Vers une approche multi-environnements pour les agents

Jean-Christophe Soulié

Université de La Réunion

Institut de Recherche en Mathématiques et Informatique Appliquées

Vers une approche multi-environnements pour les agents

THÈSE

présentée et soutenue publiquement le 3 Décembre 2001

pour obtenir le grade de

Docteur de l'Université de La Réunion

Spécialité Informatique

par

Jean-Christophe Soulié

Composition du jury

Président :	Henri Ralambondrainy	Professeur – Université de La Réunion
Rapporteurs :	Jean-Pierre Müller	Cadre de Recherche – CIRAD Montpellier
	Joël Quinqueton	Directeur de Recherche – INRIA Montpellier
Examinateurs :	Christophe Lepage	Cadre de Recherche – CIRAD Montpellier
	Jean-Michel Stretta	Docteur d'État – Représentant IRD La Réunion
Directeur de thèse :	Pierre Marcenac	Professeur – Université de La Réunion

Ecole Doctorale Interdisciplinaire de l'Université de La Réunion

Mis en page avec la classe thloria.

Remerciements

Je tiens à remercier mon directeur de thèse, Pierre Marcenac, pour m'avoir proposé ce sujet. Grâce à la très grande autonomie et liberté qu'il m'a laissées, j'ai pu, petit à petit, découvrir les joies mais également les difficultés de la recherche.

Jean-Pierre Müller et Joël Quinqueton m'ont fait l'honneur de relire et de juger mes travaux. Je tiens à les remercier vivement pour leurs commentaires qui m'ont permis de considérer mes travaux sous des angles nouveaux.

Christophe LePage a été à la base de bon nombre de mes idées. Lors de mon séjour dans l'équipe Green du Cirad Montpellier, nos échanges ont été très riches. Enfin, comment ne pas se souvenir de toutes les bonnes soirées passées ensemble autour d'un bon verre de rhum arrangé... Pour tout cela, je tiens à le remercier.

Jean-Michel Stretta a suivi l'avancée de nos travaux sur la plate-forme Muffins. Ses remarques pertinentes, tant sur les résultats que sur mes lacunes en halieutique, ont permis d'améliorer Muffins et mes connaissances en halieutique.

Après quatre années passées au sein de l'Iremia, j'ai peur d'oublier toutes les personnes qui ont pu croiser mon chemin. Si par mégarde certains ne se trouvent pas dans cette liste, c'est purement fortuit, n'en prenez pas ombrage. Néanmoins, je tiens tout particulièrement à remercier :

- Pierre Gigord, directeur de l'Iremia à l'époque, grâce à qui j'ai pu avoir ma bourse de thèse.
- Mohamed Rochdi, directeur de l'Iremia actuellement, qui m'a permis d'organiser ma soutenance dans les meilleurs conditions.
- Rémy Courdier pour toutes ses remarques et ses commentaires sur ce travail.
- Toute la bande de joyeux thésards de l'Iremia : Serge Colin, Stéphane Calderoni, Ivan Kojadinovic, David Grosser, Guillaume Rousse, Etienne Payet et Jean-Dany Vally. Pour certains, notre rencontre remonte à la licence d'informatique... Que de chemin parcouru depuis ! Nous avons souvent passé de bons moments ensemble (même lorsque Mr White has left the game). Les nombreuses Lan party que nous avons effectuées resteront d'excellent souvenirs.
- Pierre Brondeau pour sa bonne humeur permanente et pour ses bonnes blagues du matin.
- Marie-Catherine Daniel-Vatonne dont les conseils m'ont été précieux.
- Alexandre Gouverneur dont la connaissance de MySql est inépuisable. J'en ai usé et abusé, mais sa disponibilité a été constante.

Comment ne pas remercier David Guyomard (mon compagnon office, comme il se définit lui même) de l'IRD Réunion ? Nous avons passé ensemble un nombre incalculable

d'heures pour développer MUFFINS. Sa bonne humeur inébranlable est vraiment appréciable. Enfin, sa réserve de café « équitable » était toujours disponible pour enrayer nos coups de fatigue.

Lors de mon passage dans l'équipe GREEN du CIRAD Montpellier, outre Christophe LEPAGE, j'ai tout particulièrement apprécié les échanges que j'ai pu avoir avec François BOUSQUET, Innocent BAKAM, Pierre BOMMEL, Emmanuel LIEURAIN et Martine ANTONA.

François GUERRIN (alias Francesco) du CIRAD Réunion qui avec sa bonne humeur constante et ses bons conseils m'a été d'une aide précieuse.

Je tiens tout particulièrement à remercier Christian LANDRY qui, en 1992, a su me faire confiance à une époque où il n'était pas question de faire une thèse, mais plutôt de « sauver les meubles ».

Je remercie mes parents et ma proche famille pour m'avoir soutenu durant mes longues études ; sans oublier Maïté et Léopold.

Enfin, je tiens ici à exprimer toute ma gratitude envers mon épouse qui a supporté un informaticien en début, pendant et en fin de thèse (une espèce plutôt grincheuse !) sans trop s'énerver. Son soutien inconditionnel depuis bientôt 10 ans m'est précieux. Pour tout cela, je lui décerne un doctorat en patience et compréhension avec la mention très honorable assortie des félicitations du jury.

Pour finir, j'aimerais conclure ces remerciements en ayant une petite pensée pour ma fille Manon qui voue une véritable passion pour les versions préliminaires de ma thèse. Ces feuilles sont une source inépuisable de gribouillages et de jeux. Certainement un de mes relecteurs les plus assidus !

Je dédie cette thèse à Manon et à sa maman.
En espérant me faire pardonner mes absences répétées...

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Le domaine de l'informatique en tant que science peut se définir, en suivant la publication de l'Académie Française [JO:82], comme étant « la science du traitement rationnel de l'information considérée comme le support des connaissances dans les domaines scientifiques, économiques et sociaux, notamment à l'aide de machines automatiques ». Depuis sa création en tant que science, l'informatique a toujours suscité soit de la passion, de la peur, de l'envie ou de la frustration selon que l'on fasse partie ou non des initiés à cette science ô combien ésotérique.

Or, de tous temps dans l'histoire de l'informatique, les scientifiques ont toujours eu comme rêve de pouvoir rendre ces machines « intelligentes ». C'est-à-dire que nous aimerions pouvoir avoir en face de nous des machines qui pensent comme nous, qui réagissent comme nous et qui agissent comme nous. C'est pourquoi, l'intelligence artificielle a toujours été une branche majeure de l'informatique. De nombreuses avancées remarquables ont été réalisées et malheureusement aussi beaucoup de promesses n'ont pas été tenues.

Comment rendre ces machines intelligentes ? Immense défi qui représente le saint Graal de bien des scientifiques ! Nous cherchons sans cesse à concevoir de nouveaux modèles, de nouveaux concepts. Parmi ceux-ci, la thématique que nous allons aborder est celle de l'intelligence artificielle distribuée et plus spécialement les systèmes multi-agents. Depuis quelques années, les systèmes multi-agents ont pris une place de plus en plus importante dans l'informatique. Que ce soit les systèmes multi-agents vus comme outils pour la vie artificielle, la robotique collective, ... ou les systèmes multi-agents vus comme entités logicielles permettant de réaliser des tâches de manière autonomes dans des réseaux distribués.

Néanmoins, dans le cadre de la vie artificielle, on assiste à un essor de l'utilisation des systèmes multi-agents pour modéliser et simuler des phénomènes naturels complexes ou des phénomènes sociaux qui ne peuvent pas être modélisé à l'heure actuelle, soit par une complexité trop grande du phénomène étudié, soit par une méconnaissance du phénomène étudié (en sociologie par exemple). La problématique qui va être développée dans ce document découle d'une question toute simple : « comment modéliser les agents afin qu'ils puissent évoluer simultanément dans des environnements (*e.g* dans des mondes)

[JO:82] Journal Officiel du 17 janvier 1982. – http://www.journal-officiel.gouv.fr/, 1982.

différents ? ». Bien que triviale au premier abord, cette question est d'actualité dans bien des problématiques de recherche qui nous sont proposées. Dès que l'on parle de société d'agents, c'est-à-dire un certain nombre d'agents évoluant ensemble avec des buts propres, cette question apparaît tout naturellement : l'agent doit pouvoir gérer en même temps au moins son environnement social (les agents avec lesquels il cohabite) et son environnement physique dans lequel il évolue. Et de quelle manière des actions d'un de ces environnements peuvent avoir une influence sur un autre et inversement.

Pour ce faire, ce travail est décomposé en 6 chapitres :

Le premier chapitre positionne notre travail par rapport à des travaux réalisés au sein d'une équipe. Il présente l'historique des travaux menés et la réflexion qui nous a conduit à être confronté à cette problématique. Enfin, il se situe par rapport à un atelier logiciel déjà existant et dans lequel ce travail doit pouvoir venir s'intégrer.

Le deuxième chapitre consistera classiquement à présenter l'état de l'art dans le domaine des systèmes multi-agents utilisés comme outils pour simuler des phénomènes naturels ou sociaux.

Le troisième chapitre, pose, dans une première partie, la terminologie utilisée tout au long de ce travail, présente la définition d'un agent et d'un système multi-agents que nous avons utilisé. Il présente la construction de ces entités dans notre formalisme et en fonction de nos définitions.

Le quatrième chapitre présente, dequelle manière il est possible, à partir des définitions posées dans le chapitre précédent, d'étendre ce modèle à une approche multi-environnements. Ce modèle multi-environnemental va donc être construit itérativement en reprenant une à une toutes les caractéristiques présentées dans le chapitre précédent. De plus, des discussions sur les choix que nous avons fait sont réalisées afin de pouvoir illustrer quel a été le cheminement entrepris lors de ce travail.

Le cinquième chapitre va, quant à lui, présenter l'aspect implémentation du modèle multi-environnements décrit dans le chapitre précédent. Cette implémentation a pour but de définir une plate-forme générique qui permette de développer et de simuler des applications en utilisant des environnements multiples.

Enfin, le sixième chapitre décrit une application de la plate-forme présentée précédemment. L'application est celle du déplacement d'espadons dans la zone sud de l'océan Indien. Cet exemple est issu de travaux concrets menés conjointement avec l'IFREMER Réunion et l'IRD Réunion. Plus particulièrement, il fait partie intégrante du projet « palangre 2001 » mené par l'IRD Réunion. Pour ce faire, nous utiliserons des images satellites issues de divers satellites passant au dessus de la zone concernée et nous permettant d'avoir une description de l'environnement à un moment donné. Les aspects purement halieutiques, ont été pris en charge par des doctorants de l'IRD Réunion et les résultats que nous fournissons lors de ces simulations sont directement utilisés par les halieutes.

Enfin, il faut noter que ce projet n'est pas destiné à fournir une meilleure information pour les pêcheurs réunionnais, mais vise à devenir un outil de décision et un outil pédagogique afin de pouvoir obtenir une meilleure gestion des ressources renouvelables dans la zone sud de l'océan Indien.

Enfin, il nous restera à revenir sur ce travail et envisager les pistes de recherche qui permettraient d'améliorer notre modèle.

Chapitre 1

Positionnement

Sommaire

1.1 Cadre de recherche

De part sa situation géographique, ses spécificités géographiques ainsi que ses spécificités géophysiques, l'Ile de La Réunion est soumise à une forte probabilité d'apparition de catastrophes naturelles. Par exemple, lors de la saison cyclonique, on peut compter, en moyenne, une dizaine de systèmes dépressionnaires tropicaux dans le Sud-ouest de l'océan Indien [Met00] ou bien encore les récentes éruptions du Piton de la Fournaise en 1999, 2000 et tout au long de l'année 2001, ainsi que l'activité sismique liée à ce phénomène, illustrent tout à fait les risques liés à ces phénomènes. De plus, l'Ile de La Réunion des part ses caractéristiques, présente un écosystème très spécifique, très particulier et donc par conséquence très fragile. C'est pourquoi, depuis quelques années, les décideurs régionaux ont lancé, et soutenu financièrement, un grand nombre de projets visant de protection et de sauvegarde de l'environnement. Citons, par exemple, le projet de l'herbier de La Réunion, le projet de sauvegarde des récifs coralliens, ...

