
        }
}
```

A.1.2 `Neighbor` : représentation d'un agent dans les cellules voisines

```
public class Neighbor extends NeighborComputation{

        public Neighbor(boolean state){
                super(state);
        }
}
```

A.1.3 `NeighborComputation` : perception du calcul de voisins

```
public class NeighborComputation extends InteractionObject{

        public NeighborComputation(boolean state){
                super(new Boolean(state));
        }

        public boolean getState(){
                return ((Boolean) getContent()).booleanValue();
        }
}
```

A.1.4 `CellAgentRepresentation` : représentation d'un agent dans l'espace d'interaction initial

```
public class CellAgentRepresentation extends InteractionObject implements Movable {

        private boolean state;

        public CellAgentRepresentation(int abs, int ord, boolean initState){
                super(new java.awt.Point(abs,ord));
                state=initState;
        }

        public boolean getInitState(){
                return state;
        }
}
```

A.1.5 `CellObserver` : représentation de l'agent chargé de l'affichage

```
public class CellObserver extends InteractionObject implements Sensor,Movable{

        public CellObserver(String name){
                super(name);
        }
        //getting system's information by interacting with it
        public InteractionObject interaction(InteractionObject o){
                if(o instanceof CellOwner)
                        return o;
                return InteractionObject.zero;
        }
}
```

A.2 Les espaces d'interaction

A.2.1 `StartingInteractionSpace` : espace d'interaction initial

Cet espace d'interaction expulse les objets d'interaction correspondant à la représentation initiale d'un agent pour qu'elle soit absorbée par les espaces d'interaction représentant les cellules de la grille lorsque la loi de mouvement initiale sera déclenchée.

```
public class StartingInteractionSpace extends InteractionSpace{

        public StartingInteractionSpace(){
                super("start");
        }
        //handling IOs that may come out
        public InteractionObject out(Movable o){
                if(o instanceof CellAgentRepresentation || o instanceof CellObserver)
                        return (InteractionObject)o;
                return InteractionObject.zero;
        }
}
```

A.2.2 `CellInteractionSpace` : espace d'interaction représentant une cellule

Cet espace d'interaction correspond à une cellule. Lors du mouvement initial, il absorbe les objets d'interaction correspondant à la représentation initiale d'un agent et les convertit dans un objet `Neighbor` ou `CellOwner` suivant le rôle que l'agent doit jouer dans cet espace.

```
public class CellInteractionSpace extends InteractionSpace{

        int gridSize;

        public CellInteractionSpace(int abs,int ord,int gridSize){
                super(new java.awt.Point(abs,ord),"<CIS "+abs+","+ord+">");
                this.gridSize = gridSize;
        }

public MoveInResult in(Movable o){//handling IOs that can come in
 if(o instanceof CellAgentRepresentation)//computing the result of the movement
 {
  CellAgentRepresentation agt = (CellAgentRepresentation) o ;
  int abs = ((java.awt.Point) agt.getContent()).x;
  int ord = ((java.awt.Point) agt.getContent()).y;
  int x = ((java.awt.Point) getId()).x;
  int y = ((java.awt.Point) getId()).y;
  if(abs == x && ord == y)
    return new MoveInResult(InteractionObject.zero,new CellOwner(agt.getInitState()));

  if( (x == 0 && abs == gridSize-1) && (ord == y || ord == y+1 || ord == y-1 || (ord ==
       gridSize-1 && y==0) || (ord == 0 && y == gridSize-1)))
   return new MoveInResult(InteractionObject.zero,new Neighbor(agt.getInitState()));
  if( (y == 0 && ord == gridSize-1) && (abs == x || abs == x+1 || abs == x-1 || (abs ==
       gridSize-1 && x==0) || (abs == 0 && x == gridSize-1)))
   return new MoveInResult(InteractionObject.zero,new Neighbor(agt.getInitState()));
  if( (x == gridSize-1 && abs == 0) && (ord == y || ord == y+1 || ord == y-1 || (ord ==
       gridSize-1 && y==0) || (ord == 0 && y == gridSize-1)))
   return new MoveInResult(InteractionObject.zero,new Neighbor(agt.getInitState()));
  if( (y == gridSize-1 && ord == 0) && (abs == x || abs == x+1 || abs == x-1 || (abs ==
       gridSize-1 && x==0) || (abs == 0 && x == gridSize-1)))
   return new MoveInResult(InteractionObject.zero,new Neighbor(agt.getInitState()));
  if( abs == (x+1) || abs == (x-1))
    if(ord == (y+1) || ord == (y-1) || ord == y)
     return new MoveInResult(InteractionObject.zero,new Neighbor(agt.getInitState()));
  if(ord == (y+1) || ord == (y-1) )
    if(abs == (x+1) || abs == (x-1) || abs == x)
     return new MoveInResult(InteractionObject.zero,new Neighbor(agt.getInitState()));
 }
 else if(o instanceof CellObserver)
        return new MoveInResult(InteractionObject.zero,new CellObserver("spy"));
 return new MoveInResult(InteractionObject.zero,InteractionObject.zero);
 }

 public InteractionObject out(Movable o){
        return InteractionObject.zero;
 }
}
```

A.3 Les processus de calcul

A.3.1 `CellAgent` : une cellule du Jeu de la vie

```
public class CellAgent{

private AgentPlugin myPlugin;
private boolean state;

public CellAgent(AgentPlugin micPlugin, boolean initState){
        myPlugin = micPlugin;
        state = initState;
}
public AgentPlugin getAgentPlugin(){
        return myPlugin;
}
```

```
public void compute(){
        Map inbox = myPlugin.calculateInbox();//getting perceptions
        InteractionObject result=null;
        for(Iterator i = inbox.values().iterator();i.hasNext();)//for each ISpace
        {
                result = (InteractionObjectPlus) i.next();
        }
        Vector v = new Vector();
        result.getListObject(v);
        v.remove(InteractionObject.zero);
        int aliveNeighbors = 0;
        //analyzing neighbors computations
        for(Iterator i=v.iterator();i.hasNext();)
                if( ((NeighborComputation) i.next()).getState())
                        aliveNeighbors++;
        if ( state && (aliveNeighbors < 2 || aliveNeighbors >3))
                state=false;
        else if (aliveNeighbors == 3)
                state=true;
        myPlugin.refreshLocalOutboxCopy();
        //updating my representations in each ISpace I belong to
        for(Iterator i=getAgentPlugin().localOutboxCopy.entrySet().iterator(); i.
            hasNext(); )
        {
                Map.Entry e = (Map.Entry) i.next();
                if(e.getValue() instanceof CellOwner)
                        e.setValue(new CellOwner(state));
                else
                        e.setValue(new Neighbor(state));
        }
        getAgentPlugin().publishComputation();
}
}
```

A.3.2 `Viewer` : agent destiné à la représentation graphique

Cet agent possède une représentation, `CellObserver`, dans chaque espace d'interaction qui lui permet de récupérer par interaction, dans sa inbox, tous les objets d'interaction de type `CellOwner`. Ce qui lui permet de réaliser un affichage de l'état de la grille.

```
public class Viewer{
   int cellSize,envWidth;
   GridCanvas onScreen;
   private AgentPlugin myPlugin;

public void compute(Graphics g){
 Map inbox = myPlugin.calculateInbox(); //getting perceptions
 for(Iterator i=inbox.entrySet().iterator(); i.hasNext(); )
 {
        Map.Entry e = (Map.Entry) i.next();
        if( ((CellOwner)e.getValue()).getState()) //displaying
                g.setColor(Color.red);
        else
                g.setColor(Color.black);
        g.fillRect( ((Point)e.getKey()).x*cellSize,((Point)e.getKey()).y*cellSize,
            cellSize,cellSize);
 }
}

public AgentPlugin getAgentPlugin(){
        return myPlugin;
}
}
```

A.4 Programme principal

A.4.1 `GameOfLife` : création du noyau MIC* et définition de la dynamique

```
public class GameOfLife{

        MICkernel mic; int gridSize; InteractionSpace[][] grid;
        CellAgent[][] agents; Viewer theViewer;

        /** Creates a new instance of the Game */
        public GameOfLife(){
        mic = MICBooter.kernel; //the MIC engine
        gridSize = 10;
        grid = new CellInteractionSpace[gridSize][gridSize]; // cell ISpaces
        agents = new CellAgent[gridSize][gridSize]; // Process
        InteractionSpace i0 = new StartingInteractionSpace();
        mic.addISpace(i0);
        theViewer = new Viewer(mic.addAgentPlugin("Viewer Agent", i0, new CellObserver(
            "spy")),30,gridSize);
        theViewer.setup();

        //creation
        for(int i=0;i<gridSize;i++)
                for(int j=0;j<gridSize;j++)
                {
                 agents[i][j] = new CellAgent(mic.addAgentPlugin("Cell Agent("+i+","+j+
                     ")", i0, new CellAgentRepresentation(i,j,rndState)),rnd.newBoolean
                     ());
                 grid[i][j] = new CellInteractionSpace(i,j,gridSize);
                 mic.addISpace(grid[i][j]);
                }

        //populating the grid using movement laws
        for(int x=0;x<gridSize;x++)
         for(int y=0;y<gridSize;y++)
         {
          mic.getScheduler().applyMovementLaw(new MovementLaw(i0,grid[x][y],theViewer.
              getAgentPlugin()));
          for(int i=0;i<gridSize;i++)
           for(int j=0;j<gridSize;j++)
            mic.getScheduler().applyMovementLaw(new MovementLaw(i0,grid[x][y],agents[i
                ][j].getAgentPlugin()));
         }
}

public void play() //interaction then computation forever
{
        int i = 0;
        while(true)
        {
                System.err.println("iteration "+(i++));
                for(int x=0;x<gridSize;x++)
                    for(int y=0;y<gridSize;y++)
                            mic.getScheduler().applyInteractionLaw(new InteractionLaw(
                                grid[x][y].getId(),agents[x][y].getAgentPlugin()));
                for(int x=0;x<gridSize;x++)
                        for(int y=0;y<gridSize;y++)
                                agents[x][y].compute();
                theViewer.display(); //calling the viewer computation

        }
}

public static void main(String[] args){ // java MIC* is still a prototype
        GameOfLife g = new GameOfLife();
        g.play();
}
}
```

Annexe B

SugarScape

B.1 Les objets d'interaction

B.1.1 L'ensemble $\mathcal{O}_{physical}$: représentation des objets physiques du monde

`AgentPhysicalBody` : représentation du corps d'un agent

```
public class AgentPhysicalBody extends InteractionObject implements Sensor,Comparable{

        //model related
        public int fov;; // the field of view (FOV) attribute
        public double mu; // the metabolic rate of resource consumption
        public boolean sex;     // sex : male == true
        public double birthDate;
        public double lifespan;
        public double puberty; // the age when reproductive capability begins
        public double reproEnd; // the age when reproductive capability ends
        public int gestation; // the gestation period (useless for males)

        public double energy; //the amount of resource an agent holds
        public double initialEnergy; //the amount of resource an agent holds
        public double age=0;

        //implementation related
        public double lastUpdateTime=-1;
        public double lastUpdateTimeForFertile=-1;
        public boolean fertile=false;
        public int x=0,y=0;
        public int agtNb;
        public SugarAgent myBrain;
        public boolean reproducing=false;

public AgentPhysicalBody(int agtNb, boolean sex, double birthDate, int fov, double mu,
    double lifespan, double puberty, double reproEnd, int gestation, int x, int y,
    double initialEnergy){
        //implementation related
        super("AgentBody");
        this.agtNb = agtNb;
        this.x=x;
        this.y=y;
        //model related
        this.sex=sex;
        this.fov=fov;
        this.mu=mu;
        this.lifespan=lifespan;
        this.puberty=puberty;
        this.reproEnd=reproEnd;
        this.gestation=gestation;
```

```
        this.birthDate=birthDate;
        this.initialEnergy=initialEnergy;
        energy=initialEnergy;
        lastUpdateTime=birthDate;
}

public InteractionObject interaction(InteractionObject o){//getting perceptions
        InteractionObject result = interaction(o,true);
        InteractionObject tmp = interaction(o,false);
        if(tmp!=null)
                if(result==null)
                        result = tmp;
                else result = new InteractionObjectPlus(tmp,result);
        return result;
}

public InteractionObject computeResourcePerceptions(InteractionObject o){
InteractionObject result = null;
if(o instanceof Landscape)
{
        Landscape l = (Landscape) o;
        double maxValue=0;
        Patch p;
        double tmpValue;
        for(int i=1;i<=fov;i++)
        {
                tmpValue = getResourceValue(l.getPatch(x+i,y));
                if(tmpValue>maxValue)
                {
                        result = new Resource(tmpValue,i,0);
                        maxValue=tmpValue;
                }
                else if(tmpValue==maxValue)
                        result = new InteractionObjectPlus(new Resource(tmpValue,i,0),
                            result);
                tmpValue = getResourceValue(l.getPatch(x,y+i));
                if(tmpValue>maxValue)
                {
                        result = new Resource(tmpValue,0,i);
                        maxValue=tmpValue;
                }
                else if(tmpValue==maxValue)
                        result = new InteractionObjectPlus(new Resource(tmpValue,0,i),
                            result);
                tmpValue = getResourceValue(l.getPatch(x-i,y));
                if(tmpValue>maxValue)
                {
                        result = new Resource(tmpValue,-i,0);
                        maxValue=tmpValue;
                }
                else if(tmpValue==maxValue)
                        result = new InteractionObjectPlus(new Resource(tmpValue,-i,0),
                            result);
                tmpValue = getResourceValue(l.getPatch(x,y-i));
                if(tmpValue>maxValue)
                {
                        result = new Resource(tmpValue,0,-i);
                        maxValue=tmpValue;
                }
                else if(tmpValue==maxValue)
                        result = new InteractionObjectPlus(new Resource(tmpValue,0,-i),
                            result);
        }
return result;
}

public double getResourceValue(Patch p){
        p.update(GenScheduler.GVT);
        if(p.occupant == null)
                return p.getAvailableResource();
```