Tous les phénomènes que nous avons décrit ci-dessus, sont très complexes et prennent en compte un grand nombre de paramètres. Parfois même, les experts peuvent ne pas connaître l'ensemble des paramètres du phénomène étudié (c'est le cas pour l'étude du

[Met00] Météo-France à La Réunion : Systèmes Dépressionnaires Tropicaux. – http://www.meteo.fr, 2000.

comportement du volcan notamment où les géophysiciens ne peuvent pas fournir une description complète des paramètres qui rentrent en jeux lors d'une éruption volcanique). C'est pourquoi, on peut classer tous ces problèmes en tant que systèmes complexes. Or, de part leur nature intrinsèquement non linéaire, les systèmes complexes ne permettent pas aux approches classiques de l'Intelligence Artificielle d'être efficaces car il est extrêmement difficile (voire même impossible dans certains cas) de pouvoir décrire le système dans sa globalité ainsi que sa dynamique d'ensemble. L'approche par agents permet de pallier à cette limite, car elle fournit un modèle de calcul qui vise à représenter les phénomènes localement, dans lequel l'émergence d'un comportement global est le résultat d'interactions entre les agents. De plus, le fait de modéliser un système complexe comme un réseau d'agents en interactions permet de simplifier grandement la représentation et la modélisation du système.

1.2 Historique des travaux

Compte tenu de cette situation, l'équipe « système multi-agents »[1] de l'Université de La Réunion s'est investie depuis les cinq dernières années dans la modélisation et la simulation de systèmes complexes à l'aide de systèmes multi-agents. Tout d'abord, le premier domaine d'application des systèmes multi-agents a été le domaine des Systèmes Tutoriels Intelligents. Ensuite, une étude a été menée sur l'application d'aide à la résolution de problèmes en géométrie [Lem96b]. Puis, pour une étude sur les éruptions volcaniques du Piton de la Fournaise en collaboration avec l'Institut de Physique du Globe de Grenoble [GMC+96, Mar97] appelée GEOMAS[2]. Puis les systèmes multi-agents ont été utilisés pour modéliser l'apparition de séismes [CM97]. Enfin, une dernière utilisation des systèmes multi-agents a été réalisée dans le cadre d'un partenariat avec de CIRAD–Réunion visant à modéliser l'échange des effluents d'élevage à l'échelle d'une localité rurale sur le site

1. MAS[2], acronyme de MultiAgent Systems, Modelling And Simulating
2. acronyme de GEOphysics and MultiAgent Systems

[Lem96b] Stéphane LEMAN. – *TREMMA : TRansfert d'Expertise avec un Modèle Multi-Agents. Un modèle multi-agents pour la représentation dynamique des connaissances et des raisonnements d'un apprenant.* – Saint Denis, Ile de La Réunion, Thèse de Doctorat, Université de La Réunion, 1996.

[GMC+96] Sylvain GIROUX, Pierre MARCENAC, Stéphane CALDERONI, David GROSSER et Jean-Robert GRASSO. – A report of a case study with agents in simulation. *In : Proceedings of the 1st International Conference on Pratical Applications of Intelligent Agents and MultiAgent Technology.* – London, UK, avril 1996.

[Mar97] Pierre MARCENAC. – Modélisation de systèmes complexes par agents. *Technique et Sciences Informatiques – Hermes*, 1997.

[CM97] Stéphane CALDERONI et Pierre MARCENAC. – Emergence of Earthquakes by MultiAgent Simulation. *In : Proceedings of the 11th European Simulation MultiConference*, éd. par Ali Riza KAYLAN et Axel LEHMANN. pp. 665–669. – Society for Computer Simulation, Istambul, Turkey, juin 1997.

de Grand-Ilet à La Réunion [GCC+98b, GCC+98a] appelée BIOMAS. De ces diverses expérimentations et grâce à un processus en spirale récursive [Boe86, CMG98] basée sur une méthodologie <Objectifs, Expériences, Abstraction, Application>, une architecture générique pour la conception, la modélisation et la simulation de systèmes complexes à l'aide de système multi-agents a été défini. La Figure 1.1 illustre la démarche et la réflexion menée pour aboutir à cette architecture.

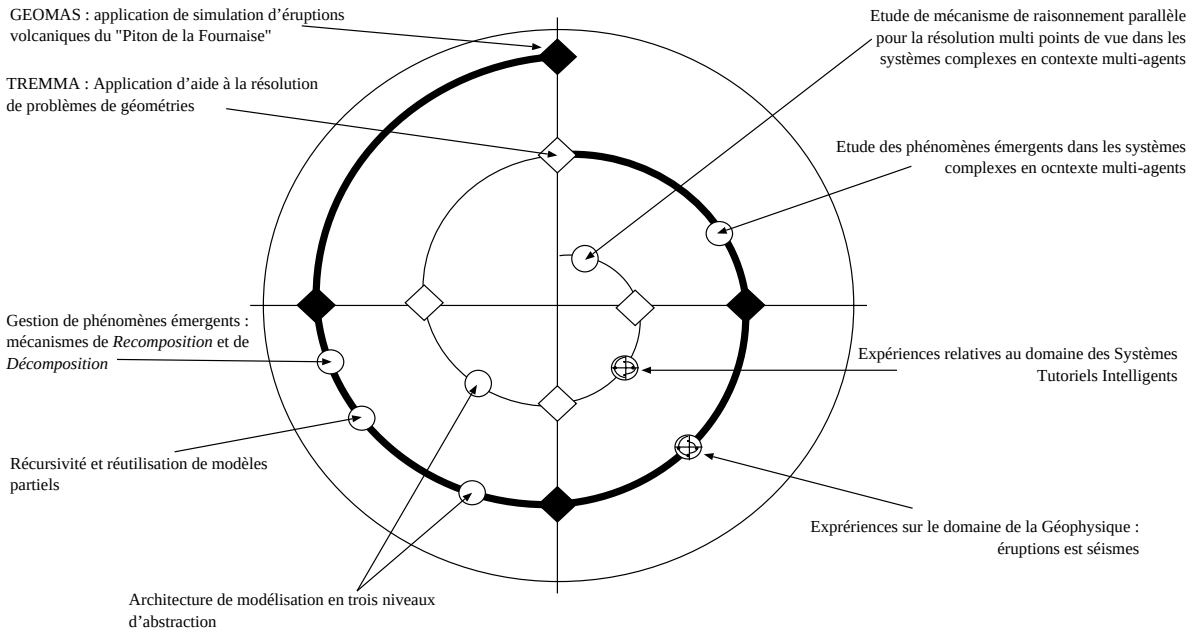


FIG. 1.1 – *La décomposition en spirale*

Cette architecture appelée GEAMAS[3], en est à sa troisième version. Dans un premier temps, GEAMAS a été écrite en Smalltalk [RG89] et a permis de réaliser les simulations

3. acronyme de GEneric Architecture for MultiAgents Simulations

[GCC+98b] François GUERRIN, Rémy COURDIER, Stéphane CALDERONI, Jean-Marie PAILLAT, Jean-Christophe SOULIÉ et Jean-Dany VALLY. – Conception d'un modèle multi-agents pour la gestion des effluents d'elevage à l'echelle d'une localité rurale. *In : Actes des* 6^e *Jounées Francophones en Intelligence Artificielle Distribuée et Systèmes Multi-Agents.* pp. 25–37. – Hermes, Nancy, France, novembre 1998.

[GCC+98a] François GUERRIN, Rémy COURDIER, Stéphane CALDERONI, Jean-Marie PAILLAT, Jean-Christophe SOULIÉ et Jean-Dany VALLY. – Biomas : un modèle multi-agents pour aider à la gestion négociée d'effluents d'elevage. *In : Actes du* 1^{er} *Colloque sur les Modèles et Systèmes Multi-Agents pour la Gestion de l'Environnement et des Territoires.* – CEMAGREF, Clermont-Ferrand, France, octobre 1998.

[Boe86] B.W BOEHM. – *A Spiral Model of Development and Enhancement.* – Software Engineering Note, 1986.

[CMG98] Rémy COURDIER, Pierre MARCENAC et Sylvain GIROUX. – Application d'une méthodologie en spirale au développement d'une plate-forme multi-agents en smalltalk. *L'Objet – Hermes*, vol. 1, n° 4, 1998.

[RG89] D. ROBSON et A. GOLDBERG. – *Smalltalk-80, The Language.* – Addison Wesley, Publishing Company, 1989.

concernant le volcan et les tremblements de terre. Ensuite, GEAMAS a été entièrement revue et réécrite en Java, c'est la version 2 [Sou97, MC99, SMCC98]. Cette version a été utilisée pour réaliser le début du projet BIOMAS. Puis, et toujours grâce à la méthodologie en spirale présentée ci-dessus, nous nous sommes aperçus que GEAMAS ne pouvait pas prendre en compte la notion de rôle pour un agent dans un système multi-agents. De cette réflexion, est née la version 3 de GEAMAS. A l'heure actuelle, GEAMAS en est à la version 4. Il s'agit d'une version probatoire utilisée au travers de différentes expérimentations (Enchères, BIOMAS, ...).

De plus, GEAMAS a été pensée comme une architecture modulaire et ouverte. Elle est composée d'un noyau minimal qui implante uniquement les mécanismes liés aux agents (autonomie, indépendance, ...) et ceux liés aux systèmes multi-agents (émergence, société d'agents, ...). De ce noyau minimal, l'architecture à trois niveaux développée par P. Marcenac [MG98] a été rajoutée en tant que surcouche. De la même manière, le travail de thèse de S. Calderoni concernant l'apprentissage individuel et collectif des agents dans les systèmes multi-agents [CM99] est considéré comme un service qui vient s'ajouter en plus de GEAMAS ; ainsi que les travaux de thèse de JD. Vally [VC98] concernant la notion de rôle des agents. La Figure 1.2 page suivante présente cette architecture.

Comme présenté dans la Figure 1.2 page ci-contre, GEAMAS, ainsi que ses interfaces graphiques permettant de suivre l'évolution d'une simulation, et ses surcouches sont développées comme des composants logiciels qui permettent d'être utilisés en fonction des desiderata des utilisateurs pour leurs simulations.

[Sou97] Jean-Christophe SOULIÉ. – *Conception et Implantation d'une Plate-Forme d'Agents en Java.* – Thèse de 3ème cycle, Laboratoire d'Informatique d'Avignon, Université d'Avignon et des Pays de Vaucluse, 1997.

[MC99] Pierre MARCENAC et Rémy COURDIER. – *Java Agent-Oriented Development Environment.* – John Wiley & Sons Books, 1999, *Journal of Object-Oriented Application Frameworks, M. Fayad.*

[SMCC98] Jean-Christophe SOULIÉ, Pierre MARCENAC, Stéphane CALDERONI et Rémy COURDIER. – GEAMAS v2.0: An object oriented platform for complex systems simulations. *In : Proceedings of the* 26^{th} *International Conference on Technology of Object-Oriented Languages and Systems – TOOLS USA'98*, éd. par M. SINGH, B. MEYER, J. GIL et R. MITCHELL. pp. 230–242. – IEEE Computer Society Press, Santa Barbara, California, USA, août 1998.

[MG98] Pierre MARCENAC et Sylvain GIROUX. – *GEAMAS: A Generic Architecture for Agent-Oriented Simulations of Complex Processes.* – International Journal of Applied Intelligence, Kluwer Academic Publishers, 1998.

[CM99] Stéphane CALDERONI et Pierre MARCENAC. – Mutant: a Genetic Learning System. *In : Proceedings of the* 12^{th} *Australian Joint Conference on Artificial Intelligence (AI-99)*, éd. par Norman FOO, *Lecture Notes in Artificial Intelligence*, volume 1747. – Springer, Berlin, décembre 1999.

[VC98] Jean-Dany VALLY et Rémy COURDIER. – A Conceptual "Role-Centered" Model for Design of Multi-Agents Systems. *In : Proceedings of First Pacific Rim International Workshop on Multi-Agents – PRIMA'98*, éd. par Toru ISHIDA, *Lecture Notes in Artificial Intelligence*, volume 1599. pp. 33–46. – Springer Verlag, Berlin, novembre 1998.

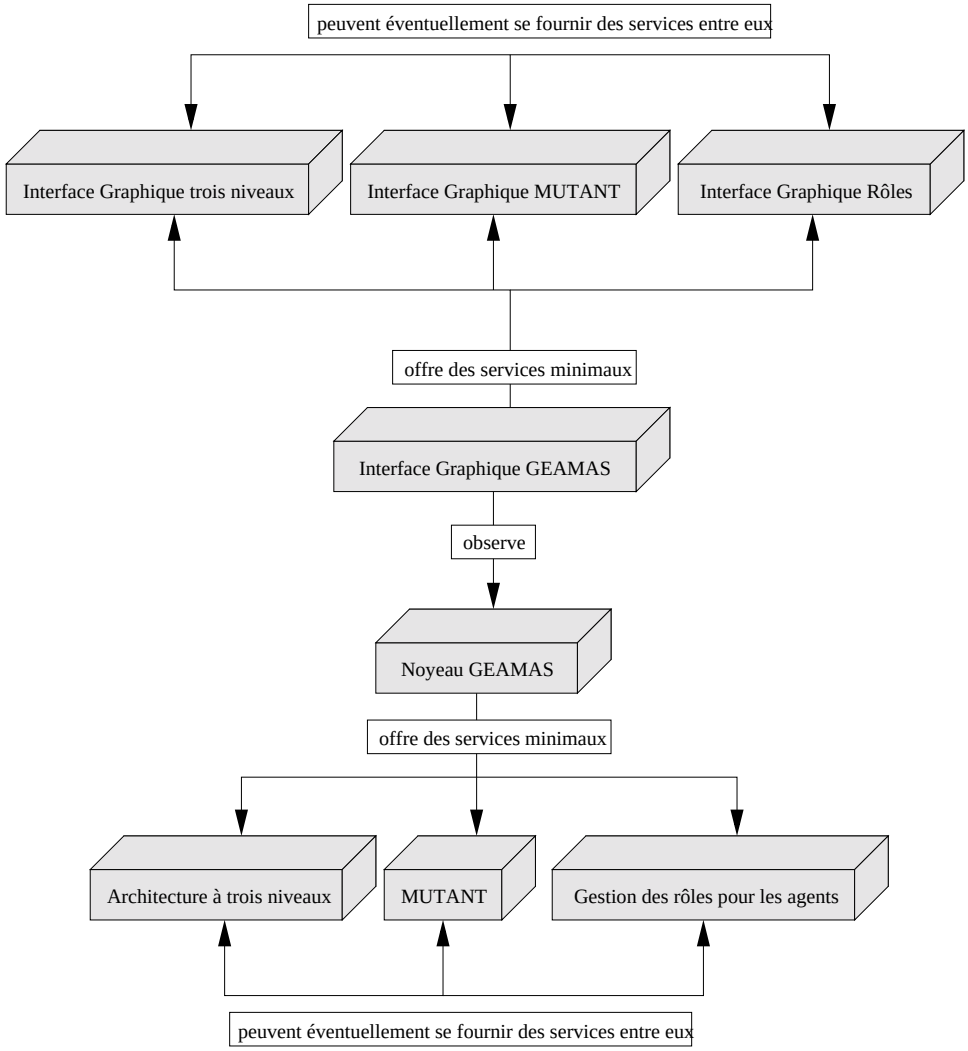


FIG. 1.2 – *L'architecture modulaire du projet* GEAMAS

1.3 Position des travaux de thèse

Les travaux réalisés dans cette thèse sont la conséquence d'une remarque toute simple qui a été mise en évidence lors des travaux préliminaires sur BIOMAS. Au tout début, en effet, nous nous sommes aperçus que GEAMAS n'était pas prévu pour prendre en compte, et donc simuler, des environnements artificiels issus d'environnements du monde réel. En effet, lors des simulations précédentes construites sur la base de GEAMAS, aucun besoin de représentation de l'environnement ne s'était fait sentir. En fait, GEAMAS avait été défini comme un système multi-agents *purement communiquant* et non pas comme un système multi-agents *situé*. C'est pourquoi ce travail de thèse est entièrement consacré à l'environnement dans les systèmes multi-agents. Comme nous l'avons décrit ci-dessus, les différents travaux menés dans l'équipe MAS[2] concernent l'apprentissage individuel et collectif des agents et les méthodologies de conception des systèmes multi-agents. Dans

le premier cas, S. Calderoni a développé pour ses besoins un modèle d'environnement totalement dédié à la simulation d'animats. Concernant le deuxième travail de thèse, JD. Vally a seulement besoin de pouvoir s'appuyer sur un modèle d'environnement pour pouvoir l'utiliser dans ses travaux de méthodologie de conception. De plus, ces travaux de cette thèse ont été aussi motivés par un nouveau projet réalisé conjointement avec l'IRD [4] de la Réunion et l'IFREMER [5] visant à modéliser le comportement (et plus précisément les trajectoires) de bancs de thons et d'espadons. Pour ce faire, nous avons à notre disposition un certain nombre de données concernant l'environnement. C'est pourquoi ce projet a été utilisé comme base de travail pour cette thèse.

En conclusion, les travaux présentés dans cette thèse s'inscrivent idéalement dans le projet GEAMAS car ils correspondent à un réel besoin au niveau de cette plate-forme. Comme GEAMAS n'était pas capable de pouvoir gérer les environnements intégrant des données environnementales réelles, il a donc fallu fournir un modèle d'environnement capable de pouvoir gérer cela. Le modèle présenté dans cette thèse n'est pas seulement un modèle d'environnement comme on peut en trouver dans toutes les plates-formes multi-agents actuelles. L'intérêt de la proposition faite dans ce modèle repose sur les aspects suivants :

Un modèle interactionnel entre l'agent et l'environnement C'est-à-dire que lorsqu'un agent perçoit des phénomènes dans l'environnement et agit sur celui-ci, on peut exhiber des mécanismes bien précis permettant de mettre en œuvre les interactions entre l'environnement et l'agent.

Un modèle multi-environnemental C'est-à- dire qu'il est proposé dans cette thèse un modèle d'agent et de système multi-agents qui puisse prendre en compte le fait qu'un agent puisse évoluer dans des mondes (*e.g* des environnements) différents. C'est notamment le cas dans notre projet d'expérimentation concernant les déplacements des bancs d'espadons et de thons ou bien encore dans les sciences sociales. Ce modèle n'est en aucun cas un modèle qui vient se substituer à celui présenté auparavant, c'est une surcouche de celui-ci, car nous sommes dans le modèle de GEAMAS et nous devons pouvoir fournir à un utilisateur des services modulaires.

Pour finir, la Figure 1.3 page suivante illustre l'apport de ces travaux de thèse dans le modèle GEAMAS.

4. Institut de Recherche pour le Développement
5. Institut Français de Recherche pour l'Exploitation de la MER

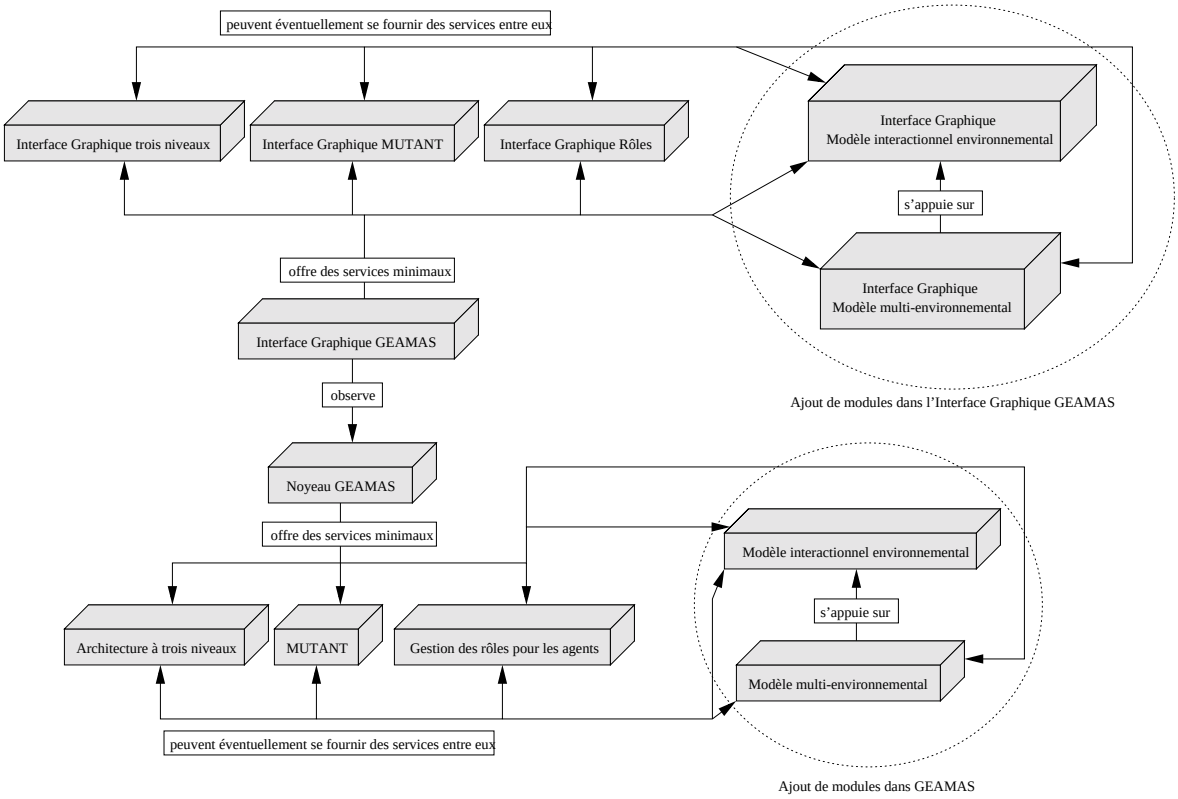


FIG. 1.3 – *La nouvelle architecture de* GEAMAS

Chapitre 2

État de l'art

Sommaire

2.1 Introduction

Bien que l'environnement dans les systèmes multi-agents n'ai pas été souvent très bien décrit [Fer95] et a fortiori formalisé, de nombreuses applications ou plate-formes qui avaient pour but de simuler des comportements naturels, des gestions de ressources communes, des comportements de populations ou de vie artificielle ont été réalisées. On peut

[Fer95] Jacques Ferber. – *Les Systèmes Multi-Agents – Vers une intelligence collective.* – iia – InterEditions, 1995.

citer, par exemple, les plate-formes suivantes : CORMAS [BBaPP98], MOBIDYC [GL98], MANTA [Dro93, DCF93], RIVAGE [PC97] ou encore SWARM [MBLA96]. Ces différentes propositions diffèrent dans bien des domaines, que ce soit par le langage d'implémentation, la philosophie agent ou bien encore le but recherché. Néanmoins, elles développent des caractéristiques communes pour la modélisation de l'environnement que l'on peut classer dans trois catégories : l'environnement centralisé, distribué ou l'environnement modélisé comme un agent (connu aussi sous le nom de « tout agent » [MBLA96]).