```
}

public InteractionObject computeNeighborsPerceptions(InteractionObject o){
InteractionObject result = null;
if(o instanceof Landscape)
{
        Landscape l = (Landscape) o;
        if(fertile(l))
        {
                InteractionObject tmp = null;
                tmp = addNeighborPerception(x+1,y,l);
                if(tmp != null)
                        result = new InteractionObjectPlus(result,tmp);
                tmp = addNeighborPerception(x,y+1,l);
                if(tmp != null)
                        result = new InteractionObjectPlus(result,tmp);
                tmp = addNeighborPerception(x-1,y,l);
                if(tmp != null)
                        result = new InteractionObjectPlus(result,tmp);
                tmp = addNeighborPerception(x,y-1,l);
                if(tmp != null)
                        result = new InteractionObjectPlus(result,tmp);
        }
}
return result;
}

public InteractionObject addNeighborPerception(int a,int b,Landscape l){
        Patch p =l.getPatch(a,b);
        p.update(GenScheduler.GVT);
        AgentPhysicalBody potential = p.occupant;
        if(potential != null && sex != potential.sex && potential.fertile(l))
                return new FertileNeighborOfOppositeSex(potential);
        else
                return null;
}

public void consumeEnergy(double GVT){
        if(lastUpdateTime<GenScheduler.GVT)
        {
                energy-=mu*(GVT-lastUpdateTime);
                lastUpdateTime=GenScheduler.GVT;
        }
}

public boolean fertile(Landscape l){
        return ((!reproducing) && age>=puberty && age <= reproEnd && initialEnergy<
            energy+gestation*(l.grid[x][y].rho-mu) && isThereAnEmptyVonNeumannPatch(
            this,l));
}

public boolean isThereAnEmptyVonNeumannPatch(AgentPhysicalBody agt,Landscape l){
        if(isThisPatchEmpty(l.getPatch(agt.x+1,agt.y))
                return true;
        if(isThisPatchEmpty(l.getPatch(agt.x,agt.y+1))
                return true;
        if(isThisPatchEmpty(l.getPatch(agt.x-1,agt.y))
                return true;
        if(isThisPatchEmpty(l.getPatch(agt.x,agt.y-1))
                return true;
        return false;
}

public boolean isThisPatchEmpty(Patch p){
        p.update(GenScheduler.GVT);
        if(p.getOccupant() == null)
                return true;
}

public boolean update(double GVT,Patch p){
```

```
        if(lastUpdateTime<GVT)
        {
                age=GVT-birthDate;
                consumeEnergy(GVT);
                energy+=p.consumeAvailableResource();
                lastUpdateTime=GVT;
        }
        return true;
}

public double surviving(double deltaT,Patch p){//will the agent survive?
        double tmp = energy + deltaT*(p.rho-mu);
        if(tmp <= 0)
        {
         double death = -energy/(p.rho-mu);
         //System.err.println(toDebug()+"acting in "+deltaT+" must be dead in "+death);
         return death;
        }
        return -1;
}

public int compareTo(Object o){
AgentPhysicalBody a = (AgentPhysicalBody) o; //crash is good here for debugging
if(a.agtNb > agtNb)
        return -1;
else if(a.agtNb < agtNb)
        return 1;
sugarscape.util.Debug.systemDebug("----------BUG BUG-bug on compare between agent");
return 0;
}
}
```

`Landscape` : représentation de l'environnement physique

```
public class Landscape extends InteractionObject implements Sensor{
        public int x,y;//=Model.defaultAgentFOV;// the field of view (FOV) attribute
        public PRNGInterface randomSource;

        //internal structure
        public Patch grid[][] = null;
        public List agentsList,birthInfluences;
        public SugarScapeModel model;

        public Landscape(int width, int height, PRNGInterface randomSource,
            SugarScapeModel model){
                super("Land");
                this.x = width;
                this.y = height;
                this.randomSource = randomSource;
                grid = new Patch[x][y];
                agentsList = new ArrayList(SugarScapeModel.agtNbEstimation);
                birthInfluences= new ArrayList(SugarScapeModel.agtNbEstimation);
                this.model= model;
                initGrid();
        }

        public InteractionObject interaction(InteractionObject o){
                return o;
        }

// interaction related
public void handleMovement(Movement m){
        if(! agentsList.contains(m.from))
                return;
        AgentPhysicalBody agt = m.from;
        Patch p = getPatch(agt.x+m.dx,agt.y+m.dy);
        if(p.getOccupant()==null)
```

```
                moveAgentTo(agt,p);
        else
                if(m.dy==0)
                        if(m.dx>0)
                                for(int i=m.dx-1;i>0;i--){
                                        int delta = normeX(agt.x+i);
                                        if(grid[delta][agt.y].getOccupant()==null)
                                        {
                                                moveAgentTo(agt,delta,agt.y);
                                                return;
                                        }
                                }

                        else
                                for(int i=m.dx+1;i<0;i++){
                                        int delta = normeX(agt.x+i);
                                        if(grid[delta][agt.y].getOccupant()==null)
                                        {
                                                moveAgentTo(agt,delta,agt.y);
                                                return;
                                        }
                                }
                else
                        if(m.dy>0)
                                for(int i=m.dy-1;i>0;i--){
                                        int delta = normeY(agt.y+i);
                                        if(grid[agt.x][delta].getOccupant()==null)
                                        {
                                                moveAgentTo(agt,agt.x,delta);
                                                return;
                                        }
                                }

                        else
                                for(int i=m.dy+1;i<0;i++){
                                        int delta = normeY(agt.y+i);
                                        if(grid[agt.x][delta].getOccupant()==null)
                                        {
                                                moveAgentTo(agt,agt.x,delta);
                                                return;
                                        }
                                }
}

public void computeMates(List reproductionInfluences){
        ReproductionAttempt tmp=null;
        if(reproductionInfluences.size()>1)
                tmp = (ReproductionAttempt) reproductionInfluences.remove(0);
        else
                return;
        for(Iterator i=reproductionInfluences.iterator();i.hasNext();)
        {
                ReproductionSuccess io = tmp.mating((ReproductionAttempt)i.next());
                if(io != null)
                {
                        LandScapeAgent.reproSuccess++;
                        io.parent2.getBrain().clearReproOutbox();
                        io.parent1.getBrain().clearReproOutbox();
                        birthInfluences.add(io);
                        if(model.gestation!=0 && ! model.discreteTime)
                                model.getScheduler().addEventInDeltaT(new ObjectEvent(
                                    this,GenScheduler.GVT+model.gestation,"manageBirths
                                    "));
                        i.remove();
                        break;
                }
        }
        computeMates(reproductionInfluences);
        if(model.gestation==0)
                manageBirths();
```

```
}

public boolean computeEventMates(Map reproductions,ReproductionAttempt r1){
        for(Iterator i = reproductions.values().iterator();i.hasNext();)
        {
                ReproductionSuccess io = r1.mating((ReproductionAttempt)i.next());
                if(io != null)
                {
                        i.remove();
                        if(! model.discreteTime)
                        {
                                model.activator6.removeAgent(io.parent1.getBrain());
                                model.activator6.removeAgent(io.parent2.getBrain());
                        }
                        if(model.gestation!=0)
                        {
                                birthInfluences.add(io);
                                model.getScheduler().addEventInDeltaT(new ObjectEvent(
                                    this,GenScheduler.GVT+model.gestation,"manageBirths
                                    "));
                        }
                        else
                                validateBirth(io);
                        return true;
                }
        }
        return false;
}

public void manageBirths(){
        Collections.sort(birthInfluences); // to be determinist : starting from a
            special ordered list before shuffle
        if(randomSource instanceof Random)
                Collections.shuffle(birthInfluences,(Random)randomSource); // to be
                    determinist
        else
                MyCollections.shuffle(birthInfluences,randomSource);
        for(Iterator i=birthInfluences.iterator();i.hasNext();)
        {
                ReproductionSuccess io = (ReproductionSuccess)i.next();
                if(GenScheduler.GVT==io.getBirthDate())
                {
                        i.remove();
                        validateBirth(io);
                }
        }
}

public void validateBirth(ReproductionSuccess rs){
        rs.parent1.energy-=(rs.parent1.initialEnergy/2);
        rs.parent2.energy-=(rs.parent2.initialEnergy/2);
        if(rs.parent1.energy<0 ||rs.parent1.energy<0 || ! rs.parent1.reproducing || !
            rs.parent2.reproducing)
                sugarscape.util.Debug.systemDebug("----------------BUG BUG
                    ----------------------------"+rs.toDebug());
        rs.parent1.reproducing=false;
        rs.parent2.reproducing=false;
        List potentials = findEmptyVonNeumannPatches(rs.parent1);
        potentials.addAll(findEmptyVonNeumannPatches(rs.parent2));
        Patch target=null;
        double maxValue=-1;
        if(randomSource instanceof Random)
                Collections.shuffle(potentials,(Random)randomSource); // to be
                    determinist
        else
                MyCollections.shuffle(potentials,randomSource);
        for(Iterator j=potentials.iterator();j.hasNext();){
                Patch potential = (Patch) j.next();
                if(potential.getAvailableResource()>maxValue)
                {
```

```
                        target=potential;
                        maxValue=target.getAvailableResource();
                }
        }
        if(target!=null)
                model.createNewAgent(rs.parent1,rs.parent2,target.x,target.y);
        else
                if(model.activator6!=null)
                {
                        model.activator6.addAgent((EventGenerator)rs.parent1.getBrain()
                            );
                        model.activator6.addAgent((EventGenerator)rs.parent2.getBrain()
                            );
                }
}

public List findEmptyVonNeumannPatches(AgentPhysicalBody agt){ //first max in
    trigonometric order
        List patches = new ArrayList();
        Patch potential = getPatch(agt.x+1,agt.y);
        if(potential .getOccupant() == null)
                patches.add(potential);
        potential = getPatch(agt.x,agt.y+1);
        if(potential .getOccupant() == null)
                patches.add(potential);
        potential = getPatch(agt.x-1,agt.y);
        if(potential .getOccupant() == null)
                patches.add(potential);
        potential = getPatch(agt.x,agt.y-1);
        if(potential .getOccupant() == null)
                patches.add(potential);
        return patches;
}

public void agentsConsumeResource(){
        for(Iterator i = agentsList.iterator();i.hasNext();){
                AgentPhysicalBody agt = (AgentPhysicalBody) i.next();
                agt.energy+=grid[agt.x][agt.y].consumeAvailableResource();
        }
}

public void updateLandscape(){
        for (int i=0; i < x; i++)
            for (int j=0; j < y; j++)
                    grid[i][j].regrowth(GenScheduler.GVT);
}

public void bringOutUrDeads(double GVT){
        for(Iterator i = agentsList.iterator();i.hasNext();)
        {
                AgentPhysicalBody agt = (AgentPhysicalBody) i.next();
                agt.consumeEnergy(GenScheduler.GVT);
                if(agt.energy<=0)
                {
                        i.remove();
                        grid[agt.x][agt.y].removeAgent(agt);
                        model.killSugarAgent(agt);
                        continue;
                }
                agt.age=GVT-agt.birthDate;
                if(agt.age>=agt.lifespan)
                {
                        i.remove();
                        grid[agt.x][agt.y].removeAgent(agt);
                        model.killSugarAgent(agt);
                }
        }
}

final void initGrid(){
```

```
for (int i=0; i < x; i++)
    for (int j=0; j < y; j++){
            grid[i][j] = new Patch(initialCapacity(i,j),SugarScapeModel.RHO,i,j,this);
        }
}

//function used by L-P http://www.cs.wm.edu/~bglaws/research/model.html
public double f(int a,int b){
        return SugarScapeModel.MaxPatchCapacity * Math.exp(- Math.pow(a / (0.3*x),2) -
            Math.pow(b / (0.3*y),2));
}

public double initialCapacity(int a, int b){
        return f(a - x/4,b -y/4)+ f(a - 3*x/4,b -3*y/4);
}

public double computeSurviving(double deltaT,AgentPhysicalBody agt){
        double tmp = agt.surviving(deltaT,grid[agt.x][agt.y]);
        if(tmp>0)
                model.scheduleDeath(GenScheduler.GVT+deltaT,agt);
        return tmp;
}

}
```

B.1.2 L'ensemble $\mathcal{O}_{influences}$: modélisation des influences

`Influence` : super classe des influences

```
public class Influence extends InteractionObject implements Comparable{
        int from;

        public Influence(int agtNb){
                setContent("influence");
                from = agtNb;
        }

        public int compareTo(Object o){
                Influence i = (Influence) o; //crash is good here for debugging purpuse
                if(i.from > from)
                        return -1;
                else if(i.from < from)
                        return 1;
                sugarscape.util.Debug.systemDebug("BUG ----on compare"+this+" with "+o);
                return 0;
        }
}
```

`Movement` : influence de mouvement

```
public class Movement extends Influence{

        public int dx,dy;
        public AgentPhysicalBody from;

        public Movement(AgentPhysicalBody who,Resource r){
                super(who.agtNb);
                from = who;
                dx = r.x;
                dy = r.y;
        }
}
```

`ReproductionAttempt` : tentative de reproduction

```
public class ReproductionAttempt extends Influence{

        public AgentPhysicalBody from;
        public AgentPhysicalBody target;

        public ReproductionAttempt(AgentPhysicalBody who,FertileNeighborOfOppositeSex
            target){
                super(who.agtNb);
                from = who;
                this.target = target.who;
        }

        // computing the result of a strong interaction
        public ReproductionSuccess mating(ReproductionAttempt other){
                if(other.from == target && other.target==from)
                        if(from.agtNb>target.agtNb)      // to be determinist
                                return new ReproductionSuccess(target,from);
                        else
                                return new ReproductionSuccess(from,target);
                else
                        return null;
        }
}
```

`ReproductionSuccess` : validation de deux tentatives de reproduction