2.2 L'environnement centralisé

Dans un environnement centralisé, c'est une seule et unique structure qui définit l'environnement. Cette structure, une sorte de bloc, représente et contient tous les éléments de l'environnement. Les agents perçoivent l'environnement par l'intermédiaire d'un lien qui les relie à celui-ci. Les agents accèdent à ce bloc par l'intermédiaire de demandes qu'ils formulent à l'environnement. Celui-ci répond ensuite à ces demandes en renvoyant les informations relatives à cette demande. La figure 2.1 page ci-contre issue de [Fer95] illustre ce mécanisme.

L'environnement centralisé présente comme principal défaut le fait qu'il s'adapte mal à une augmentation de sa complexité. En effet, lorsque l'environnement devient plus fourni, il faut pouvoir gérer toutes les entités qui interagissent dans celui-ci en même temps et on arrive vite à une explosion combinatoire des processus à gérer. Néanmoins, l'environnement centralisé a été utilisé dans de nombreuses applications et pour en citer

[BBaPP98] François BOUSQUET, Innocent BAKAM, Hubert PROTON et Christophe Le PAGE. – Cormas: Common-pool ressources and multi-agents system. *In : Proceedings of the* 11^{th} *International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, éd. par A.P del POBIL et M. ALI, *Lecture Notes in Artificial Intelligence*, volume 1416. pp. 826–838. – Springer Verlag, juin 1998.

[GL98] Vincent GINOT et Christophe LE PAGE. – Mobidyc, a generic multi-agents simulator for modeling populations dynamics. *In : Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, éd. par A.P del POBIL et M. ALI, *Lecture Notes in Artificial Intelligence*, volume 1416. pp. 805–814. – Springer Verlag, juin 1998.

[Dro93] Alexis DROGOUL. – *De la Simulation Multi-Agents à la Résolution Collective de Problèmes.* – Paris, France, Thèse de Doctorat, Université Pierre et Marie Curie, 1993.

[DCF93] A. DROGOUL, B. CORBARA et D. FRESNEAU. – Manta: New experimental results on the emergence of (artificial) ant societies. *In : Simulating Societies Symposium*, éd. par C. CASTELFRANCHI. – 1993.

[PC97] E. PERRIER et C. CAMBIER. – *Une Approche Multi-Agents pour simuler les Interactions entres les Acteurs Hétérogènes de l'infiltration et du Ruissellement d'eau sur une Surface de Sol*, chap. unknown. – Elsevier, 1997, *Tendances Nouvelles en Modélisation.*

[MBLA96] N. MINAR, R. BURKHART, C. LANGTON et M. ASKENAZI. – *The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations.* – Rapport de recherche, Santa Fe Institute, 1996.

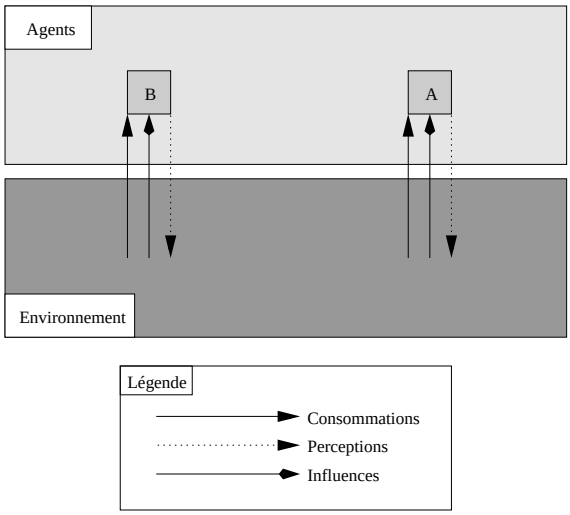


FIG. 2.1 – *Représentation schématique d'un environnement centralisé*

quelques unes : CRAASH [ZF93] et MACE [GBH87].

2.3 L'environnement distribué

L'environnement distribué est constitué d'un ensemble de cellules (ou bien encore appelées « patch ») qui forment une grille. Chaque cellule peut être vue comme un petit environnement centralisé. Par exemple, dans le cas du déplacement d'un agent, si celui-ci effectue un déplacement qui ne le fait pas sortir de sa cellule, il va effectuer le déplacement à l'intérieur de celle-ci ; dans le cas contraire, il va falloir d'abord que l'agent change de cellule et ensuite se positionne de manière adéquate dans sa nouvelles cellule. Les cellules peuvent contenir des agents ou des objets qui sont des éléments de l'environnement. Ces cellules mises les unes à côté des autres forment une surface et c'est cette surface qui représente l'environnement. Dans la plupart des cas, cette surface définit un plan Cartésien, mais on peut imaginer d'autres représentations (en trois dimensions, en graphe, ...). La figure 2.2 page suivante montre la représentation d'un environnement distribué.

De manière plus formelle, on peut dire que l'environnement est défini comme suit : soit un ensemble E de n cellules, on note E_i (avec $0 < i \leq n$) le i-ème élément de E et on a : $Environnement = \bigcup_{i=1}^{n} E_i$. De plus, à partir de la définition précédente, il est possible de définir le voisinage d'une cellule, ainsi que sa connexité. Pour ce faire, il faut

[ZF93] K. ZEGHAL et J. FERBER. – A coordinated collision avoidance system. *In : Proceedings of the European Simulation Multiconference*. – Lyon, France, 1993.

[GBH87] L. GASSER, C. BRAGANZA et N. HERMAN. – *MACE: A Flexible Testbed Distributed AI Research*, pp. 119–152. – Distributed Artificial Intelligence, 1987.

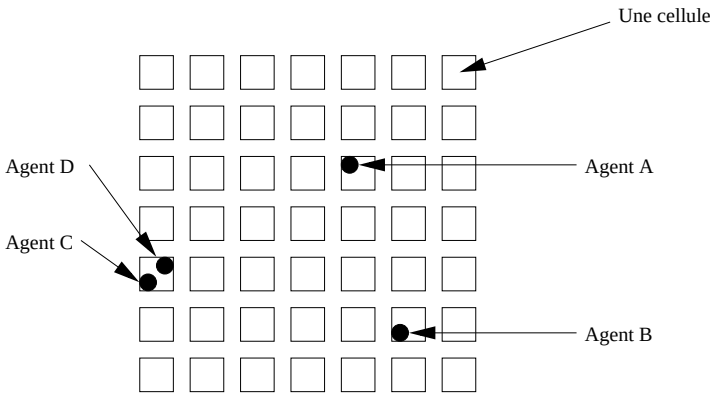


FIG. 2.2 – *Un environnement distribué et ses agents*

définir une fonction appelée *voisins* : elle renvoie un sous-ensemble de E qui contient le voisinage d'une cellule E_i. Pour la connexité, cela correspond juste au nombre de cellules qui sont en voisinage direct avec E_i. Par exemple, pour un environnement de connexité 4, la fonction $voisins(E_i)$ relativement à un point donné p de coordonnées (x,y) va renvoyer toutes les cellules du type : $\{(x+1,y),(x,y+1),(x-1,y),(x,y-1)\}$ et de même, pour un environnement de connexité 8, la fonction $voisins(E_i)$ va renvoyer, en plus de celles citées ci-dessus, les cellules du type : $\{(x+1,y+1),(x-1,y+1),(x-1,y-1),(x+1,y-1)\}$. De cette manière, il est possible de modifier le nombre de cellules que peut percevoir un agent autour de lui. Ces modifications auront pour conséquence le fait que en fonction de la connexité définit à un instant t par le concepteur ou l'utilisateur, l'agent va pouvoir ou pas percevoir des phénomènes ou des informations qui peuvent être vitales pour son évolution. Il est évident que la meilleure solution dans ce cas, serait que l'agent puisse à tout moment connaître l'ensemble des états de l'environnement. Malheureusement, hormis le fait que cela entraînerait des infaisabilités informatiques (du fait de la complexité éventuelle de l'environnement), cela casserait une partie de la définition de J. Ferber [Fer95] concernant les agent :

> *On appelle agent une entité physique ou virtuelle*
> ...
> *qui est capable de percevoir (mais de manière limitée) son environnement*
> ...

Ces définitions permettent donc de modéliser un environnement comme étant la composition de cellules qui elles mêmes peuvent contenir des agents ou des objets. Cette définition de l'environnement est très largement utilisée dans les systèmes multi-agents aujourd'hui car elle permet de mixer les deux approches : approche centralisée et approche distribuée. Chaque petite cellule est un petit environnement centralisé, mais comme sa taille est considérablement réduite par rapport à un environnement global centralisé, les problèmes de complexité, notamment au niveau de la mémoire, sont ici considérablement réduits.

2.4 L'environnement vu comme un agent

Cette forme de modélisation a été initiée par C. Langton dans leur fameuse plate-forme SWARM [MBLA96], et plus récemment par J. Ferber et O. Gutknecht dans MADKIT [FG98, GF97]. Cette manière de décrire l'environnement consiste, comme son nom l'indique, à tout représenter sous forme d'agents. Tous les objets de l'environnement deviennent des agents, ainsi que l'entité qui leur sert de support (d'environnement). Les agents « objets de l'espace » qui vont réaliser des interactions possèdent des liens d'accointances[6] entre eux.

Enfin, dans MADKIT, qui est une plate-forme de systèmes multi-agents multiples, le système qui représente l'ensemble de l'environnement et de ses composantes, peut-être vu comme un simple agent dans un autre système. Cela permet donc de faire interagir des agents appartenant à d'autres systèmes avec un seul et unique agent qui représente un environnement complet.

2.5 Un mot sur la dynamique de l'environnement

Selon l'étude réalisée, l'environnement du système multi-agents associé peut être plus ou moins dynamique. Par le terme dynamique, on entend que l'état de l'environnement, ainsi que les entités qui le compose, évolue à chaque pas de simulation en fonction de lois. Ces lois peuvent être des phénomènes extrêmement bien définis ou décrits par les scientifiques. Dans ce cas, on pourra exprimer à l'aide d'équations ou d'automates ces transitions. Si ces lois ne sont que le reflets de constatations in situ de l'évolution de l'environnement, il faudra alors faire des abstractions très fortes pour pouvoir reproduire le comportement de l'environnement dans le système.

2.5.1 La dynamique dans un environnement centralisé

Une approche classique pour représenter la dynamique de l'évolution de l'environnement dans un environnement centralisé consiste à utiliser une représentation par auto-

6. dans le cas d'un graphe non orienté, on n'aura qu'un seul lien entre deux agents et dans le cas d'un graphe orienté, on aura deux liens entre deux agents

[FG98] Jacques FERBER et Olivier GUTKNECHT. – A meta-model for the analysis and design of organizations in multi-agents systems. *In : Proceedings of the International Conference on Multi-Agents Systems – ICMAS'98*. pp. 128–135. – IEEE Computer Society Press, Paris, France, juillet 1998.

[GF97] Olivier GUTKNECHT et Jacques FERBER. – *MadKit: Organizing Heterogeneity in a Platform for Multiple Multi-Agents Systems*. – Rapport de recherche n° LIRMM97189, Laboratoire d'Informatique, de Robotique et de Micro-électonique de Montpellier, décembre 1997.

mates à états finis. Cette dynamique sera donc codée par des états et des transitions. C'est à dire qu'à un instant donné t, l'environnement est dans un état δ_t. Son état δ_{t+1}, à l'instant $t+1$, dépendra donc de la transition existant entre δ_t et δ_{t+1}. Cette méthode consiste alors à définir un automate à états finis comme tel :

- Un ensemble fini d'états Q qui va inclure l'état initial ainsi que les états finaux ;
- Un ensemble fini E de symbole d'entrée ;
- Un état initial q_0 $(q_0 \in Q)$;
- Un ensemble T d'états finaux $(T \subset Q)$;
- Une fonction de transition $d : Q \times (E \cup \{\varepsilon\}) \rightarrow P(Q)$, où ε représente le mot vide.

Par exemple, si on modélise l'environnement comme une parcelle de terrain avec une culture quelconque dessus, on aura l'automate suivant :

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_{fin}\}$,
- $E = \{friche, stade_1, stade_2, récolte, repos\}$,
- q_0,
- $T = \{q_{fin}\}$,
- d définie de la manière suivante :

$$\begin{aligned} d(q_0, friche) &\rightarrow q_1 \\ d(q_1, stade_1) &\rightarrow q_2 \\ d(q_2, stade_2) &\rightarrow q_3 \\ d(q_3, récolte &\rightarrow q_4 \\ d(q_4, repos) &\rightarrow q_{fin} \end{aligned}$$

Grâce à la fonction de transition d on a donc tous les états possibles de l'environnement. Or, à lecture de cet exemple, on constate que dans le cas d'environnements complexes et hétérogènes, l'expression et la description de cette fonction de transition d risque d'impliquer une explosion combinatoire. D'autant plus que l'on ne possède pas toujours la connaissance exacte des étapes par lesquelles passe l'environnement (dans le cas d'un volcan, par exemple, les géophysiciens ne savent pas décrire exactement les différentes étapes de l'évolution du volcan).

2.5.2 La dynamique dans un environnement distribué

Comme présenté dans la section 2.3 page 15, l'environnement distribué peut être vu comme une composition d'un ensemble d'environnements centralisés partiels. Le problème

ici est de savoir comment utiliser ces cellules pour reproduire l'éventuelle dynamique de l'environnement. Deux solutions classiques sont utilisées : la première consiste à représenter l'environnement sous la forme d'un automate cellulaire et la deuxième consiste à utiliser des objets et de représenter la dynamique aux travers d'envoi de messages objet par invocation de méthodes.

La dynamique représentée à l'aide d'automates cellulaires

Un automate cellulaire [FTW84] est un système de cellules interagissant localement de manière simple et qui manifeste un comportement global complexe. S. Wolfram a par ailleurs souligné une autre propriété importante : il s'agit de modèles dynamiques où espace, temps et états sont discrets. Ceci signifie que l'espace est divisé en cellules considérées comme des entités individuelles, que celles-ci peuvent prendre plusieurs états possibles et enfin, qu'elles sont susceptibles de changer d'état à des moments fixes selon une règle de transition fondée sur la configuration d'états au voisinage de chaque cellule.

Les éléments de base d'un automate cellulaire sont les cellules. Une cellule peut être considérée comme une mémoire unitaire souvent qualifiée d'état. Dans le modèle le plus simple d'un automate cellulaire, les états sont binaires, c'est-à-dire qu'ils contiennent les valeurs 0 ou 1. En revanche, dans des modèles plus complexes, les cellules peuvent prendre plusieurs états différents (≥ 2).

Les deux exemples les plus connus sur les automates cellulaires sont :

- *L'automate de Conway*, plus connu sous le nom de *Jeu de la vie*. Cet automate est à deux dimensions et deux états. Chaque cellule peut donc être soit égale à 1 (vivante), soit égale à 0 (morte), en fonction de l'état de ses 8 cellules voisines. La règle de transition est simple et se résume de la façon suivante :

  R1 Si une cellule est entourée par moins de « 2 » cellules vivantes, alors elle meurt (état 0).

  R2 Si une cellule est entourée par « 2 » cellules vivantes, alors elle conserve son état.

  R3 Si une cellule est entourée par « 3 » cellules vivantes, alors elle devient vivante (état1).

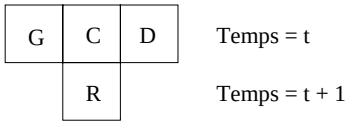
  R4 Si une cellule est entourée par plus de « 3 » cellules vivantes, alors elle meurt (état 0).

  En d'autres termes, la naissance d'une cellule a lieu lors de la rencontre de trois congénères, la mort provient pour cause d'isolement(moins de « 2 » cellules vivantes) ou de surpopulation (plus de « 3 » cellules vivantes). Malgré la simplicité et le caractère déterministe de ces règles, il est pratiquement impossible de prédire le

[FTW84] D. Farmer, T. Toffoli et S. Wolfram. – *Cellular Automata.* – New York, 1984.

résultat de leur application sur un motif arbitraire, sans effectivement dérouler le résultat ou l'exécuter sur un ordinateur.

– *L'automate cellulaire de Wolfram.* L'automate de Wolfram est un automate à deux états et à une dimension. Le déroulement du temps est suivant l'axe vertical. Les cellules sont disposées sur une ligne ; chaque cellule C a une voisine à gauche et une voisine à droite. Chaque état nouveau se traduit par une ligne nouvelle. Pour passer d'une ligne à la suivante on applique à chaque unité de temps une règle de transformation qui tient compte de l'état précédent des cellules :



Le codage binaire donne ainsi un nombre à trois chiffres pour choisir l'état nouveau. Par exemple :

	111	110	101	100	011	010	001	000
Règle 30	0	0	0	1	1	1	1	0

Où 0 correspond à l'absence de cellule et 1 à sa présence. On peut noter que la règle 30 génère du chaos quelque soient les conditions initiales.

Concernant l'environnement dans les systèmes multi-agents, l'utilisation qu'il en est faite ressemble beaucoup au jeu de la vie décrit ci-dessus. L'environnement est, en effet, composé de n cellules qui sont toutes interconnectées entre elles. Le fonctionnement de celui-ci, dans l'optique de la modélisation de la dynamique de l'environnement, est le suivant : à un instant donné t, l'état d'une cellule C est fonction de l'état à l'instant $t-1$ des m cellules[7] avec lesquelles C est connectée et de son état propre à l'instant $t-1$. On voit donc que l'évolution des objets contenus dans une cellule est bien fonction de ce qui s'est passé « avant » au sein de l'environnement. Et comme les cellules sont reliées entre elles, toute modification au sein d'une cellule C pourra éventuellement impliquer une modifications au sein d'une cellule C'.

Néanmoins, ce style de modélisation pour l'environnement peut poser des problèmes selon que l'on effectue les modifications de l'environnement de manière asynchrone ou synchrone. Ces deux options sont, en effet, envisageables.

– Le cas du fonctionnement synchrone est le seul qui permette d'obtenir une évolution satisfaisante des objets au sein d'un environnement à base d'automate cellulaire. C'est à dire que chaque cellule, ainsi que les objets qu'elle contient, va évoluer en

7. m peut prendre pour valeur : 2, 4, 6, ... Cela va dépendre du degré de connexité qu'il aura été défini pour l'environnement simulé

parallèle. Toutes les cellules vont donc avoir une évolution autonome. Néanmoins, comme on se trouve dans un automate cellulaire, chaque cellule va donc aller chercher les informations qui lui conviennent auprès des autres cellules qui se trouvent dans son voisinage. Et c'est à ce niveau que se situe le principal problème de ce type de modélisation de l'environnement. Celle-ci, en effet oblige chaque cellule (ou une autre entité qui aura la charge de ce mécanisme) à conserver dans une mémoire tous ses états pour tous les temps t_i de la simulation afin qu'une cellule qui se trouve dans un état e_{t_i} à un instant t_i de la simulation puisse trouver les informations relatives à l'instant t_i de la simulation dans une de ses cellules voisines, même si celle-ci se situe à un temps t_j et donc qu'elle est dans un état e_{t_j}. Dans ce cas, même si l'on est gagnant en terme de représentativité de l'évolution de l'environnement, on perd par contre de façon significative en terme de complexité en espace pour la simulation du fait de l'obligation de garder en mémoire tous les états de chaque cellule pour l'ensemble de la simulation.

– Dans le cas d'un fonctionnement asynchrone, il faut choisir une cellule qui va initier le passage d'un temps t à un temps $t + 1$. Le choix de cette cellule est crucial car c'est par elle que l'on va commencer pour chaque évolution de l'environnement. Néanmoins, bien que tout à fait valide de par son concept, cette méthode ne semble pas tout à fait représentative de ce qui peut se passer au sein d'un environnement réel. Un premier palliatif à cette version consiste à choisir, à chaque évolution de l'environnement, une cellule de manière aléatoire. C'est à dire que pour chaque passage d'un instant t à un instant $t + 1$, la cellule servant de base pour la diffusion des changements sera différente. De cette manière, on va pouvoir reproduire le comportement indéterministe de l'environnement. Néanmoins, et comme toujours en informatique, se pose le problème de la génération de l'aléatoire en informatique. On n'est absolument pas sûr que l'on ne va pas retomber sur la même séquence de sélection des cellules et dans ce cas, on se retrouve donc dans la situation précédente à plus ou moins long terme.

Ce dernier type de représentation de l'environnement est la plus couramment utilisé et il est donc très largement présent dans de nombreuses plate-formes de simulation multi-agents. C'est le cas notamment des plate-formes CORMAS de F. Bousquet [BBaPP98] et MOBIDYC de V. Ginot [GL98]. Finalement on peut noter que CORMAS permet d'utiliser les deux types de représentation : asynchrone et synchrone indifféremment.

La dynamique représentée à l'aide d'objets

Une autre manière de modéliser de environnements distribués consiste à définir une grille qui va représenter la spatialité de l'environnement et d'y insérer des objets qui eux vont représenter les composantes de l'environnement. Ces objets dans cet environnement sont passifs et les agents définis pour la simulation considérée sont situés dans cet espace (c'est ce que l'on appelle *un système multi-agents situé* [Fer95]). Ceux-ci perçoivent et

agissent sur cet environnement. La perception est effectuée à l'aide de senseurs et l'action se fait à l'aide d'actuateurs. Les agents, afin de se faire percevoir, émettent des stimuli qui sont perceptibles par les autres, permettant ainsi la perception de la proximité d'un agent ou d'un objet. Ce stimulus est caractérisé par une intensité et une atténuation qui va être proportionnelle à l'éloignement de la source du stimulus. La figure 2.3 illustre ce mécanisme d'atténuation.

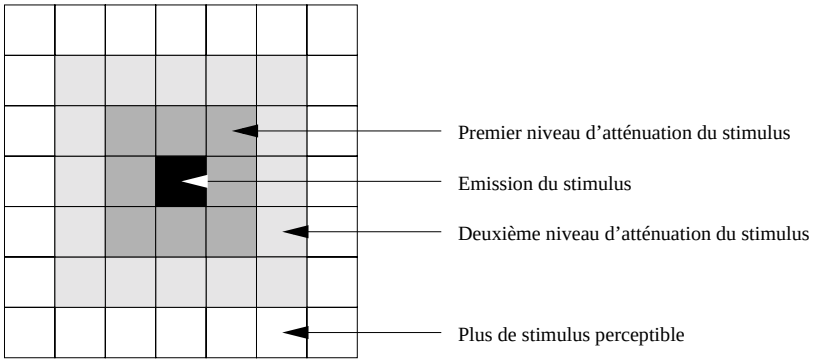


FIG. 2.3 – *L'atténuation d'un stimulus au sein d'un environnement*

Il est à noter que les objets passifs de l'environnement n'émettent pas de stimulus. Dans MANTA [DCF93], A. Drogoul utilise cette notion de diffusion de l'information dans l'environnement à l'aide de stimulus.

Toujours dans MANTA, les objets sont passifs dans l'environnement. Néanmoins, dès qu'un objet peut éventuellement posséder un aspect dynamique, celui-ci sera automatiquement transformé en un agent réactif ; c'est pourquoi les objets de l'environnement ne possèdent pas de possibilité en terme de dynamicité.

2.5.3 La dynamique dans un environnement vu comme un agent

Cette manière de représenter l'environnement consiste à considérer que l'environnement, pour les agents de la simulation considérée, est un autre agent de la simulation. Cet agent est bien entendu un peu « spécial » par rapport aux autres du fait de sa spécificité. Dans ce cas, la dynamicité de l'environnement est intrinsèque car du part la définition d'un agent : autonomie, indépendance, tendance propre, ... Si l'environnement est un agent il possède donc toutes ces capacités et donc il sera par définition dynamique.