```
public class ReproductionSuccess extends Influence{

        public AgentPhysicalBody parent1,parent2;
        public double birthDate;

        public ReproductionSuccess(AgentPhysicalBody parent1,AgentPhysicalBody parent2)
            {
                super(parent1.agtNb);
                this.parent1=parent1;
                this.parent2=parent2;
                parent1.reproducing=true;
                parent2.reproducing=true;
                birthDate=(GenScheduler.GVT+parent1.gestation);
        }

        public double getBirthDate(){
                return birthDate;
        }
}
```

B.1.3 L'ensemble $\mathcal{O}_{perceptions}$: modélisation des perceptions

`Perception` : super classe des perceptions

```
public class Perception extends InteractionObject{

        public Perception(){
                setContent("percept");
        }
}
```

`FertileNeighborOfOppositeSex` : perception d'un partenaire potentiel

```
public class FertileNeighborOfOppositeSex extends Perception implements Comparable{
        public AgentPhysicalBody who;

        public FertileNeighborOfOppositeSex(AgentPhysicalBody agt){
                who = agt;
        }

        public double getEnergy(){
                return who.energy;
        }

        public int compareTo(Object o){//to sort
                FertileNeighborOfOppositeSex f = (FertileNeighborOfOppositeSex) o;
                if(f.who.agtNb > who.agtNb)
                        return -1;
                else if(f.who.agtNb < who.agtNb)
                        return 1;
                sugarscape.util.Debug.systemDebug("BUG ----on compare");
                return 0;
        }

}
```

`Resource` : perception de la ressource disponible dans une cellule

```
public class Resource extends Perception implements Comparable{
        public int x,y;
        public double resource;

        public Resource(double value,int x,int y){
                resource=value;
                this.x=x;
                this.y=y;
        }

        public double getAvailableResource(){
                return resource;
        }

        public int compareTo(Object o){
                Resource r = (Resource) o;
                if(Math.abs(x)<Math.abs(r.x) || Math.abs(y)<Math.abs(r.y))
                        return -1;
                if(x>r.x)
                        return -1;
                if(y>r.y)
                        return -1;
                return 1;
        }
}
```

B.2 Les processus de calcul

B.2.1 Les agents simulés : `MicAgent` et sa sous-classe `SugarAgent`

```
public class MicAgent extends GenAgent{//extends a MadKit agent class
        //mic and implementation related
        private AgentPlugin myPlugin;
        private AgentPhysicalBody body;
        private Object movesInfluenceSpace,landscape,reproductionInfluenceSpace;
```

```
        protected List resourcesPerception;
        protected List neighborsPerception;
        public InteractionObject perceptions;
        public PRNGInterface randomSource;

        public MicAgent(String community,PRNGInterface source,Object
            movesInfluenceSpaceId, Object landscapeSpaceId, Object
            reproductionInfluenceSpaceId, AgentPhysicalBody body){
                super(community,source);
                movesInfluenceSpace = movesInfluenceSpaceId;
                reproductionInfluenceSpace = reproductionInfluenceSpaceId;
                landscape = landscapeSpaceId;
                this.body = body;
                InteractionObject perceptions;
                randomSource = source;
        }

public void setAgentPlugin(AgentPlugin plugin){ myPlugin = plugin;}
public AgentPlugin getAgentPlugin(){return myPlugin;}
public AgentPhysicalBody getBody(){return body;}

protected void produceMoveInfluence(Resource target){
        if(target != null)
                getAgentPlugin().setOutBox(movesInfluenceSpace,new Movement(body,target
                    ));
}

protected void produceReproductionInfluence(FertileNeighborOfOppositeSex target){
        if(target != null)
                getAgentPlugin().setOutBox(reproductionInfluenceSpace,new
                    ReproductionAttempt(body,target));
}

public void updateReproPerceptions(){
        neighborsPerception = new ArrayList();

        if(perceptions != null)
                if(perceptions instanceof InteractionObjectPlus)
                {
                        InteractionObjectPlus tmp = (InteractionObjectPlus) perceptions
                            ;
                        List v = new ArrayList();
                        tmp.getListObject(v);
                        for(Iterator i = v.iterator();i.hasNext();)
                        {
                                InteractionObject io = (InteractionObject) i.next();
                                if(io instanceof FertileNeighborOfOppositeSex)
                                        neighborsPerception.add(io);
                        }
                }
                else
                        if(perceptions instanceof FertileNeighborOfOppositeSex)
                                neighborsPerception.add(perceptions);
}

public void updateMovePerceptions(){
        resourcesPerception = new ArrayList();
        if(perceptions != null)
                if(perceptions instanceof InteractionObjectPlus)
                {
                        InteractionObjectPlus tmp = (InteractionObjectPlus) perceptions
                            ;
                        List v = new ArrayList();
                        tmp.getListObject(v);
                        for(Iterator i = v.iterator();i.hasNext();)
                        {
                                InteractionObject io = (InteractionObject) i.next();
                                if(io instanceof Resource)
                                        resourcesPerception.add(io);
                        }
```

```
                }
                else
                        if(perceptions instanceof Resource)
                                resourcesPerception.add(perceptions);
}

public void clearOutbox(){
        MICBooter.kernel.getOutboxMatrix().set(getAgentPlugin(),movesInfluenceSpace,
            null);
        MICBooter.kernel.getOutboxMatrix().set(getAgentPlugin(),
            reproductionInfluenceSpace,null);
}

public void clearMoveOutbox(){
        MICBooter.kernel.getOutboxMatrix().set(getAgentPlugin(),movesInfluenceSpace,
            null);
}

public void clearReproOutbox(){
        MICBooter.kernel.getOutboxMatrix().set(getAgentPlugin(),
            reproductionInfluenceSpace,null);
}

}
```

```
public class SugarAgent extends MicAgent implements EventGenerator,Comparable{
        public boolean last;
        public double actionRate=2;

        public SugarAgent(String community,PRNGInterface source,Object
            movesInfluenceSpaceId, Object landscapeSpaceId, Object
            reproductionInfluenceSpaceId, AgentPhysicalBody body){
                super(community,source,movesInfluenceSpaceId, landscapeSpaceId,
                    reproductionInfluenceSpaceId,body);
                last = true;
        }

public double nextEventDeltaTimeFor(String behavior){
        return -Math.log(randomSource.nextDouble())/actionRate; // Poisson law
}

public InteractionObject moveEvent(){
        updateMovePerceptions();
        if(!resourcesPerception.isEmpty())
                return new Movement(getBody(),(Resource)resourcesPerception.get(
                    randomSource.nextInt(resourcesPerception.size())));
        return null;
}

public InteractionObject reproduceEvent(){
        updateReproPerceptions();
        double maxValue=0;
        FertileNeighborOfOppositeSex target=null;

        ArrayList potentials = new ArrayList();
        for(Iterator i=neighborsPerception.iterator();i.hasNext();)
        {
                FertileNeighborOfOppositeSex agt = (FertileNeighborOfOppositeSex) i.
                    next();
                if(agt.getEnergy()>=maxValue)
                {
                        if(agt.getEnergy()==maxValue)
                                potentials.add(agt);
                        else
                        {
                                maxValue=agt.getEnergy();
                                potentials.clear();
                                potentials.add(agt);
                        }
```

```
                }
        }
        if(potentials.size()>1)
                target=(FertileNeighborOfOppositeSex) potentials.get(randomSource.
                    nextInt(potentials.size()));
        else if (potentials.size()>0)
                target=(FertileNeighborOfOppositeSex) potentials.get(0);
        if(target != null)
        {
                if(SugarScapeModel.debug) System.err.println("\nrepro with--"+target);
                return new ReproductionAttempt(getBody(),target);
        }
        return null;
}

public int compareTo(Object o){
        return getBody().compareTo(((SugarAgent) o).getBody());
}

}
```

B.2.2 Le processus environnement : `LandScapeAgent`

```
public class LandScapeAgent extends GenAgent implements EventGenerator{
        private AgentPlugin myPlugin;
        private Object movesInfluenceSpace,landscape,reproductionInfluenceSpace;
        public List movementInfluences;
        public List reproductionInfluences;
        public Landscape myRepresentation;
        public Map reproductionsMap;
        public int attempts;
        public static int reproSuccess;

        public LandScapeAgent(String community,PRNGInterface source,Object
            movesInfluenceSpaceId, Object reproductionInfluenceSpaceId, Object
            landscapeSpaceId,Landscape world)
        {
                super(community,source);
                movesInfluenceSpace = movesInfluenceSpaceId;
                landscape = landscapeSpaceId;
                reproductionInfluenceSpace=reproductionInfluenceSpaceId;
                myRepresentation = world;
                movementInfluences = new ArrayList();
                reproductionInfluences = new ArrayList();
                reproductionsMap = new HashMap(SugarScapeModel.agtNbEstimation);
        }

    public void activate(){
        requestRole(community, GenModel.ENGINE_GROUP, GenModel.ENVIRONMENT_ROLE,null);
        requestRole(community, GenModel.MODEL_GROUP, GenModel.ENVIRONMENT_ROLE,null);
        requestRole(community, GenModel.MODEL_GROUP, "LandScapeAgent",null);
    }

public void updatePerceptions(){
        reproductionInfluences = new ArrayList();
        Map m=getAgentPlugin().calculateInbox();
        InteractionObject io1;
        movementInfluences = new ArrayList();
        io1 = (InteractionObject) m.get(movesInfluenceSpace);
        if(io1 != null)
                if(io1 instanceof InteractionObjectPlus)
                {
                        InteractionObjectPlus tmp = (InteractionObjectPlus) io1;
                        List v = new ArrayList();
                        tmp.getListObject(v);
                        for(Iterator i = v.iterator();i.hasNext();)
                        {
```

```
                                InteractionObject io = (InteractionObject) i.next();
                                if(io instanceof Movement)
                                        movementInfluences.add(io);
                        }
                }
                else
                        if(io1 instanceof Movement)
                                movementInfluences.add(io1);
        io1 = (InteractionObject) m.get(reproductionInfluenceSpace);
        if(io1 != null)
                if(io1 instanceof InteractionObjectPlus)
                {
                        InteractionObjectPlus tmp = (InteractionObjectPlus) io1;
                        List v = new ArrayList();
                        tmp.getListObject(v);
                        for(Iterator i = v.iterator();i.hasNext();)
                        {
                                InteractionObject io = (InteractionObject) i.next();
                                if(io instanceof ReproductionAttempt)
                                        reproductionInfluences.add(io);
                        }
                }
                else
                        if(io1 instanceof ReproductionAttempt)
                                reproductionInfluences.add(io1);
}
public void validateNewSystemState(){
        updateLandscape();
        myRepresentation.bringOutUrDeads(GenScheduler.GVT);
        manageBirths();
}
public void computeReaction(){
        computeReactionToReproductionAttempts();
        agentsConsuming();
}
public void computeEventBirthReaction(ReproductionAttempt ra,SugarAgent agt){
        if(! myRepresentation.computeEventMates(reproductionsMap,ra)){
                reproductionsMap.put(agt,ra);
        }
}
public void computeEventMove(Movement move){
        myRepresentation.handleMovement(move);
}
public boolean updateAgent(AgentPhysicalBody agt){
        return myRepresentation.updateAgent(agt);
}
public void updateLandscape(){
        myRepresentation.updateLandscape();
}
public String printSystemState(){
        return myRepresentation.printSystemState();
}
public void agentsConsuming(){
        myRepresentation.agentsConsumeResource();
}
public void computeReactionToMovements(){
        Collections.sort(movementInfluences); // to be determinist
        if(randomSource instanceof Random)
                Collections.shuffle(movementInfluences,(Random)randomSource);
        else
                MyCollections.shuffle(movementInfluences,randomSource);
        for(Iterator i=movementInfluences.iterator();i.hasNext();)
        {
                Movement m = (Movement) i.next();
                myRepresentation.handleMovement(m);
                m.from.getBrain().clearOutbox();
        }
}
public void computeReactionToReproductionAttempts(){
        if(! reproductionsMap.isEmpty())
```

```
                myRepresentation.computeMates(new ArrayList(reproductionsMap.values()))
                    ;
        reproductionsMap.clear();
}
public double agentSurviving(double deltaT,AgentPhysicalBody agt){
        return myRepresentation.computeSurviving(deltaT,agt);
}

}
```

B.3 Le moteur de simulation

B.3.1 L'agent *model* : SugarScapeModel

```
public class SugarScapeModel extends GenModel{
        //model related
        public int minFOV = 1, maxFOV=6;// the field of view (FOV) attribute
        public double minMU=1,maxMU=4;//defaultAgentMU // the metabolic rate of
            resource consumption
        public double minLAMBDA=60,maxLAMBDA=100;//=Model.defaultAgentLifespan;
        public double minALPHA=12,maxALPHA=15;//=Model.defaultAgentPuberty; // the age
            when reproductive capability begins
        public double minMaleOMEGA=50,maxMaleOMEGA=60,minFemaleOMEGA=40,maxFemaleOMEGA
            =50;// the age when reproductive capability ends
        public static int gestation=0;//=Model.defaultAgentGestation; // the gestation
            period (useless for males)
        public static int agentsInitialEnergy=10;

        public static int MaxPatchCapacity=4;
        public static double RHO=1; //regrowth rate


        public int squareSize;
        //implementation related
        long randomSeed;

        public int initialAgentsNb;//5
        public static int totalAgentsNb=0;
        public static boolean debug = false;
        public static int agtNbEstimation; //optimization parameter for large scale
            simulations
        boolean graphicMode=true;

        List modelAndEngineAgents;
        public DiscreteTimeActivator activator2,activator3,activator5;
        public AgentsInfluencesDiscreteTimeActivator activator1,activator4;
        public AgentsMoveReproDiscreteTimeActivator activator7;
        public AgentsInfluencesDiscreteEventActivator activator6;
        public RandomBehaviorDiscreteTimeActivator activator8;
        Landscape land;
        LandScapeAgent envAgent;

        //MIC varibales part
        InteractionSpace world, moveInfluencesSpace,reproductionInfluenceSpace;
        public MICkernel mic;
        InteractionLawForPerception perceptionLaw;
        InteractionLawForReactionComputation reactionLaw;
        public static boolean discreteTime;

        Map deaths = new HashMap(agtNbEstimation);
        public double lastUpdateBirths=0;
        public static int birthsCount=0;
        //List recycleBin;
        long tempSeed;
```