Néanmoins, toujours du fait de la définition d'un agent, on peut se poser des questions quant à la validité de cette représentation. En effet, du fait l'environnement puisse éventuellement posséder des capacités d'autonomie et d'indépendance, il va donc pouvoir évoluer tout seul et il pourra même, par exemple ignorer les actions des agents au sein de l'environnement. Et dans ce cas, il y aura une perte au niveau de l'intégrité des don-

nées au sein de l'environnement. Afin de se prémunir de ce genre de problèmes, il faut donc « brider » l'agent environnement afin qu'il ne puisse pas faire ce qu'il veut dans son coin sans tenir compte des injonctions des autres agents. C'est à dire qu'il doit perdre sa capacité d'autonomie, et dans une moindre mesure sa capacité d'indépendance. Or, en réalisant cette opération, on ne peut plus parler d'agent en tant que tel. Dans le meilleur des cas, on va pouvoir parler d'acteur [Gir93] et dans le pire des cas on va se retrouver avec un objet environnement et on va donc être dans la situation présentée en 2.2 page 14 et 2.3 page 15.

Enfin, comme toute modification ou consultation de l'environnement entraîne de facto un envoi de message entre le (ou les) agent(s) concerné(s) et l'environnement, on peut se demander si cette modélisation n'entraîne pas un surcroît d'échange de messages au sein du système multi-agents considéré.

2.6 La représentation et le stockage des données environnementales

Dans le monde des systèmes multi-agents voués à la modélisation de phénomènes naturels, on a souvent besoin d'informations concrètes et d'observations réalisées sur le terrain, ainsi que d'éléments concernant la spatialité des zones couvertes par la simulation. C'est pourquoi, afin d'obtenir ce genre d'informations, on se tourne tout naturellement vers le monde des géographes qui, grâce à leurs travaux sur le terrain, ont emmagasiné un grand nombre de connaissances sous la forme des Modèles Numériques de Terrain (MNT) ou des Systèmes d'Information Géographique (SIGs) [DS96, Jon97].

2.6.1 Les Modèles Numériques de Terrain

Un Modèle Numérique de Terrain (MNT) est une représentation numérique simplifiée de la surface d'un territoire, en coordonnées altimétriques (le plus souvent exprimées en mètres par rapport au niveau de la mer) et planimétriques (calées dans un repère géographique).

Il existe, parmi une grande variété de représentations possibles d'une surface, deux modes largement utilisés pour les MNT qui s'apparentent aux deux modes de représenta-

[Gir93] Sylvain Giroux. – *Agents et Systèmes : une nécessaire unité.* – Thèse de Doctorat, Université de Montréal, Canada, 1993.

[DS96] J. Denèche et F. Salgé. – *Les Systèmes d'Information Géographique.* – Presses Universitaires de France, 1996.

[Jon97] C. Jones. – *Geographical Information Systems and Computer Cartography.* – Addison Wesley, 1997, 1st édition.

tion de l'information géométrique plane : le vecteur (polygones) ou raster (pixels). Dans le cas particulier du relief, les polygones utilisés sont des triangles (le polygone le plus simple pour représenter un élément de surface orienté dans l'espace).

Le mode Raster

Un MNT sous forme raster est aussi appelé matrice d'altitudes. Il s'agit d'un ensemble de valeurs numériques représentant des altitudes. Chaque altitude ainsi positionnée correspond à ce qui doit être alors considéré comme une altitude moyenne d'un élément de surface du terrain.

Cette distribution régulière de points définit alors un maillage de la surface du terrain, les dimensions de la maille (de fait, rectangulaire ou carrée) définissant ce qu'on appelle la résolution spatiale planimétrique du MNT. Chaque point est au centre d'une maille. Plus l'espacement des points est faible, plus la résolution est grande et plus le MNT est fin et riche en détails topographiques. La figure 2.4 représente un MNT avec des altitudes.

6	9	9	11	12	12	12	12	9	6
9	15	16	18	21	21	19	20	15	8
10	18	22	25	26	26	25	25	20	11
11	19	27	29	30	31	29	28	23	14
11	20	28	33	35	34	34	31	25	13
12	22	29	34	37	37	35	32	24	13
12	22	29	33	35	36	34	29	24	14
9	17	25	29	34	34	32	28	23	13
7	13	18	22	27	27	26	23	18	11
5	8	10	14	15	16	14	12	11	6

FIG. 2.4 – *Un MNT avec des altitudes*

Et les figures 2.5 page suivante, 2.6 page ci-contre et 2.7 page 26 illustrent trois représentations graphiques possibles à partir d'un MNT.

Les MNT en mode raster peuvent être reproduits de différentes manières, et à partir de sources différentes :

- photographies aériennes (très coûteux et long) ;
- courbes de niveaux numérisées (le plus simple, mais nécessite automatiquement des approximations) ;
- images numériques (le plus souvent des images satellites : cette méthode est pratiquement automatisée aujourd'hui et elle est assez efficace).

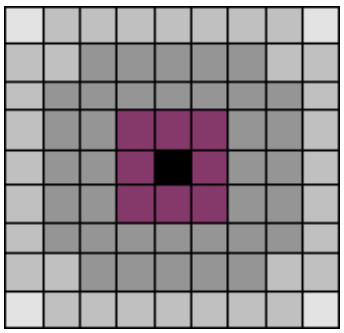


FIG. 2.5 – *Une représentation planaire*

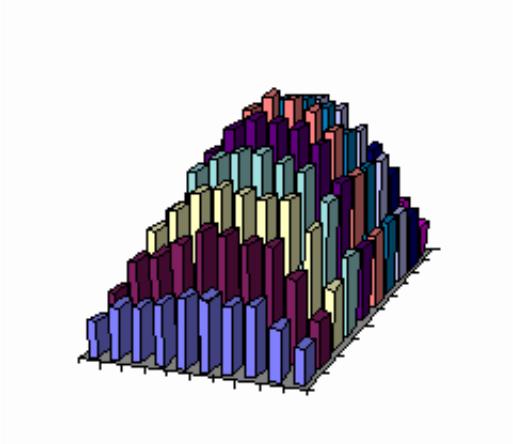


FIG. 2.6 – *Une représentation en 3 dimensions*

Le mode Vecteur

La seule alternative au mode raster est d'utiliser un « pavage » de la surface du terrain à l'aide de triangles. On parlera alors simplement de triangulation. Cette méthode est utilisée pour les MNTs depuis les années 70, c'est-à-dire dès l'arrivée des premiers SIGs vectoriels. De tels MNTs ont pour caractéristique principale de s'appuyer sur un semis de points de mesure (le plus souvent ceux-ci sont disposés de façon irrégulière et leurs densités peuvent augmenter en fonction de la complexité du relief et de la précision recherchée). Chaque point est relié à deux voisins pour former un réseau de triangles. Ce réseau ne doit laisser apparaître aucun « trou ». Il est tel qu'aucun des triangles n'en superpose un autre. On parle alors de TIN (de l'anglais : Triangular Irregular Network).

Différents algorithmes existent pour construire de tels pavages. On peut citer par exemple la triangulation de Delaunay [Pop93] ou la triangulation sous contraintes. La figure 2.8 page suivante illustre l'application de la triangulation de Delaunay sur un nuage

[Pop93] Ion POPESCU. – *Construction of Delaunay and Voronoï Diagram. Efficient Sequential and Parallel Implementation.* – Rapport de recherche n° 93-53, Johannes Kepler University, Linz, Austria, RISC-Linz, 1993.

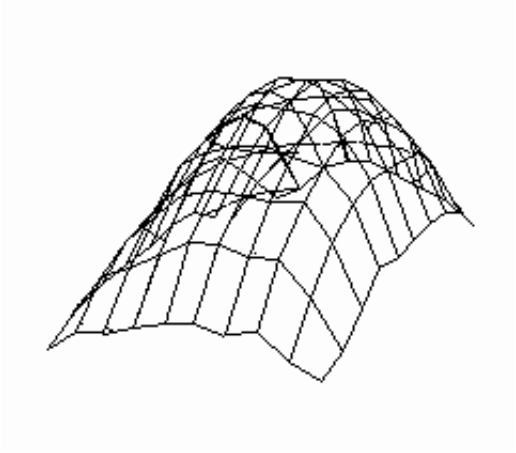


FIG. 2.7 – *Une autre représentation en 3 dimensions à l'aide d'un maillage*

de points.

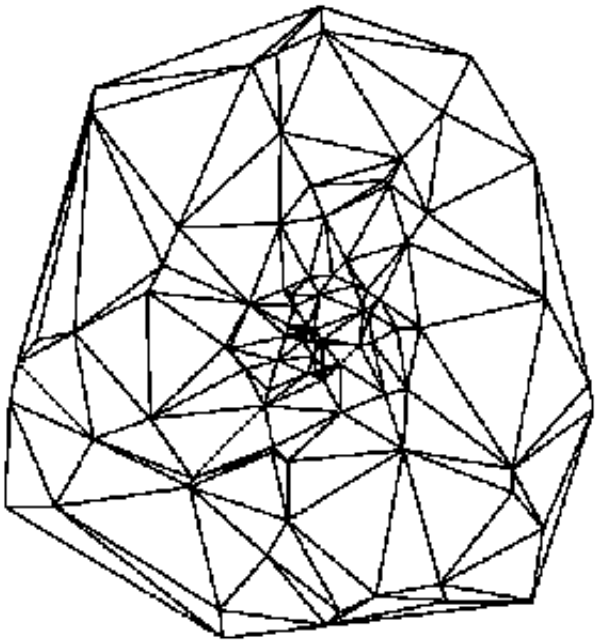


FIG. 2.8 – *Résultat de la triangulation de Delaunay sur un nuage de points*

2.6.2 Les Systèmes d'Information Géographique

Le concept de Système d'Information Géographique (SIG) est apparu dans le courant des années 80 sous l'impulsion d'un nouveau domaine d'étude et de recherche émergeant appelé *Géomatique*. Il existe plusieurs définitions pour les SIGs, mais on peut en dégager

deux principales [DS96] :

> « *Un SIG est un système informatique de matériels, de logiciels et de processus conçus pour permettre la collecte, la gestion, la manipulation et l'affichage de données à référence spatiale afin de résoudre des problèmes complexes d'aménagement et de gestion* »

Cette première définition est proposée par le comité fédéral de coordination inter-agences américain pour la cartographie numérique en 1988 ; et une deuxième ci-dessous est donnée par M. Didier en 1990 :

> « *Un SIG est un ensemble de données repérées dans l'espace, structurées de façon à pouvoir en extraire commodément des synthèses utiles à la décision* »

Concrètement, les SIGs sont des bases de données spatialisées et on définit ce qu'on appelle « un fond de carte » qui représente l'espace topographique. Dans la plupart des cas, ce fond de carte est un MNT. Sur ce fond de carte, viennent s'ajouter une succession de cartes. Ces cartes représentent des catégories d'informations spécifiques. Par exemple, une carte peut contenir un réseau routier d'une région, une autre les zones agricoles et enfin une dernière les zones d'habitations. Ces cartes venant s'empiler sur le fond de carte, on possède ainsi avoir une vision globale de l'ensemble des données, mais il est possible aussi de restreindre la vision à une ou plusieurs cartes. Enfin, on peut effectuer des requêtes verticales sur plusieurs cartes et ainsi obtenir des informations croisées.

2.7 Et le temps dans tout ça?

La notion de temps, dans les systèmes multi-agents est très importantes. Surtout si l'on considère des simulations concernant des phénomènes naturels, physiques ou sociaux.