```
        public SugarScapeModel(String community, GenScheduler theScheduler, long seed,
             boolean graphicMode,int envSize,int initialAgentsNb,boolean dynamic,
            PRNGInterface randomSource)
        {
                super(community, theScheduler, randomSource);
                randomSeed = seed;
                agtNbEstimation=(int) ((envSize*envSize)/1.6);
                modelAndEngineAgents = new ArrayList();
                this.graphicMode=graphicMode;
                this.initialAgentsNb=initialAgentsNb;
                squareSize = envSize;
                discreteTime=dynamic;
                tempSeed=seed;
        }

public void buildModel()
{
        //if(biasbuildModel())  return;
        if(debug)
                sugarscape.util.Debug.setDebugMode(1);

        System.err.println("building model");
        System.err.println("building mic engine ");
        initMIC();
        initEnvironmentAgent();

        if(discreteTime)
        {
                //creating dynamic
                activator3 = new InitializeInfluenceReactionDiscreteTimeActivator(
                    community,GenModel.MODEL_GROUP,"LandScapeAgent","
                    validateNewSystemState",1,myScheduler,0.5,0,getRandomSource());
                activator1 = new AgentsInfluencesDiscreteTimeActivator(community,
                    GenModel.MODEL_GROUP,GenModel.SIMAGENT_ROLE,"move",3,myScheduler
                    ,1,0,reactionLaw,perceptionLaw,land.getAgentsList(),envAgent,
                    getRandomSource());
                activator4 = new AgentsInfluencesDiscreteTimeActivator(community,
                    GenModel.MODEL_GROUP,GenModel.SIMAGENT_ROLE,"reproduce",3,
                    myScheduler,1,0.5,reactionLaw,perceptionLaw,land.getAgentsList(),
                    envAgent,getRandomSource());
                activator2 = new EnvironmentReactionDiscreteTimeActivator(community,
                    GenModel.MODEL_GROUP,"LandScapeAgent","computeReaction",5,
                    myScheduler,0.5,0,reactionLaw,getRandomSource());
                addActivatorToSim(activator1);
                addActivatorToSim(activator2);
                addActivatorToSim(activator3);
                addActivatorToSim(activator4);
                activator6 = null;
        }
        else
        {
                activator6 = new AgentsInfluencesDiscreteEventActivator(community,
                    GenModel.MODEL_GROUP,GenModel.SIMAGENT_ROLE,"compute",3,myScheduler
                    ,reactionLaw,perceptionLaw,envAgent);
                addActivatorToSim(activator6); //using discrete event simulation this
                    time. Can be switched during runtime. can run both at same time
                    also
                activator6.initialize(land.agentsList);
        }
        activator5 = new DiscreteTimeActivator(community,GenModel.ENGINE_GROUP,"agents
            observer","observe",-1,myScheduler,1,0,getRandomSource());
        addActivatorToSim(activator5);

        for(int i =1;i<=initialAgentsNb;i++)
                createNewAgent();
        System.err.println("TIME is "+GenScheduler.GVT+"------init-"+getRandomSource()+
            "---"+discreteTime+"---"+randomSeed);
}

public void initMIC()
```

```
{
        mickernel.kernel.Debug.mode=0;
        MICBooter.kernel=new MICkernel();
        mic = MICBooter.kernel;
        // creating ISpaces
        world = new InteractionSpace("world");
        mic.addISpace(world);
        moveInfluencesSpace = new InteractionSpace("moves");
        reproductionInfluenceSpace = new InteractionSpace("repro");
        mic.addISpace(moveInfluencesSpace);
        mic.addISpace(reproductionInfluenceSpace);
}

public void initNewAgent(boolean sex, double birthDate, int fov, double mu, double
    lambda, double alpha, double omega, int gestation, int x, int y,double
    initialEnergy)
{
        totalAgentsNb++;
        SugarAgent a;
        AgentPhysicalBody body = new AgentPhysicalBody(totalAgentsNb,sex,birthDate,fov,
            mu,lambda,alpha,omega,gestation,x,y,initialEnergy);
        a = new SugarAgent(community,getRandomSource(),moveInfluencesSpace,world,
            reproductionInfluenceSpace, body);
        body.setBrain(a);
        if(activator6!=null)
        {
                myScheduler.addEventInDeltaT(new ObjectEvent(this,birthDate+body.
                    lifespan,"killDead"));
                deaths.put(new Double(birthDate+body.lifespan),a);
                activator6.addAgent((EventGenerator)a);
        }
        land.addAgent(a.getBody());
}

public void initEnvironmentAgent()
{
        land = new Landscape(squareSize,squareSize,getRandomSource(),this);
        AgentPlugin plugin = mic.addAgentPlugin("0", world, land);
        mic.getOutboxMatrix().set(plugin,moveInfluencesSpace,land);
        mic.getOutboxMatrix().set(plugin,reproductionInfluenceSpace,land);
        envAgent = new LandScapeAgent(community,getRandomSource(),moveInfluencesSpace.
            getId(),reproductionInfluenceSpace.getId(),world.getId(),land);
        envAgent.setAgentPlugin(plugin);
        launchAgent(envAgent,"Environment",false);
        Viewer v = new Viewer(getCommunity(),land.grid,randomSeed);
            modelAndEngineAgents.add(v);
        if(graphicMode)
        {
                launchAgent(v,getCommunity()+" viewer",true);
        }
        AgentsNbObserver obs = new AgentsNbObserver(getCommunity(),randomSeed,land.
            getAgentsList());     modelAndEngineAgents.add(obs); launchAgent(obs,"
            observer",false);
        modelAndEngineAgents.add(envAgent);
        perceptionLaw = new InteractionLawForPerception(world,land) ;
        reactionLaw = new InteractionLawForReactionComputation(moveInfluencesSpace,
            reproductionInfluenceSpace,envAgent);
}

public void killSugarAgent(AgentPhysicalBody agt)
{
        agt.reproducing=true;
        if(envAgent.reproductionsMap!=null)
                envAgent.reproductionsMap.remove(agt.getBrain());
        if(activator6 != null)
                activator6.removeAgent(agt.getBrain());
}

public void createNewAgent()
{
```

```
        boolean sex=getRandomSource().nextBoolean();
        int fov = getRandomSource().nextInt(5)+1;
        double mu = getRandomSource().nextDouble()*(maxMU-minMU)+minMU;
        double lambda = getRandomSource().nextDouble()*(maxLAMBDA-minLAMBDA)+minLAMBDA;
        double alpha = getRandomSource().nextDouble()*(maxALPHA-minALPHA)+minALPHA;
        double omega;
        if(sex)
                omega = getRandomSource().nextDouble()*(maxMaleOMEGA-minMaleOMEGA)+
                    minMaleOMEGA;
        else
                omega = getRandomSource().nextDouble()*(maxFemaleOMEGA-minFemaleOMEGA)+
                    minFemaleOMEGA;
        int Xlocation;
        int Ylocation;
        do
        {
                Xlocation = getRandomSource().nextInt(squareSize);
                Ylocation = getRandomSource().nextInt(squareSize);
        }
        while(land.grid[Xlocation][Ylocation].getOccupant()!=null);
        double birthDate=getScheduler().getGVT();
        initNewAgent(sex,birthDate,fov,mu,lambda,alpha,omega,gestation,Xlocation,
            Ylocation,agentsInitialEnergy);
}

public void createNewAgent(AgentPhysicalBody parent1, AgentPhysicalBody parent2,int
    Xlocation,int Ylocation)
{
        boolean sex=getRandomSource().nextBoolean();
        double omega;
        if(sex)
                if(parent1.sex)
                        omega = parent1.reproEnd;
                else
                        omega = parent2.reproEnd;
        else
                if(parent1.sex)
                        omega = parent2.reproEnd;
                else
                        omega = parent1.reproEnd;
        double mu;
        double lambda;
        double alpha;
        int fov;

        boolean coinFlip = getRandomSource().nextBoolean();
        AgentPhysicalBody selectedParent;

        if(coinFlip)
                selectedParent = parent1;
        else
                selectedParent = parent2;

       mu = selectedParent.mu;
       lambda = selectedParent.lifespan;
       alpha = selectedParent.puberty;
       fov = selectedParent.fov;

        double birthDate=getScheduler().getGVT();

        initNewAgent(sex,birthDate,fov,mu,lambda,alpha,omega,gestation,Xlocation,
            Ylocation,parent1.initialEnergy/2+parent2.initialEnergy/2);
        if(activator6!=null)
        {
                activator6.addAgent((EventGenerator)parent1.getBrain());
                activator6.addAgent((EventGenerator)parent2.getBrain());
        }
        if(getScheduler().getGVT()-lastUpdateBirths>=1)
        {
                birthsCount=0;
```

```
                lastUpdateBirths++;
        }
        birthsCount++;
}

public void end()
{
        for(Iterator i=modelAndEngineAgents.iterator();i.hasNext();)
        {
                AbstractAgent a = (AbstractAgent)i.next();
                killAgent(a);
        }
        totalAgentsNb=0;
        println("simulaiton end !!!");
        super.end();
}

public void killDead()
{
        SugarAgent sa = (SugarAgent) deaths.remove(new Double(GenScheduler.GVT));
        if(sa != null)
                land.killAgent(sa.getBody());
        else
                sugarscape.util.Debug.systemDebug("----------------BUG BUG---on kill
                    deads"+deaths);
}

public void createInitialPopulation()
{
        boolean sex=getRandomSource().nextBoolean();
        int fov = getRandomSource().nextInt(5)+1;
        double mu = getRandomSource().nextDouble()*(maxMU-minMU)+minMU;
        double lambda = getRandomSource().nextDouble()*(maxLAMBDA-minLAMBDA)+minLAMBDA;
        double alpha = getRandomSource().nextDouble()*(maxALPHA-minALPHA)+minALPHA;
        double omega;
        if(sex)
                omega = getRandomSource().nextDouble()*(maxMaleOMEGA-minMaleOMEGA)+
                    minMaleOMEGA;
        else
                omega = getRandomSource().nextDouble()*(maxFemaleOMEGA-minFemaleOMEGA)+
                    minFemaleOMEGA;
        int Xlocation;
        int Ylocation;

        do
        {
                Xlocation = getRandomSource().nextInt(squareSize);
                Ylocation = getRandomSource().nextInt(squareSize);
        }
        while(land.grid[Xlocation][Ylocation].getOccupant()!=null);
        double birthDate=getScheduler().getGVT();
        initNewAgentPopulation(sex,birthDate,fov,mu,lambda,alpha,omega,gestation,
            Xlocation,Ylocation,agentsInitialEnergy+randomSource.nextInt(50));
}

public void initNewAgentPopulation(boolean sex, double birthDate, int fov, double mu,
    double lambda, double alpha, double omega, int gestation, int x, int y,double
    initialEnergy)
{
        totalAgentsNb++;
        SugarAgent a;
        AgentPhysicalBody body = new AgentPhysicalBody(totalAgentsNb,sex,birthDate,fov,
            mu,lambda,alpha,omega,gestation,x,y,initialEnergy);
        a = new SugarAgent(community,getRandomSource(),moveInfluencesSpace,world,
            reproductionInfluenceSpace, body);
        body.setBrain(a);
        body.age=getRandomSource().nextDouble()*(maxLAMBDA);
        body.age-=20;
        if(body.age<0)
                body.age=0;
```

```
        body.birthDate = -body.age;
        if(activator6!=null)
        {
                myScheduler.addEventInDeltaT(new ObjectEvent(this,birthDate+body.
                    lifespan,"killDead"));
                deaths.put(new Double(birthDate+body.lifespan),a);
                activator6.addAgent((EventGenerator)a);
        }
        land.addAgent(a.getBody());
}

public void scheduleDeath(double time,AgentPhysicalBody agt)
{
        if(activator6!=null)
        {
                myScheduler.addEventInDeltaT(new ObjectEvent(this,time,"killDead"));
                deaths.put(new Double(time),agt.getBrain());
        }
}

}
```

B.3.2 Les activateurs

`AgentsInfluencesDiscreteTimeActivator` : activateur des agents en mode synchrone

```
public class AgentsInfluencesDiscreteTimeActivator extends DiscreteTimeActivator{
        InteractionLawForReactionComputation environmentReaction;
        InteractionLawForPerception perceptionLaw;
        List agentsSet;
        LandScapeAgent landscape;

    public AgentsInfluencesDiscreteTimeActivator(String community, String group, String
         role, String behaviorName,int priority, GenScheduler scheduler, double
        intervalStep,double startTime,InteractionLawForReactionComputation
        computationLaw,InteractionLawForPerception perceptionLaw, List l,
        LandScapeAgent landscape,PRNGInterface source){
        super(community, group, role, behaviorName,priority,scheduler,intervalStep,
            startTime,source);
        environmentReaction = computationLaw;
        this.perceptionLaw=perceptionLaw;
        agentsSet = l;
        this.landscape=landscape;
    }
synchronized public SimEvent execute(double GVT){
        if(method.equals("reproduce"))
                for (Iterator i = agentsSet.iterator();i.hasNext();)
                {
                        SugarAgent ma = (SugarAgent)((AgentPhysicalBody)i.next()).
                            getBrain();
                        //ma.clearOutbox();
                        if(! ma.getBody().reproducing)
                        {
                                perceptionLaw.giveAgentPerceptions(ma,false);
                                InteractionObject repro = ma.reproduceEvent();
                                if(repro != null)
                                        landscape.reproductionsMap.put(ma,repro);
                        }
                }
        else
        {
                List tmp = new ArrayList(agentsSet);
                Collections.sort(tmp);
                if(randomSource instanceof Random)
                        Collections.shuffle(tmp,(Random)randomSource); // to be
                            determinist
```

```
                else
                        MyCollections.shuffle(tmp,randomSource);
                for (Iterator i = tmp.iterator();i.hasNext();)
                {
                        SugarAgent ma = (SugarAgent)((AgentPhysicalBody)i.next()).
                            getBrain();
                        if(! ma.getBody().reproducing)
                        {
                                perceptionLaw.giveAgentPerceptions(ma,true);
                                InteractionObject result = ma.moveEvent();
                                if(result != null)
                                        landscape.computeEventMove((Movement)result);
                        }
                }
         }
        return new SimEvent(this, intervalStep+GVT);
}

}
```

`AgentsInfluencesDiscreteEventActivator` : activateur des agents en mode asynchrone

```
public class AgentsInfluencesDiscreteEventActivator extends DiscreteEventActivator{
        InteractionLawForReactionComputation environmentReaction;
        InteractionLawForPerception perceptionLaw;
        LandScapeAgent landscape;
        HashMap events = new HashMap(SugarScapeModel.agtNbEstimation);

    public AgentsInfluencesDiscreteEventActivator(String community, String group,
        String role, String behaviorName,int priority, GenScheduler scheduler,
        InteractionLawForReactionComputation computationLaw,InteractionLawForPerception
         perceptionLaw, LandScapeAgent landscape){
        super(community, group, role, behaviorName,priority,scheduler);
        environmentReaction = computationLaw;
        this.landscape=landscape;
        this.perceptionLaw=perceptionLaw;
    }

synchronized public void update(AbstractAgent theAgent, boolean added){
        if(added)
                addAgent((EventGenerator) theAgent);
        else
                removeAgent(theAgent);
}

synchronized public void initialize(List agentsSet){
        for (Iterator i = agentsSet.iterator();i.hasNext();)
        {
                EventGenerator eg = (EventGenerator) ((AgentPhysicalBody)i.next()).
                    getBrain();
                double deltaT= GenScheduler.GVT + eg.nextEventDeltaTimeFor(method);
                SimEvent event = new DiscreteSimEvent(this,deltaT,eg);
                events.put(eg,event);
                myScheduler.addEventInDeltaT(event);
        }
}

synchronized public void addAgent(EventGenerator eg){
        double deltaT= GenScheduler.GVT + eg.nextEventDeltaTimeFor(method);
        SimEvent event = new DiscreteSimEvent(this,deltaT,eg);
        events.put(eg,event);
        myScheduler.addEventInDeltaT(event);
}

synchronized public void removeAgent(Object o){
        SimEvent se = (SimEvent) events.remove(o);
```

```
        if(se != null)
                myScheduler.removeEvent(se);
}

synchronized public SimEvent execute(double GVT, Object what){
        SugarAgent ma = (SugarAgent) what;
        if(landscape.updateAgent(ma.getBody()))//updating the agent
        {
                if(! ma.getBody().reproducing)
                {
                        landscape.reproductionsMap.remove(ma);
                        perceptionLaw.giveAgentPerceptions(ma,ma.last);
                        InteractionObject result = ma.compute();
                        if(result != null)
                                if(result instanceof Movement)
                                        landscape.computeEventMove((Movement)result);
                                else
                                        landscape.computeEventBirthReaction((
                                            ReproductionAttempt)result,ma);
                        if(! events.containsKey(what))
                        {
                                double nextEventDeltaTime = GVT+ma.
                                    nextEventDeltaTimeFor(method);
                                double tmp = landscape.agentSurviving(
                                    nextEventDeltaTime-GVT,ma.getBody());
                                if(tmp==-1)
                                {
                                        SimEvent event = new DiscreteSimEvent(this,
                                            nextEventDeltaTime,what);
                                        events.put(what,event);
                                        return event;
                                }
                        }
                }
        }
return null;
}

}
```

`EnvironmentReactionDiscreteTimeActivator` : activation de la réaction

```
public class EnvironmentReactionDiscreteTimeActivator extends DiscreteTimeActivator
{
        InteractionLawForReactionComputation environmentReaction;

    public EnvironmentReactionDiscreteTimeActivator(String community, String group,
        String role, String behaviorName,int priority, GenScheduler scheduler, double
        intervalStep,double startTime,InteractionLawForReactionComputation
        interactionLaw,PRNGInterface source){
        super(community, group, role, behaviorName,priority,scheduler,intervalStep,
            startTime,source);
        environmentReaction = interactionLaw;
    }

synchronized public SimEvent execute(double GVT){
        for (Iterator i = getAgentsIterator();i.hasNext();)
        {
                LandScapeAgent la = (LandScapeAgent)i.next();
                //environment gets influences
                environmentReaction.computeInteraction(la.getAgentPlugin());
                executeBehaviorOf(la); //environment computes reaction
        }
        return new SimEvent(this, intervalStep+GVT);
}

}
```

B.3.3 Deux lois d'évolution MIC*

`InteractionLawForPerception` : perception des agents

```
public class InteractionLawForPerception{
        InteractionSpace w;
        Landscape l;

        public InteractionLawForPerception(InteractionSpace world,Landscape l){
                w = world;
                this.l=l;
        }

public void computeAgentPerceptions(AgentPlugin a){
        MICKernel.getScheduler().applyInteractionLaw(new InteractionLaw(w.getId(),a));
}
}
```

`InteractionLawForReactionComputation` : perception des influences par l'environnement

```
public class InteractionLawForReactionComputation
{
        InteractionSpace moveInfluencesSpace,reproductionInfluenceSpace;
        LandScapeAgent envAgent;

        public InteractionLawForReactionComputation(InteractionSpace
            moveInfluencesSpace,InteractionSpace reproductionInfluenceSpace,
            LandScapeAgent envAgent){
                this.moveInfluencesSpace = moveInfluencesSpace;
                this.reproductionInfluenceSpace = reproductionInfluenceSpace;
                this.envAgent = envAgent;
        }

public void computeInteraction(AgentPlugin worldPlugin){
        MICKernel.getScheduler().applyInteractionLaw(new InteractionLaw(
            moveInfluencesSpace.getId(),worldPlugin));
        MICKernel.getScheduler().applyInteractionLaw(new InteractionLaw(
            reproductionInfluenceSpace.getId(),worldPlugin));
}