Il existe trois notions bien distinctes pour exprimer le temps dans les systèmes multi-agents [FTPD98] :

- **Le temps réel** qui représente de temps pendant lequel le phénomène physique ou naturel est apparu ;

[FTPD98] E. Fianyo, Jean-Pierre Treuil, Edith Perrier et Yves Demazeau. – Multi-agent architecture integrating heterogeneous models for dynamical processes: The representation of time. *In : Proceedings of Multi-Agent Systems and Agent-Based Simulations – MABS'98.* – Paris, France, juin 1998.

- **Le temps virtuel** qui représente le temps réel à l'intérieur du simulateur multi-agents informatique ;
- **Le temps d'exécution** qui représente le temps que prend un simulateur multi-agents informatique pour fournir des résultats.

L'approche la plus classique consiste à faire coïncider le temps réel et le temps virtuel. Néanmoins, on préfère souvent raccourcir le temps virtuel et ainsi pouvoir obtenir un temps d'exécution qui soit plus court que le temps réel. Dans certains cas, il est, en effet, impensable de vouloir reproduire un temps réel exact. Par exemple, dans le cadre d'une simulation concernant le comportement d'un volcan, on ne va pas pouvoir représenter telles quelles les quelques millions d'années nécessaires à la formation de celui-ci !

De part leur aspect distribué, les systèmes multi-agents posent un certain nombre de problèmes concernant le temps. Comme les agents ont une capacité d'évolution autonome et indépendante, ils ont une représentation locale et propre du temps. De plus, il convient de synchroniser ceux-ci afin de garder une cohérence temporelle au niveau de la simulation dans sa globalité. Il est, en effet, obligatoire qu'une tâche qui doit s'accomplir à un temps $t+1$ ne soit pas réalisées au temps t par exemple. Afin de pouvoir prendre en compte cette notion de temps de la manière la plus représentative qu'il soit, deux principales approches sont à distinguer : l'approche avec un temps centralisé et l'approche avec un temps dirigé par les événements.

2.7.1 Le temps centralisé

Dans le cas d'un temps centralisé, la durée de la simulation est découpée en un certain nombre de pas de temps. L'intervalle entre ces deux pas de temps est identique. Il n'existe qu'une seule horloge au sein du système multi-agents et c'est elle qui assure le tempo de la simulation. A chaque nouveau pas de temps, l'horloge envoie une impulsion à tous les agents et c'est le début pour eux d'une nouvelle séquence d'exécution de tâches. Chaque agent doit donc posséder une liste de tâches à accomplir (cette liste peut éventuellement être vide ou ne contenir qu'une seule tâche). Quand les agents ont terminé leurs tâches respectives, ils se mettent en sommeil en attendant de recevoir une nouvelle impulsion venant de l'horloge principale.

Cette approche a été, et est toujours, utilisée dans de nombreux simulateurs multi-agents, citons par exemple CORMAS [BBaPP98] ou bien encore MANTA [DCF93]. C'est, en effet, la manière la plus simple et facile d'intégrer le temps dans les systèmes multi-agents, d'où sont succès. Néanmoins, cette approche pose un problème crucial concernant le plus petit pas de temps minimum alloué à toutes les tâches des agents [PG97]. Il est, en effet, nécessaire de trouver cet intervalle de temps minimum pour que toutes les tâches à

[PG97] Christophe Le PAGE et Vincent GINOT. – Vers un simulateur générique de peuplements piscicoles. *In : Proceedings Des 5èmes Journées Francophones d'Intelligence Artificielle et*

réaliser par un agent soient prises en compte sans courir le risque d'en oublier une. Cette recherche n'est pas du tout aisée et facile dans le cas des systèmes complexes avec un grand nombre de tâches à réaliser pour chaque agent.

Enfin, la figure 2.9 illustre ce mécanisme de temps centralisé.

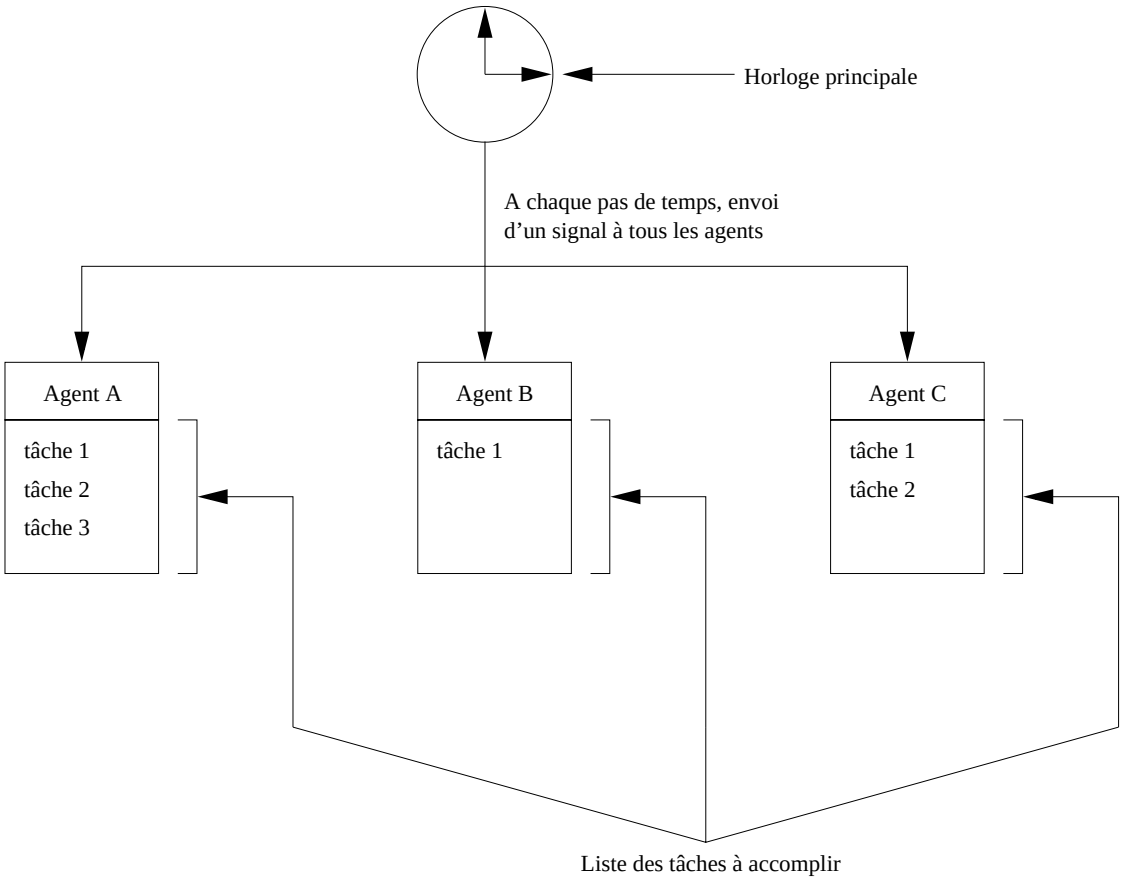


FIG. 2.9 – *Représentation d'un temps centralisé*

De plus, cette représentation apparaît comme assez peu souple. Il est, en effet, impossible de prendre en compte des évolutions d'intervalle de temps au cours de la simulation. Une fois l'intervalle de temps fixé, il ne peut plus être modifié.

2.7.2 Le temps dirigé par les évènements

Dans le cas d'une approche dirigée par les événements, le progression du temps n'est pas linéaire et elle dépend donc des événements. Ceux-ci sont générés à des dates précises.

Systèmes Multi-Agents – JFIADSMA'97, éd. par J. QUINQUETON, M.C THOMAS et B. TROUSSE. – Hermès, La Colle sur Loup, France, avril 1997.

Les agents sont sensibles à ces évènements, et lorsqu'ils en captent un qui les concerne, ils déclenchent les tâches appropriées au type d'évènement reçu. Quand il n'y a plus d'évènements à générer, les agents se mettent en sommeil et restent à l'écoute d'éventuels nouveaux évènements. Le système qui contrôle la simulation passe alors automatiquement au temps le plus proche qui nécessite de générer de nouveaux évènements.

Cette approche semble assez similaire que celle décrite en section 2.7.1 page 28 car elle fonctionne par envois d'évènements alors que la première fonctionne par envois de signaux. Mais, néanmoins, le fait qu'elle soit capable de sauter des intervalles de temps permet justement de « gagner du temps » par rapport à l'approche en temps centralisé. De plus, elle évite de générer des impulsions qui ne servent à rien s'il n'y a rien à faire. Enfin, elle permet une plus grande souplesse en ce qui concerne l'évolution des intervalles de temps. Il est, en effet, possible de rajouter dynamiquement des évènements qui seront pris en compte automatiquement en compte par les agents concernés par ce type d'évènement. Par contre, elle reste aussi pauvre que la précédente en terme de prise en compte de la représentation locale du temps.

2.8 Conclusion

Les systèmes multi-agents étant de plus en plus utilisés pour modéliser et simuler des phénomènes naturels complexes, la prise en compte de l'environnement prend de nos jour une place de plus en plus prépondérante au sein du travail de modélisation multi-agents. Or, à lecture de cet état de l'art et en s'appuyant sur une remarque de J. Ferber dans [Fer95] (pp. 226–227) :

> *[...] « il faut d'abord passer par un problème clé, l'environnement. Celui-ci constitue, en effet, une partie essentielle des systèmes multi-agents situés. Malheureusement, très peu de travaux ont été consacrés à leur modélisation, et les exposés portant sur l'environnement sont en général perdus dans les explications des systèmes les ayant implémentés, voire totalement noyés dans le code de leur implémentation » [...]*

On s'aperçoit que la notion d'environnement reste à ce jour assez pauvre au sein des systèmes multi-agents. Que ce soit dans les divers modèles utilisés à ce jour (centralisé, distribué, ...), ou dans les différentes plate-formes existantes, la modélisation de l'environnement reste en quelque sorte le « parent pauvre » de la simulation multi-agents. Or celui-ci s'avère être le médium crucial pour les systèmes multi-agents. C'est pourquoi les travaux présentés dans ce document visent à fournir un cadre de travail précis permettent de modéliser l'environnement et par la même occasion, un modèle visant à représenter de manière générique les échanges inévitables existant entre l'agent et son environnement.

Chapitre 3

D'un modèle mono-environnemental ...

Sommaire

3.1 Terminologie

Dans cette section nous allons nous attacher à définir une terminologie précise qui sera utilisée ultérieurement dans nos travaux.

Pour ce faire, reprenons les définitions énoncées par J. Ferber dans [Fer95].

Définition 1 *On appelle agent une entité physique ou virtuelle*

- *qui est capable d'agir dans un environnement ;*
- *qui peut communiquer directement avec d'autres agents ;*
- *qui est mue par un ensemble de tendances ;*
- *qui possède des ressources propres ;*

[Fer95] Jacques Ferber. – *Les Systèmes Multi-Agents – Vers une intelligence collective.* – iia – InterEditions, 1995.

- *qui est capable de percevoir (mais de manière limitée) son environnement ;*
- *qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune) ;*
- *qui possède des compétences et offre des services ;*
- *qui peut éventuellement se reproduire ;*
- *dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.*

De même, J. Ferber définit un système multi-agents comme suit :

Définition 2 *On appelle système multi-agents, un système composé des éléments suivants :*

- *Un environnement E, c'est-à-dire un espace disposant généralement d'une métrique ;*
- *Un ensemble d'objets O. Ces objets sont situés, c'est-à-dire que, pour tout objet, il est possible, à un moment donné, d'associer une position dans E. Ces objets sont passifs, c'est-à-dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents ;*
- *Un ensemble A d'agents, qui sont des objets particuliers ($A \subseteq O$), lesquels représentent les entités actives du système ;*
- *Un ensemble de relations R qui unissent des objets (et donc des agents) entre eux ;*
- *Un ensemble d'opérations Op permettant aux agents de A de percevoir, produire, consommer, transformer et manipuler des objets de O ;*
- *Des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on appellera loi de l'univers.*

De cette définition générale d'un système multi-agents, on peut mettre en exergue deux types bien distincts de systèmes multi-agents :

1. Lorsque $E = O$, on a $E = \emptyset$. Dans ce cas, les relations R définissent un réseau, c'est-à-dire que chaque agent est lié directement à un ensemble d'autres agents. C'est ce que l'on appelle un *système multi-agents purement communiquant* et le contenu de E peut être vu comme la structure du réseau de communication sous-jacent. C'est ce que l'on appelle le *réseau d'accointances* de l'agent.
2. Lorsque E est muni d'une métrique ainsi que d'un espace métrique, on parlera d'un *système multi-agents situé*. Par exemple, il suffit de munir E d'une distance Euclidienne pour que E devienne un espace géométrique Euclidien.