}
```

Références bibliographiques

[Amblard *et al.* , 2003]Amblard, Frédéric, Hill, David R.C., Bernard, Stéphan, Truffot, Jérome, & Deffuant, Guillaume. 2003. MDA Compliant Design of SimExplorer : A Software Tool to Handle Simulation Experimental Frameworks. *Pages 279–284 of :* Bruzzone, Agostino G. (ed), *Proceedings of 2003 Summer Computer Simulation Conference (SCSC'03)*. Montreal, Canada : Society for Computer Simulation.

[Amiguet *et al.* , 2003]Amiguet, Matthieu, Müller, Jean-Pierre, Báez-Barranco, José-Antonio, & Nagy, Adina. 2003. The MOCA Platform. *Pages 70–88 of :* Sichman, Jaime Simão, Bousquet, François, & Davidsson, Paul (eds), *Multi-Agent-Based Simulation II, Proceedings of MABS 2002, Third International Worshop*. LNAI, vol. 2581. Springer.

[Arthur & Nance, 1996]Arthur, James D., & Nance, Richard E. 1996. Independent verification and validation : a missing link in simulation methodology ? *Pages 230–236 of : Proceedings of the 28th conference on Winter simulation*. ACM Press.

[Arthur & Nance, 2000]Arthur, James D., & Nance, Richard E. 2000. V&A ; III : verification and validation without independence : a recipe for failure. *Pages 859–865 of : Proceedings of the 32nd conference on Winter simulation*. Society for Computer Simulation International.

[Axelrod, 1997]Axelrod, Robert. 1997. Advancing the Art of Simulation in the Social Sciences. *Pages 21–40 of :* Conte, Rosaria, Hegselmann, Rainer, & Terna, Pietro (eds), *Simulating Social Phenomena*, vol. 456. Berlin : Lecture Notes in Economics and Mathematical Systems, Springer-Verlag.

[Axtell, 2000a]Axtell, Robert L. 2000a. Effects of Interaction Topology and Activation Regime in Several Multi-Agent Systems. *Pages 33–48 of :* Moss, Scott, & Davidson, Paul (eds), *Proceedings of the* 2^{nd} *Workshop on Modelling Agent Based Systems, MABS'00*, vol. 1979. Lecture Notes in Artificial Intelligence LNAI, Springer-Verlag, Berlin.

[Axtell, 2000b]Axtell, Robert L. 2000b (November). *Why Agents ? On the Varied Motivations for Agent Computing in the Social Sciences, CSED Working Paper No. 17*. Tech. rept. Center on Social and Economic Dynamics, The Brookings Institution.

[Balasubramaniyan *et al.* , 1998]Balasubramaniyan, J. S., Garcia-Fernandez, J. O., Isacoff, D., Spafford, Eugene H., & Zamboni, Diego. 1998. An Architecture for Intrusion Detection Using Autonomous Agents. *Pages 13–24 of : Proceedings of the 14th Annual Computer Security Applications Conference (ACSAC 1998), Scottsdale, AZ, USA*. IEEE Computer Society.

[Balci, 1988]Balci, Osman. 1988. The implementation of four conceptual frameworks for simulation modeling in high-level languages. *Pages 287–295 of : Proceedings of the 20th conference on Winter simulation*. ACM Press.

[Balci, 1998]Balci, Osman. 1998. Verification, validation, and accreditation. *Pages 41–4 of : Proceedings of the 30th conference on Winter simulation*. IEEE Computer Society Press.

[Barber *et al.* , 2001]Barber, K. Suzanne, McKay, Ryan, MacMahon, Matt, Martin, Cheryl E., Lam, Dung N., Goel, Anuj, Han, David C., & Kim, Joonoo. 2001. Sensible agents : an implemented multi-agent system and testbed. *Pages 92–99 of : Proceedings of the fifth international conference on Autonomous agents.* ACM Press.

[Barros, 1997]Barros, Fernando J. 1997. Modeling formalisms for dynamic structure systems. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, **7**(4), 501–515.

[Beaufils *et al.* , 1998]Beaufils, Bruno, Delahaye, Jean-Paul, & Mathieu, Philippe. 1998. Complete Classes of Strategies for the Classical Iterated Prisoner's Dilemma. *Pages 33–41 of : Proceedings of the 7th International Conference on Evolutionary Programming VII.* Springer-Verlag.

[Beurier *et al.* , 2002]Beurier, Gregory, Simonin, Olivier, & Ferber, Jacques. 2002 (December). Model and Simulation of Multi-Level Emergence. *In : in the 2nd IEEE International Symposium on Signal Processing and Information Technology, ISSPIT'02.*

[Bousquet *et al.* , 1998]Bousquet, François, Bakam, Innocent, Proton, Hubert, & Page, Christophe Le. 1998. Cormas : COmmon-pool Resources and Multi-Agent Systems. *Pages 826–837 of : Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial In telligence and Expert Systems.* Springer-Verlag.

[Briot & Demazeau, 2001a]Briot, Jean-Pierre, & Demazeau, Yves (eds). 2001a. *Introduction aux agents. In :*[Briot & Demazeau, 2001b]. Pages 17–25.

[Briot & Demazeau, 2001b]Briot, Jean-Pierre, & Demazeau, Yves (eds). 2001b. *Principes et architecture des systèmes multi-agents.* Collection IC2. Paris : Hermes Science Publications.

[Brooks, 1991]Brooks, Rodney A. 1991. Intelligence Without Reason. *Pages 569–595 of :* Myopoulos, John, & Reiter, Ray (eds), *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91).* Morgan Kaufmann publishers Inc. : San Mateo, CA, USA.

[Brooks, 1992]Brooks, Rodney A. 1992. Artificial Life and Real Robots. *Pages 3–10 of :* Varela, Francisco J., & Bourgine, Paul (eds), *Toward a Practice of Autonomous Systems : Proceedings of the First European Conference on Artificial Life.* MIT press, Cambridge, Massachusetts, USA.

[Brooks & Connell, 1986]Brooks, Rodney A., & Connell, Jonathan H. 1986 (October). Asynchronous distributed control system for a mobile robot. *Pages 77–84 of :* Wolfe, W., & Marquina, N. (eds), *SPIE's Cambridge Symposium on Optical and Opto-Elecronic Engineering*, vol. 727.

[Buettner & Siler, 1976]Buettner, Grace M., & Siler, William. 1976. Variability in predator-prey experiments : simulation using a stochastic model. *Pages 194–201 of : Proceedings of the 14th annual Southeast regional conference.* ACM Press.

[Campos, 2000]Campos, André M. C. 2000 (8 septembre). *Une architecture logicielle pour le dveloppement de simulations visuelles et interactives individu-centrées : application à la simulation d'écosystèmes et à la simulation sur le Web.* Thèse de Doctorat, Université Blaise Pascal, Clermont II.

[Cardelli, 1999]Cardelli, Luca. 1999. Abstractions for Mobile Computation. *Pages 51–94 of : Secure Internet Programming : Security Issues for Mobile and Distributed Object.* Lecture notes in computer science LNCS, vol. 1603. Springer Verlag.

[Cariani, 1991]Cariani, Peter. 1991. Emergence and Artificial Life. *Pages 775–798 of :* Langton, Christopher G., Taylor, Charles, Farmer, J. Doyne, & Rasmussen, Steen (eds), *Artificial Life II, Proceedings of the Workshop on Artificial Life.* Santa Fe Institute, Studies in the Sciences of Complexity, vol. X. Santa Fe, New Mexico : Addison-Wesley.

[Cellier, 1991a]Cellier, François E. 1991a. *Continuous System Modeling.* Springer-Verlag.

[Cellier, 1991b]Cellier, François E. 1991b. Qualitative modeling and simulation : promise or illusion. *Pages 1086–1090 of : Proceedings of the 23rd conference on Winter simulation.* IEEE Computer Society.

[Cellier, 1986]Cellier, François E. 1986. Combined continuous/discrete simulation : applications, techniques and tools. *Pages 24–33 of : Proceedings of the 18th conference on Winter simulation.* ACM Press.

[Chang *et al.* , 2005]Chang, Paul H., Chen, Kuang-Tai, Chien, Yu-Hung, Kao, Edward, & Soo, Von-Wun. 2005. From reality to mind : A cognitive Middle Layer of Environment Concepts for Believable Agents. *Pages 57–73 of :* [Weyns *et al.* , 2005a].

[Chapelle *et al.* , 2002]Chapelle, Jérôme, Simonin, Olivier, & Ferber, Jacques. 2002 (July 21-26). How Situated Agents can Learn to Cooperate by Monitoring their Neighbors' Satisfaction. *In : Proceedings of the* 15^{th} *European Conference on Artificial Intelligence ECAI'2002.*

[Chow & Zeigler, 1994]Chow, Alex Chung Hen, & Zeigler, Bernard P. 1994. Parallel DEVS : a parallel, hierarchical, modular, modeling formalism. *Pages 716–722 of : Proceedings of the 26th conference on Winter simulation.* Society for Computer Simulation International.

[Cohen & Levesque, 1991]Cohen, Philip R., & Levesque, Hector J. 1991. Teamwork. *Nous, Special Issue on Cognitive Science and Artificial Intelligence*, **25**(4), 487–512.

[Cohen *et al.* , 1989]Cohen, Philip R., Greenberg, M. L., Hart, D. M., & Howe, A. E. 1989. Trial by fire : understanding the design requirements for agents in complex environments. *AI Magazine.*, **10**(3), 34–48.

[Collier, 2002]Collier, Nick. 2002. *RePast : the REcursive Porous Agent Toolkit.* http ://repast.sourceforge.net/.

[Conte *et al.* , 1998]Conte, Rosaria, Gilbert, Nigel, & ao Sichman, Jaime Sim̀1998. MAS and Social Simulation : A Suitable Commitment. *Pages 1–9 of : Proceedings of the First International Workshop on Multi-Agent Systems and Agent-Based Simulation.* Springer-Verlag.

[Courdier *et al.* , 1998]Courdier, Rémy, Marcenac, Pierre, & Calderoni, Stéphane. 1998. Zooming on a Multiagent Simulation System : From the Conceptual Architecture to the Interaction Protocol. *Page 411 of : Proceedings of the 3rd International Conference on Multi Agent Systems.* IEEE Computer Society.

[da Silva & Demazeau, 2002]da Silva, Joao Luis T., & Demazeau, Yves. 2002. Vowels co-ordination model. *Pages 1129–1136 of : Proceedings of the first international joint conference on Autonomous agents and multiagent systems AAMAS'02.* ACM Press.

[David *et al.* , 2002]David, Nuno, Sichman, Jaime S., & Coelho, Helder. 2002. Towards an Emergence-Driven Software Process for Agent-Based Simulation. *Pages 89–104 of :* Sichman, Jaime S., Bousquet, François, & Davidsson, Paul (eds), *Multi-Agent-Based Simulation II, Proceedings of MABS 2002, Third International Worshop.* LNAI, vol. 2581. Springer-Verlag 2003.

[Dàvila & Tucci, 2000]Dàvila, Jacinto, & Tucci, Kay. 2000 (October 22-24). Towards a logic-based, multi-agent simulation theory. *In : International Conference on Modeling, Simulation and Neural Networks MSNN'2000.*

[Dàvila & Uzcágegui, 2000]Dàvila, Jacinto, & Uzcágegui, Mayerlin. 2000 (October 22-24). GALATEA : A Multi-agent, simulation platform. *In : International Conference on Modeling, Simulation and Neural Networks MSNN'2000.*

[Decker & Lesser, 1993]Decker, Keith, & Lesser, Victor R. 1993. Quantitative Modeling of Complex Environments. *International Journal of Intelligent Systems in Accounting, Finance*

and Management. Special Issue on Mathematical and Computational Models and Characteristics of Agent Behaviour., **2**(January), 215–234.

[Decker, 1996]Decker, Keith S. 1996. Distributed artificial intelligence testbeds. *Chap. 3, pages 119–138 of :* O'Hare, Gregory M. P., & Jennings, Nick. R. (eds), *Foundations of distributed artificial intelligence.* John Wiley & Sons, Inc.

[Deffuant *et al.* , 2002]Deffuant, Guillaume, Amblard, Frédéric, Weisbuch, Gérard, & Faure, Thierry. 2002. How can extremism prevail ? A study based on the relative agreement interaction model. *The Journal of Artificial Societies and Social Simulation, JASSS*, **5**(4).

[Demazeau, 1995]Demazeau, Yves. 1995 (April 4-7). From interactions to collective behaviour in agent-based systems. *Pages 117–132 of : the First European conference on cognitive science ECCS '95.*

[Demazeau, 2001]Demazeau, Yves. 2001 (avril). *VOYELLES.* Habilitation à Diriger des Recherches, Institut National Polytechnique de Grenoble INPG, Grenoble.

[Dresner & Stone, 2004]Dresner, Kurt, & Stone, Peter. 2004. Multiagent Traffic Management : A Reservation-Based Intersection Control Mechanism. *Pages 530–537 of :* Jennings, Nicholas R., Sierra, Carles, Sonenberg, Liz, & Tambe, Milind (eds), *Proceedings of The Third International Joint Conference on Autonomous Agents & Multi Agent Systems (AAMAS'2004)*, vol. 2. ACM Press.

[Drogoul, 1993]Drogoul, Alexis. 1993 (23 Novembre). *De La Simulation Multi-Agent A La Résolution Collective de Problèmes : Une Étude De l'Émergence De Structures D'Organisation Dans Les Systèmes Multi-Agents.* Thèse de Doctorat, Université Paris VI.

[Drogoul & Ferber, 1992]Drogoul, Alexis, & Ferber, Jacques. 1992. Multi-Agent Simulation as a Tool for Modeling Societies : Application to Social Differentiation in Ant Colonies. *Pages 3–23 of :* Castelfranchi, Cristiano, & Werner, Eric (eds), *Artificial Social Systems, 4^{th} European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW '92, S. Martino al Cimino, Italy, July 29-31, 1992, Selected Papers.* Lecture Notes in Computer Science, vol. 830. Springer-Verlag 1994.

[Drogoul & Picault, 1999]Drogoul, Alexis, & Picault, Sébastien. 1999. MICRobES : Vers des collectivités de robots socialement situés. *Pages 265–278 of :* Gleizes, Marie-Pierre, & Marcenac, Pierre (eds), *VII^{èmes} Journées Francophones pour l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents JFIADSMA'99.* Hermès, Paris.

[Drogoul *et al.* , 2002]Drogoul, Alexis, Vanbergue, Diane, & Meurisse, Thomas. 2002. Multi-Agent Based Simulation : Where are the Agents ? *Pages 89–104 of :* Sichman, Jaime S., Bousquet, François, & Davidsson, Paul (eds), *Multi-Agent-Based Simulation II, Proceedings of MABS 2002, Third International Worshop.* LNAI, vol. 2581. Springer-Verlag.

[Duboz, 2004]Duboz, Raphaël. 2004 (30 mars). *Intégration de modèles hétérogènes pour la modélisation et la simulation de systèmes complexes. Application à la modélisation multi-échelles en écologie marine.* Thèse de Doctorat, Université du Littoral, Côte d'Opale, Calais.

[Duffy, 2001]Duffy, John. 2001. Learning to Speculate : Experiments with Artificial and Real Agents. *Journal of Economic Dynamics and Control*, **25**(3/4), 295–319.

[Dumont & Hill, 2001]Dumont, Bertrand, & Hill, David R.C. 2001. Multi-agent simulation of group foraging in sheep : effects of spatial memory, conspecific attraction and plot size. *Ecological Modelling*, **141**(July 1), 201–215.

[Edmonds & Hales, 2003]Edmonds, Bruce, & Hales, David. 2003 (April). Replication, Replication and Replication – Some Hard Lessons from Model Alignment. *In : Model to Model Workshop M2M.*

[Epstein & Axtell, 1996]Epstein, Joshua M., & Axtell, Robert L. 1996. *Growing Artificial Societies.* Brookings Institution Press, Washington D.C.

[Ferber, 1999]Ferber, Jacques. 1999. *Multi-Agent Systems : An Introduction to Distributed Artificial Intelligence.* Addison-Wesley Longman Publishing Co., Inc.

[Ferber & Carle, 1991]Ferber, Jacques, & Carle, Patrice. 1991. Actors and agents as reflective concurrent objects : a MERING IV perspective. *IEEE Transactions on Systems, Man, and Cybernetics*, **21**(6), 1420–1436.

[Ferber & Gutknecht, 1998]Ferber, Jacques, & Gutknecht, Olivier. 1998. A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems. *Page 128 of : Proceedings of the 3rd International Conference on Multi Agent Systems.* IEEE Computer Society.

[Ferber & Müller, 1996]Ferber, Jacques, & Müller, Jean-Pierre. 1996. Influences and Reaction : a Model of Situated Multi-agent Systems. *Pages 72–79 of :* Tokoro, Mario (ed), *Proceedings of the 2nd International Conference on Multi-agent Systems (ICMAS-96).* The AAAI Press.

[Ferber *et al.* , 2003]Ferber, Jacques, Gutknecht, Olivier, & Michel, Fabien. 2003. From Agents to Organizations : an Organizational View of Multi-Agent Systems. *Pages 185–202 of :* Paolo Giorgini, Jörg P. Müller, James Odell (ed), *Agent-Oriented Software Engineering IV : 4th International Workshop, Aose 2003.* Lecture notes in computer science LNCS. Springer Verlag.

[Ferber *et al.* , 2005]Ferber, Jacques, Michel, Fabien, & Báez-Barranco, José-Antonio. 2005. AGRE : Integrating Environments with Organizations. *Pages 48–56 of :* [Weyns *et al.* , 2005a].

[Fishwick, 1997]Fishwick, Paul A. 1997. Computer simulation : growth through extension. *Transactions of the Society for Computer Simulation International*, **14**(1), 13–23.

[Fishwick & Zeigler, 1992]Fishwick, Paul A., & Zeigler, Bernard P. 1992. A multimodel methodology for qualitative model engineering. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, **2**(1), 52–81.

[Fyanio *et al.* , 1998]Fyanio, Edem, Treuil, Jean-Pierre, Perrier, Edith, & Demazeau, Yves. 1998. Multi-agent Architecture Integrating Heterogeneous Models of Dynamical Processes : the Representation of Time. *In :* Schiman, Jaime S., Conte, Rosaria, & Gilbert, Nigel (eds), *Proceedings of the* 1^{st} *Workshop on Modelling Agent Based Systems, MABS'98*, vol. 1534. Lecture Notes in Artificial Intelligence LNAI, Springer-Verlag, Berlin.

[Gasser & Kakugawa, 2002]Gasser, Les, & Kakugawa, Kelvin. 2002. MACE3J : fast flexible distributed simulation of large, large-grain multi-agent systems. *Pages 745–752 of : Proceedings of the first international joint conference on Autonomous agents and multiagent systems.* ACM Press.

[Gasser *et al.* , 1987]Gasser, Les, Braganza, Carl, & Herman, Nava. 1987. MACE : A Flexible Testbed for Distributed AI Research. *Distributed Artificial Intelligence*, 119–152.

[Genesereth & Nilsson, 1987]Genesereth, Michael R., & Nilsson, Nils J. 1987. *Logical foundations of artificial intelligence.* Morgan Kaufmann Publishers Inc.

[Gilbert & Troitzsch, 1999]Gilbert, Nigel, & Troitzsch, Klaus G. 1999. *Simulation for the Social Scientist.* Open University Press.

[Ginot & Page, 1998]Ginot, Vincent, & Page, Christophe Le. 1998. Mobidyc, a Generic Multi-Agents Simulator for Modeling Populations Dynamics. *Pages 805–814 of : Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial In telligence and Expert Systems.* Springer-Verlag.

[Goldspink, 2002]Goldspink, Chris. 2002. Methodological Implications Of Complex Systems Approaches to Sociality : Simulation as a foundation for knowledge. *The Journal of Artificial Societies and Social Simulation, JASSS*, **5**(1).

[Gouaïch & Guiraud, 2002]Gouaïch, Abdelkader, & Guiraud, Yves. 2002. {Movement, Interaction, Calculus}* : an algebraic environment for distributed and mobile calculus. *In : Proceedings of the First International NAISO Congress on Autonomous Intelligent Systems (ICAIS 2002).* NAISO Academic Press, Canada/The Netherlands.

[Gouaïch, 2003]Gouaïch, Abdelkader. 2003 (July 15). Requirements for achieving software agents autonomy and defining their responsibility. *In : The First International Workshop on Computational autonomy - Potential, Risks, Solutions (autonomy 2003).*

[Gouaïch, 2005]Gouaïch, Abdelkader. 2005 (juillet). *Movement, Interaction, Calculation as Primitives for Everywhere & Anytime Computing.* Thèse de Doctorat, Université Montpellier II, Montpellier.

[Gouaïch *et al.* , 2003]Gouaïch, Abdelkader, Guiraud, Yves, & Michel, Fabien. 2003 (July). MIC* : An Agent Formal Environment. *In : the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003), session on Agent Based Computing ABC'03.*

[Gouaïch *et al.* , 2005]Gouaïch, Abdelkader, Michel, Fabien, & Guiraud, Yves. 2005. MIC* : A Deployment Environment for Autonomous Agents. *Pages 109–126 of :* [Weyns *et al.* , 2005a].

[Griffiths & Luck, 1999]Griffiths, Nathan, & Luck, Michael. 1999. Cooperative Plan Selection through Trust. *Pages 162–174 of :* Garijo, Francisco J., & Boman, Magnus (eds), *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99)*, vol. 1647. Springer-Verlag : Heidelberg, Germany.

[Grimm, 1999]Grimm, Volker. 1999. Ten years of individual-based modelling in ecology : what have we learned, and what could we learn in the future ? *Ecological Modelling*, **115**, 129–148.

[Guessoum, 1996]Guessoum, Zahia. 1996 (mai). *Un environnement opérationnel de conception et de réalisation de systèmes multi-agents.* Thèse de Doctorat, Université Paris 6, LAFORIA, Paris.

[Guessoum, 2000]Guessoum, Zahia. 2000. A multi-agent simulation framework. *Transactions of the Society for Computer Simulation International*, **17**(1), 2–11.

[Guessoum & Briot, 1999]Guessoum, Zahia, & Briot, Jean-Pierre. 1999. From Active Objects to Autonomous Agents. *IEEE Concurrency, Special series on Actors and Agents*, **7**(3), 68–76.

[Gutknecht, 2001]Gutknecht, Olivier. 2001 (14 septembre). *Proposition d'un modèle générique de systèmes multi-agents. Examen de ses conséquences formelles, implémentatoires et méthodologiques.* Thèse de Doctorat, Université Montpellier II.

[Gutknecht & Ferber, 1998]Gutknecht, Olivier, & Ferber, Jacques. 1998. Un meta-modèle pour l'analyse, la conception et l'exècution de systémes multi-agents. *In :* Barthes, Jean-Pierre (ed), *Actes des Journèes Francophones en Intelligence Artificielle Distribuèe et Systémes Multi-Agents 1998.* Hermés.

[Gutknecht *et al.* , 2001]Gutknecht, Olivier, Ferber, Jacques, & Michel, Fabien. 2001. Integrating tools and infrastructures for generic multi-agent systems. *Pages 441–448 of : Proceedings of the fifth international conference on Autonomous agents, AA 2001.* ACM Press.

[Hanks *et al.* , 1993]Hanks, Steve, Pollack, Martha E., & Cohen, Paul R. 1993. Benchmarks, test beds, controlled experimentation, and the design of agent architectures. *AI Magazine*, **14**(4), 17–42.

[Hill, 2003]Hill, David R.C. 2003. URNG : A portable optimization technique for software applications requiring pseudo-random numbers. *Simulation Modelling Practice and Theory*, **11**(7-8), 643–654.

[Hraber *et al.* , 1997]Hraber, Peter T., Jones, Terry, & Forrest, Stephanie. 1997. The ecology of echo. *Artificial Life*, **3**(3), 165–190.

[Huberman & Glance, 1993]Huberman, B. A., & Glance, N. S. 1993. Evolutionary Games and Computer Simulations. *Pages 7716–7718 of : Proceedings of the National Academy of Science USA*, vol. 90.

[Huget, 2001]Huget, Marc Philippe. 2001 (15 juin). *Une ingénierie des protocoles d'interaction pour les systèmes multi-agents.* Thèse de Doctorat, Université Paris IX, Dauphine, Paris.

[Hymore, 1981]Hymore, A. Hayne. 1981. Applications of mathematical system theory to system design, modelling and simulation. *Pages 209–219 of : Proceedings of the 13th conference on Winter simulation.*

[Ingalls, 2001]Ingalls, Ricki G. 2001. Introduction to simulation. *Pages 7–16 of : Proceedings of the 33nd conference on Winter simulation.* IEEE Computer Society.

[Jennings & Wooldridge, 1999]Jennings, Nicholas R., & Wooldridge, Michael. 1999. Agent-Oriented Software Engineering. *Pages 1–7 of :* Garijo, Francisco J., & Boman, Magnus (eds), *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99)*, vol. 1647. Springer-Verlag : Heidelberg, Germany.

[Kieken, 2003]Kieken, Hubert. 2003. Le rôle des modèles dans la gestion de l'environnement. *Pages 141–151 of :* Müller, Jean-Pierre (ed), *Le statut épistmologique de la simulation - 10èmes journeés de Rochebrune : rencontre interdisciplinaire sur les systèmes complexes naturels et artificiels.* Telecom Paris.

[Kinny *et al.* , 1992]Kinny, David, Ljungberg, Magnus, Anand, Sonenberg, Elizabeth, Tidhar, Gil, & Werner, Eric. 1992. Planned Team Activity. *Pages 226–256 of :* Castelfranchi, Cristiano, & Werner, Eric (eds), *Artificial Social Systems — Selected Papers from the Fourth European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW-92 (LNAI Volume 830).* Springer-Verlag : Heidelberg, Germany.

[Kitano *et al.* , 1997]Kitano, Hiroaki, Asada, Minoru, Kuniyoshi, Yasuo, Noda, Itsuki, & Osawa, Eiichi. 1997. RoboCup : The Robot World Cup Initiative. *Pages 340–347 of : Proceedings of the first international conference on Autonomous agents.* ACM Press.

[Klir, 1985]Klir, George J. 1985. *Architecture of Systems Problem Solving.* Plenum Pub Corp.

[Klügl *et al.* , 2002]Klügl, Franziska, Oechslein, Christoph, Puppe, Frank, & Dornhaus, Anna. 2002. Multi-Agent Modelling in Comparison to Standard Modelling. *Pages 105–110 of :* Barros, Fernando J., & Giambasi, Norbert (eds), *Artificial Intelligence, Simulation and Planning in High Autonomous Systems.* Lisbon, Portugal : SCS Publishing House.

[Lawson & Park, 2000]Lawson, Barry G., & Park, Steve. 2000. Asynchronous Time Evolution in an Artificial Society Mode. *The Journal of Artificial Societies and Social Simulation, JASSS*, **3**(1).

[L'Ecuyer, 1990]L'Ecuyer, Pierre. 1990. Random numbers for simulation. *Communications of the ACM*, **33**(10), 85–97.

[Lesser & Corkill, 1983]Lesser, Vicotr R., & Corkill, Daniel D. 1983. The Distributed Vehicle Monitoring Testbed : A Tool for Investigating Distributed Problem Solving Networks. *AI Magazine*, **4**(3), 15–33.

[Lhuillier, 1998]Lhuillier, Marc. 1998 (février). *Une approche à base de composants logiciels pour la conception d'agents. Principe et mise en oeuvre à travers la plate-forme Maleva.* Thèse de Doctorat, Université Paris VI.

[Lucidarm *et al.* , 2002]Lucidarm, Philippe, Simonin, Olivier, & Liégois, Alain. 2002 (May 11-15). Implementation and Evaluation of a Satisfaction/Altruism Based Architecture for Multi-Robot Systems. *Pages 1007–1012 of : Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'02.*

[MacKenzie *et al.* , 1997]MacKenzie, Douglas C., Arkin, Ronald, & Cameron, Jonathan M. 1997. Multiagent Mission Specification and Execution. *Autonomous Robots*, **4**(1), 29–52.

[Magnin, 1996]Magnin, Laurent. 1996 (28 Novembre). *Modélisation et simulation de l'environnement dans les systèmes multi-agents : application aux robots footballeurs.* Thèse de Doctorat, Université Paris VI.

[Malone & Crowston, 1994]Malone, Thomas W., & Crowston, Kevin. 1994. The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR)*, **26**(1), 87–119.

[Marcenac & Giroux, 1998]Marcenac, Pierre, & Giroux, Sylvain. 1998. GEAMAS : A Generic Architecture for Agent-Oriented Simulations of Complex Processes. *Applied Intelligence*, **8**(3), 247–267.

[Matsumoto & Nishimura, 1998]Matsumoto, Makato, & Nishimura, Takuji. 1998. Mersenne Twister : A 623-dimensionnaly equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulations Special Issue on Uniform Random Number Generation*, **8**(1), 3–30.

[Meurisse, 2004]Meurisse, Thomas. 2004 (1^{er} juillet). *Simulation multi-agent : Du modèle à l'opérationnalisation.* Thèse de Doctorat, Université de Paris VI, Paris.

[Meurisse & Vanbergue, 2001]Meurisse, Thomas, & Vanbergue, Diane. 2001 (septembre 6-8). Et maintenant à qui le tour ? Aperçu de Problématiques de Conception de Simulations Multi-Agents. *In : Actes de la conférence Agents Logiciels, Coopration, Apprentissage et Activités humaines ALCAA'01.*

[Michel, 2000]Michel, Fabien. 2000 (Septembre). Une approche méthodologique pour l'analyse et la conception de simulateur multi-agents. *Pages 269–279 of : Cinquièmes rencontres des Jeunes Chercheurs en Intelligence Artificielle.* AFIA.

[Michel, 2001]Michel, Fabien. 2001 (21-23 Mai). Le modèle Influence/Réaction pour la Simulation Multi-Agents. *Pages 391–406 of :* Chaib-draa, Brahim, & Enjalbert, Patrice (eds), *Actes des* $1^{ères}$ *Journées Francophones des Modèles Formels de l'Interaction, MFI' 01*, vol. 3.

[Michel, 2002]Michel, Fabien. 2002 (June). *An Introduction to TurtleKit : a Platform for Building Logo Based Multi-Agent Simulations with MadKit.* Tech. rept. RR LIRMM 002215. Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier, LIRMM, CNRS, Montpellier.

[Michel & Bommel, 2002]Michel, Fabien, & Bommel, Pierre. 2002. Simulation Distribuée Interactive sous MadKit. *Pages 175–178 of :* Mathieu, Philippe, & Müller, Jean-Pierre (eds), $X^{èmes}$ *Journées Francophones pour l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents JFIADSMA'02.* Hermès, Paris.

[Michel *et al.* , 2003]Michel, Fabien, Gouaïch, Abdelkader, & Ferber, Jacques. 2003. Weak Interaction and Strong Interaction in Agent Based Simulations. *Pages 43–56 of :* Hales, David, Edmonds, Bruce, Norling, Emma, & Rouchier, Juliette (eds), *Multi-Agent-Based Simulation III, Proceedings of MABS 2003, Fourth International Worshop*, vol. LNAI 2927. Lecture Note in Artificial Intelligence, Springer-Verlag.

[Milner, 1999]Milner, Robin. 1999. *Communicating and mobile systems : the π-calculus.* Cambridge University Press.

[Minar *et al.* , 1996]Minar, Nelson, Burkhart, Rogert, Langton, Chris, & Askenazi, Manor. 1996. *The Swarm Simulation System : A Toolkit for Building Multi-Agent Simulations.* Santa Fe Institute Working Paper #96-06-042.

[Minsky, 1986]Minsky, Marvin. 1986. *The society of mind.* Simon & Schuster, Inc.

[Moss *et al.* , 1998]Moss, Scott, Gaylard, Helen, Wallis, Steve, & Edmonds, Bruce. 1998. SDML : A Multi-Agent Language for Organizational Modelling. *Computational & Mathematical Organization Theory*, **4**(1), 43–69.

[Odell *et al.* , 2002]Odell, James, Parunak, H. Van Dyke, Fleischer, Mitch, & Breuckner, Sven. 2002. Modeling Agents and their Environment. *Pages 16–31 of :* Giunchiglia, Fausto, Odell, James, & Weiss, Gerhard (eds), *Agent-Oriented Software Engineering (AOSE) III.* Lecture Notes on Computer Science, vol. 2585. Springer, Berlin.

[Okuyama *et al.* , 2005]Okuyama, Fabio Y., Bordini, Rafael H., & da Rocha Costa, Antônio Carlos. 2005. ELMS : An Environment Description Language for Multi-Agent Simulations. *Pages 24–33 of :* [Weyns *et al.* , 2005a].

[Orcutt, 1957]Orcutt, Guy. H. 1957. A New Type of Socio-Economic Systems. *The Review of Economics and Statistics*, **58**, 773–797.

[Parker, 2001]Parker, Miles T. 2001. What is Ascape and Why Should You Care ? *The Journal of Artificial Societies and Social Simulation, JASSS*, **4**(1).

[Parunak & Odell, 2002]Parunak, H. Van Dyke, & Odell, James. 2002. Representing social structures in UML. *Pages 1–16 of : Agent-Oriented Software Engineering II.* Lecture notes in computer science LNCS, vol. 2222. Springer.

[Parunak *et al.* , 1998]Parunak, H. Van Dyke, Savit, Robert, & Riolo, Rick L. 1998. Agent-Based Modeling vs. Equation-Based Modeling : A Case Study and Users' Guide. *In :* Schiman, Jaime S., Conte, Rosaria, & Gilbert, Nigel (eds), *Proceedings of the* 1^{st} *Workshop on Modelling Agent Based Systems, MABS'98*, vol. LNAI 1534. Lecture Notes in Artificial Intelligence LNAI, Springer-Verlag, Berlin.

[Parunak *et al.* , 2003]Parunak, H. Van Dyke, Breuckner, Sven, Fleischer, Mitch, & Odell, James. 2003. A Design Taxonomy of Multi-Agent Interactions. *Pages 123–137 of :* Paolo Giorgini, Jörg P. Müller, James Odell (ed), *Agent-Oriented Software Engineering IV : 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, 2003, Revised Papers.* Lecture notes in computer science LNCS, vol. 2935. Springer.

[Phan, 2002]Phan, Denis. 2002 (24 Mai). *Régimes et changements structurels. Essais d'économie historique.* Thèse de Doctorat, université d'Orléans.

[Pollack & Ringuette, 1990]Pollack, Martha, & Ringuette, Marc. 1990. Introducing the Tileworld : experimentally evaluating agent architectures. *Pages 183–189 of :* Dietterich, Thomas, & Swartout, William (eds), *Proceedings of the Eighth National Conference on Artificial Intelligence.* Menlo Park, CA : AAAI Press.

[Praehofer *et al.* , 1993]Praehofer, H., Auering, F., & Reisinger, G. 1993. An Environment for DEVS-Based Multi-Formalism Simulation in Common Lisp/CLOS. *Discrete Event Dynamic Systems : Theory and Applications*, **3**(2/3), 119–149.

[Rao & Georgeff, 1992]Rao, Anand S., & Georgeff, Michael P. 1992 (October 25-29). An Abstract Architecture for Rational Agents. *Pages 439–449 of :* Nebel, Bernhard, Rich, Charles, & Swartout, William R. (eds), *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92).*

[Resnick, 1990]Resnick, Mitchel. 1990. MultiLogo : A Study of Children and Concurrent Programming. *Interactive Learning Environments*, **1**(3).

[Resnick, 1994]Resnick, Mitchel. 1994. *Turtles, termites, and traffic jams : explorations in massively parallel microworlds.* MIT Press.

[Ricordel & Demazeau, 2002]Ricordel, Pierre-Michel, & Demazeau, Yves. 2002. Volcano, a Vowels-Oriented Multi-agent Platform. *Pages 253–262 of : Revised Papers from the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems.* Springer-Verlag.

[Riolo *et al.* , 2001]Riolo, Rick L., Cohen, Michael D., & Axelrod, Robert. 2001. Evolution of cooperation without reciprocity. *Nature*, **411**(6862), 441–443.

[Rouchier, 2003]Rouchier, Juliette. 2003 (April). Re-implementing John Duffy's model of speculative learning agents in a small scale society : Problems, interest and issues. *In : Model to Model Workshop M2M.*

[Rozenblit & Zeigler, 1993]Rozenblit, Jerzy W., & Zeigler, Bernard P. 1993. Representing and constructing system specifications using the system entity structure concepts. *Pages 604–611 of : Proceedings of the 25th conference on Winter simulation.* ACM Press.

[Russell & Norvig, 2003]Russell, Stuart J., & Norvig, Peter. 2003. *Artificial Intelligence : A Modern Approach.* 2 edn. Pearson Education.

[Sargent, 2001]Sargent, Robert G. 2001. Verification and validation : some approaches and paradigms for verifying and validating simulation models. *Pages 106–114 of : Proceedings of the 33nd conference on Winter simulation.* IEEE Computer Society.

[Shannon, 1976]Shannon, Robert E. 1976. Simulation modeling and methodology. *Pages 9–15 of : Proceedings of the 76 Bicentennial conference on Winter simulation.*

[Shannon, 1998]Shannon, Robert E. 1998. Introduction to the art and science of simulation. *Pages 7–14 of : Proceedings of the 30th conference on Winter simulation.* IEEE Computer Society Press.

[Sichman *et al.* , 1994]Sichman, Jaime Simão, Conte, Rosaria, Castelfranchi, Cristiano, & Demazeau, Yves. 1994. A Social Reasoning Mechanism Based On Dependence Networks. *Pages 188–192 of :* Cohn, A. G. (ed), *Proceedings of the Eleventh European Conference on Artificial Intelligence.* Chichester : John Wiley & Sons.

[Simonin, 2001]Simonin, Olivier. 2001 (20 décembre). *Le modèle satisfaction-altruisme : coopération et résolution de conflits entre agents situés réactifs, application à la robotique.* Thèse de Doctorat, Université Montpellier II.

[Simonin & Ferber, 2000]Simonin, Olivier, & Ferber, Jacques. 2000. Modeling Self Satisfaction and Altruism to handle Action Selection and Reactive Cooperation. *Pages 314–323 of :* Meyer, Jean Arcady, Berthoz, Alain, Floreano, Dario, Roitblat, Herbert, & Wilson, Stewart W. (eds), *From Animals to Animats 6, the sixth International Conference on Simulation of Adaptive Behavior SAB'00.* Cambridge, Massachussets, USA : MIT Press.

[Simonin & Ferber, 2001]Simonin, Olivier, & Ferber, Jacques. 2001. Modélisation des satisfactions personnelle et interactive d'agents situés coopératifs. *Pages 215–226 of :* Seghourouchni, Amal El Fallah, & Magnin, Laurent (eds), *IX^{èmes} Journées Francophones pour l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents JFIADSMA'01.* Hermès, Paris.

[Simonin *et al.* , 2002]Simonin, Olivier, Michel, Fabien, Chapelle, Jérôme, & Ferber, Jacques. 2002. Un simulateur de systèmes multi-robots dans MadKit. *Pages 167–170 of :* Mathieu, Philippe, & Müller, Jean-Pierre (eds), *X^{èmes} Journées Francophones pour l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents JFIADSMA'02.* Hermès, Paris.

[Smith, 1980]Smith, Reid G. 1980. The Contract Net Protocol. *IEEE Transactions on Computers*, **C29**(12), 1104–1113.

[Soulié, 2001]Soulié, Jean-Christophe. 2001 (3 Décembre). *Vers une approche multi-environnements pour les agents.* Thèse de Doctorat, Université de la Réunion.

[Steels, 1990]Steels, Luc. 1990. Cooperation Between Distributed Agents Through Self-Organisation. *Pages 175–196 of :* Demazeau, Yves, & Müller, Jean-Pierre (eds), *Decentralized A.I. : Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-89), Cambridge, England, August.* Amsterdam, North-Holland : Elsevier Science Publishers.

[Steels, 1995]Steels, Luc. 1995. The Biology and Technology of Intelligent Autonomous Agents. *Robotics and Autonomous Systems*, **15**(1-2).

[Stratulat, 2002]Stratulat, Tiberiu. 2002 (13 décembre). *Systèmes d'agents normatifs : concepts et outils logiques.* Thèse de Doctorat, université de Caen.

[Terna, 2001]Terna, Pietro. 2001. Creating Artificial Worlds : A Note on Sugarscape and Two Comments. *The Journal of Artificial Societies and Social Simulation, JASSS*, **4**(2).

[Theodoropoulos & Logan, 1999]Theodoropoulos, Georgios, & Logan, Brian. 1999. A framework for the distributed simulation of agent-based systems. *Pages 58–65 of :* Szczerbicka, Helena (ed), *Modelling and Simulation : a tool for the next millenium, Proceedings of the 13th European Simulation Multiconference (ESM'99)*, vol. 1. SCS, Society for Computer Simulation International.

[Travers, 1996]Travers, Michael D. 1996 (June). *Programming with Agents : New metaphors for thinking about computation.* Ph.D. thesis, Massachusetts Institute of Technology, MIT.

[Uhrmacher, 2001a]Uhrmacher, Adelinde M. 2001a. Dynamic structures in modeling and simulation : a reflective approach. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, **11**(2), 206–232.

[Uhrmacher, 2001b]Uhrmacher, Adelinde M. 2001b. A System Theoretic Approach to Constructing Test Beds for Multi-Agent Systems. Springer-Verlag New York, Inc.

[Uhrmacher & Schattenberg, 1998]Uhrmacher, Adelinde M., & Schattenberg, Bernd. 1998. Agents in Discrete Event Simulation. *Pages 129–136 of :* Bargiela, Andre, & Kerckhoffs, Eugene (eds), *10TH European Simulation Symposium "Simulation in Industry – Simulation Technology : Science and Art" (ESS'98).* Nottingham, UK : SCS Publications, Ghent, for The Society for Computer Simulation International (SCS).

[Vanbergue, 2003]Vanbergue, Diane. 2003 (12 décembre). *Conception de simulation multi-agents : Application à la simulation des migrations intra-urbaines de la ville de Bogota.* Thèse de Doctorat, Université de Paris VI, Paris.

[Vanbergue *et al.* , 2000]Vanbergue, Diane, Treuil, Jean-Pierre, & Drogoul, Alexis. 2000. Modelling urban phenomena with cellular automata. *Advances in Complex Systems*, **3, 1-4**, 127–140.

[Vincent *et al.* , 2001]Vincent, Regis, Horling, Bryan, & Lesser, Victor R. 2001. An Agent Infrastructure to Build and Evaluate Multi-Agent Systems : The Java Agent Framework and Multi-Agent System Simulator. *Lecture Notes in Artificial Intelligence : Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems.*, **1887**(January).

[Wagner *et al.* , 2003]Wagner, Tom, Guralnik, Valerie, & Phelps, John. 2003. A key-based coordination algorithm for dynamic readiness and repair service coordination. *Pages 757–764 of : Proceedings of the second international joint conference on Autonomous agents and multiagent systems.* ACM Press.

[Weiss, 2000]Weiss, Gerhard (ed). 2000. *Multiagent systems : a modern approach to distributed artificial intelligence.* MIT Press.

[Weiss *et al.* , 2003]Weiss, Gerhard, Rovatsos, Michael, & Nickles, Matthias. 2003. Capturing agent autonomy in roles and XML. *Pages 105–112 of : Proceedings of the second international joint conference on Autonomous agents and multiagent systems.* ACM Press.

[Weyns & Holvoet, 2003a]Weyns, Danny, & Holvoet, Tom. 2003a. Model for Simultaneous Actions in Situated Multi-Agent Systems. *In : First German Conference on Multi-Agent System Technologies, MATES 03.* to appear in LNAI volume, Springer-Verlag, Berlin.

[Weyns & Holvoet, 2003b]Weyns, Danny, & Holvoet, Tom. 2003b. Regional Synchronization for Simultaneous Actions in Situated Multi-Agent Systems. *Pages 497–511 of :* Marik, Vladimir, Müller, Jörg P., & Pechoucek, Michal (eds), 3^{rd} *International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003*, vol. 2691. Lecture Notes in Computer Science LNCS, Springer-Verlag, Berlin.

[Weyns *et al.* , 2005a]Weyns, Danny, Parunak, H. Van Dyke, & Michel, Fabien (eds). 2005a. *Environments for Multi-Agent Systems, First International Workshop, E4MAS 2004, New York, NY, USA, July 19, 2004, Revised Selected Papers.* LNAI, vol. 3374. Springer.

[Weyns *et al.* , 2005b]Weyns, Danny, Parunak, H. Van Dyke, Michel, Fabien, Holvoet, Tom, & Ferber, Jacques. 2005b. Environments for Multiagent Systems : State-of-the-Art and Research Challenges. *Pages 1–47 of :* [Weyns *et al.* , 2005a].

[Wood, 1986]Wood, David O. 1986. MIT model analysis program : what we have learned about policy model review. *Pages 248–252 of : Proceedings of the 18th conference on Winter simulation.* ACM Press.

[Wooldridge, 2000]Wooldridge, Michael. 2000. Intelligent agents. *Chap. 1, pages 27–77 of :* Weiss, Gerhard (ed), *Multiagent systems : a modern approach to distributed artificial intelligence.* MIT Press.

[Wooldridge & Ciancarini, 2000]Wooldridge, Michael, & Ciancarini, Paolo. 2000. Agent-Oriented Software Engineering : The State of the Art. *Pages 1–28 of :* Ciancarini, Paolo, & Wooldridge, Michael (eds), *First Int. Workshop on Agent-Oriented Software Engineering*, vol. 1957. Springer-Verlag, Berlin.

[Wooldridge & Jennings, 1995]Wooldridge, Michael, & Jennings, Nicholas R. 1995. Intelligent agents : Theory and practice. *The Knowledge Engineering Review*, **10**(2), 115–152.

[Zambonelli & Parunak, 2002]Zambonelli, Franco, & Parunak, H. Van Dyke. 2002. From design to intention : signs of a revolution. *Pages 455–456 of : Proceedings of the first international joint conference on Autonomous agents and multiagent systems.* ACM Press.

[Zeigler, 1972]Zeigler, Bernard P. 1972. Toward a Formal Theory of Modeling and Simulation : Structure Preserving Morphisms. *Journal of the ACM (JACM)*, **19**(4), 742–764.

[Zeigler & Ören, 1986]Zeigler, Bernard P., & Ören, Tuncer I. 1986. Multifaceted, multiparadigm modeling perspectives : tools for the 90's. *Pages 708–712 of :* Wilson, J., Henrickson, J., & Roberts, S. (eds), *Proceedings of the 18th conference on Winter simulation (WSC 86).* ACM Press.

[Zeigler *et al.* , 2000]Zeigler, Bernard P., Kim, Tag Gon, & Praehofer, Herbert. 2000. *Theory of Modeling and Simulation.* Academic Press, Inc.

Liste des tables

Liste des figures

Index

Formalisme, outils et éléments méthodologiques pour la modélisation et la simulation multi-agents

Résumé

Cette thèse aborde des questions liées à la simulation informatique de systèmes complexes modélisés à l'aide du paradigme multi-agents. Cette approche repose sur l'idée qu'il est possible de représenter directement le comportement et les interactions d'un ensemble d'entités autonomes évoluant dans un environnement commun. Mal maîtrisée, la complexité des modèles considérés pose encore aujourd'hui de nombreux problèmes quant à leur élaboration et à leur implémentation. L'une des conséquences les plus critiques réside dans la difficulté de reproduire fidèlement les modèles publiés dans la littérature. Ce qui pose le problème de la vérification et de la validation de ce type de simulations. Dans cette thèse, nous considérons que ce problème est en majeure partie lié à un manque de correspondance entre les spécifications des modèles et les structures informatiques qui permettent de les exécuter. Le but de nos travaux est de faire un pas vers une mise en correspondance effective de ces deux parties. Pour cela, nous considérons deux approches complémentaires. La première se focalise sur l'élaboration d'outils génériques de conception de simulateurs. Il s'agit de rendre plus explicites les structures informatiques utilisées pour l'implémentation. La deuxième repose sur l'étude des moyens aujourd'hui utilisés pour modéliser un système multi-agents. Dans ce cadre, nous proposons une réflexion globale sur ce paradigme et nous identifions un ensemble de contraintes liées à son utilisation. A partir de cette réflexion, nous proposons un principe de modélisation basé sur l'approche Influence/Réaction et sur l'utilisation d'un modèle formel d'environnement pour systèmes multi-agents appelé MIC*. Nous montrons en quoi notre approche permet de respecter les différentes contraintes de modélisation identifiées et nous illustrons sa faisabilité à l'aide d'un exemple concret.

Mots-clés : modélisation et simulation informatique, systèmes multi-agents, principe Influence/-Réaction, interaction.

Formalism, tools and methodological elements for the modeling and simulation of multi-agents systems

Abstract

This thesis addresses questions related to the modeling and simulation of complex systems which are based on the multi-agents system (MAS) paradigm. Multi-Agent Based Simulations (MABS) rely on the idea that it is possible to directly represent the behavior and the interactions of a set of autonomous entities situated in a common environment. One critical issue is the difficulty to accurately achieve replication of published models. This raises problems for verification and validation of MABS. In this thesis, we advocate that MABS verification and validation problems are mainly related to the gap that exists between models specifications and the computational structures which are used to execute them. The motivation of our work is to take steps toward an effective mapping between models specifications and concrete software structures considering two complementary approaches. The first approach proposes generic simulator engineering tools. The idea is to make explicit the computational structures which are used for the implementation. The second approach relies on the study of today MAS modeling means : we propose a reflexion about the MAS paradigm and we identify a set of modeling constraints which are related to it. We then propose modeling principles based on (1) the Influence/Reaction approach and (2) a formal model of the environment for MASs, namely MIC*. We show how our approach handles the various identified modeling constraints and we illustrate its feasibility through a concrete example.

Keywords : modeling and simulation, multi-agents systems, Influence/Reaction model, interaction.