3.1.1 Terminologie liée à l'agent

Comme on a pu le lire dans la définition 1, on peut voir l'agent comme un assemblage de diverses composantes. On peut ainsi isoler 4 composantes principales :

- Une composante de *perception* et *d'action* ;
- Une composante de *communication* ;
- Une composante d'*autonomie* ;
- Une composante de *représentation environnementale*.

La composante de perception et d'action aura en charge de collecter des informations provenant de l'environnement et de ses communications avec les autres agents. La composante de communication aura en charge de communiquer avec les autres agents que ce soit des réceptions ou des envois de message. La composante d'autonomie aura en charge de gérer ses *tendances* propres (que ce soit une fonction de satisfaction ou de survie que l'agent cherche à optimiser ou des buts individuels à satisfaire). Enfin, la composante de représentation environnementale sera « plongée » dans l'environnement physique et aura en charge de pouvoir fournir une représentation de celui-ci. On peut donc voir un agent comme décrit dans la figure 3.1.

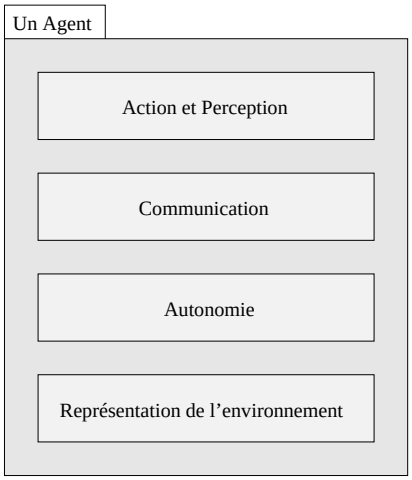


FIG. 3.1 – *Les 4 composantes d'un agent*

Néanmoins, cette construction peut être encore plus généralisée. Il est possible, en effet, de fusionner les composantes de communication et d'action/perception : on peut, effectivement, voir la communication entre deux agents comme une action quand on envoie le message et comme une perception quand on reçoit le message. Dans le cas d'un système multi-agents purement communiquant, l'action et la perception ne seront que des actes de communication et dans le cas d'un système multi-agents situé, il pourra y avoir de l'action, de la perception et de la communication. C'est pourquoi on peut donc construire

un agent avec seulement 3 composantes principales, sans avoir de perte au niveau de ses capacités, comme décrit dans le figure 3.2.

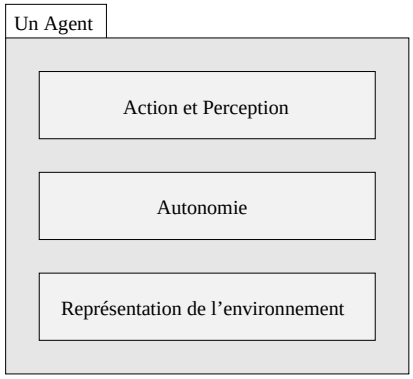


FIG. 3.2 – *Les 3 principales composantes d'un agent*

Maintenant que nous avons isolé les 3 composantes principales d'un agent, mous pouvons élaborer une architecture et une terminologie liées à celui-ci. Pour ce faire, tout d'abord, nous pouvons remarquer que la composante « action et perception » est très intimement liée à l'environnement, alors que les composantes « autonomie » et « représentation de l'environnement » sont beaucoup plus indépendantes de son environnement (même si elles doivent traîter des informations provenant de celui-ci). C'est pourquoi, nous avons été conduits a représenter l'agent comme une composition de deux parties distinctes : *la partie conative* qui va comprendre la composante d'autonomie et la composante représentation de l'environnement ; et la partie *instance de l'agent dans l'environnement* qui va comprendre la composante d'action et perception. La figure 3.3 illustre cette décomposition :

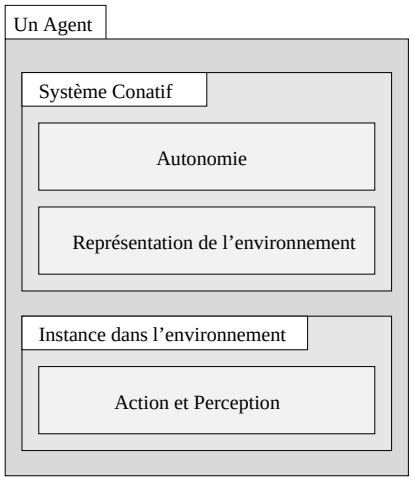


FIG. 3.3 – *Un agent avec ses deux parties*

L'indéniable intérêt de cette décomposition réside dans le fait que l'on peut isoler, lors de l'étape de modélisation, de manière claire ce qui est à la charge du système conatif et ce

qui est à la charge de l'instance de l'agent dans l'environnement. Néanmoins, la figure 3.3 page ci-contre est encore incomplète, car si on isole le système conatif et son instance dans l'environnement en deux entités, elles doivent pouvoir continuer à communiquer entre elles. C'est pourquoi, nous ajoutons un lien de communication bidirectionnel appelé *lien de dépendance bidirectionnel*, car le système conatif de l'agent dépend, entre autre, de son instance dans l'environnement ; et vice-versa, l'instance dans l'environnement a besoin d'un système conatif. La figure 3.4 illustre cette décomposition finale.

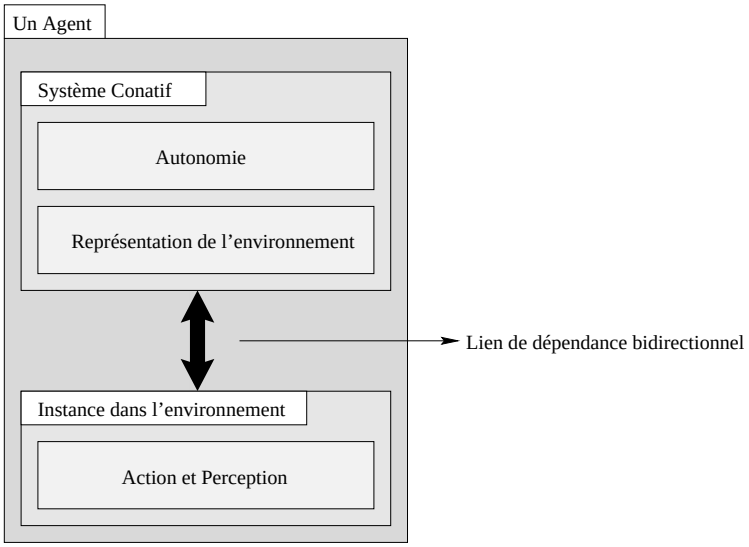


FIG. 3.4 – *Un agent avec ses deux parties et son lien de dépendance bidirectionnel*

L'instance dans l'environnement

Cette instance, qui est plongée dans l'environnement, va être équipée de dispositifs qui vont lui permettre de percevoir son environnement. Pour ce faire, nous utiliserons la terminologie de *capteurs*. Tout d'abord, regardons la définition d'un capteur telle que proposée dans [Dic] :

> *Organe capable de détecter un phénomène (bruit, lumière, etc.) à sa source et d'envoyer l'information vers un système plus complexe (par exemple, un calculateur en temps réel)*

On peut s'apercevoir que la définition d'un capteur correspond tout à fait à celle que nous voulions. C'est, en effet, un dispositif qui va capter un phénomène bien précis et va ensuite le transmettre à un autre dispositif. De plus, comme un capteur est dédié à un phénomène spécifique, nous allons avoir autant de capteurs que de phénomènes à étudier.

[Dic] Dictionnaire francophone universel en ligne. – http://www.francophonie.hachette-livre.fr/.

Par exemple, nous pourrons avoir un capteur de chaleur, de luminosité, de vorticité [SM99] ou bien encore de vision. Nous verrons en détail le fonctionnement de cet équipement plus en avant dans ce document.

De manière analogue, l'instance dans l'environnement doit être capable de pouvoir agir sur celui-ci. Pour ce faire, nous utiliserons la terminologie d'*effecteur*. Tout d'abord, reportons nous à la définition fournie par [Dic] :

> *Organe qui agit sous l'influence d'une commande nerveuse ou hormonale, en réponse aux stimulations reçues par les organes récepteurs.*

On peut s'apercevoir que la définition d'un effecteur s'adapte tout à fait à nos besoins car nous avons effectivement besoin d'un dispositif qui agisse en fonction de réactions perçues précédemment. C'est tout à fait le sens de cette définition. De le même manière que pour les capteurs, nous aurons des effecteurs qui seront dédiés à des actions bien précises et c'est la somme de ces actions qui composera le comportement global de l'agent. Par exemple, nous pourrons voir des effecteurs de rotation, de déplacement [SM99], d'accélération ou bien encore d'action sur la chaleur. De même, nous verrons plus en détail le fonctionnement de cet équipement plus en avant dans ce document.

Le système conatif

Comme décrit précédemment, le système conatif est en quelque sorte « l'âme » de l'agent. C'est lui qui va prendre des décisions et qui va posséder une capacité plus ou moins grande de réflexion. Lorsque le système conatif est doué d'une grande capacité de réflexion (proche, ou éventuellement égale, à un système expert), on parlera, dans ce cas, d'un agent cognitif [WJ95]. A contrario, lorsque le système conatif se contente de juste réagir aux stimuli, on parlera d'un agent réactif [Fer94]. Il est bien évident qu'il n'existe pas de frontière strictement établie entre ces deux notions, c'est pourquoi ici, nous ne nous attacherons pas à une forme d'agent plutôt qu'à une autre. Pour nous le modèle d'environnement et d'interactions entre les agents et l'environnement doit être suffisamment générique pour pouvoir s'appliquer indifféremment à un agent cognitif ou à un agent réactif.

[SM99] Jean-Christophe Soulié et Pierre Marcenac. – Environmental simulations using multiagent systems. *In : Proceedings of the IEEE International Conference on Information, Intelligence and Systems – ICIIS'99.* – IEEE Computer Society Press, Washington, DC, USA, novembre 1999.

[WJ95] M. Wooldridge et N.R Jennings. – *Intelligent Agents: Theory and Practice*, chap. unkown. – Knowledge Engineering Review, 1995.

[Fer94] Jacques Ferber. – Reactive distributed artificial intelligence: Principles and applications. *In : Sixth Generation Computer Technology*, éd. par G.M.P O'Hare et N.R. Jennings. pp. 287–314. – Waley-Interscience Publication, New York, USA, 1994.

Dans la section précédente, nous avons décrit l'instance de l'agent dans l'environnement et nous avons défini deux entités qui permettent de percevoir et d'agir sur l'environnement. Afin que le système conatif puisse lui aussi prendre en compte ces perceptions et prendre des décisions, le système conatif va donc être équipé de deux registres :

- Un registre qui va contenir les perceptions provenant de l'environnement. Ces perceptions vont lui être fournies par le lien de dépendance bidirectionnel qui existe entre le système conatif et son instance dans l'environnement ;

- Un registre qui va contenir les commandes que désire effectuer l'agent dans l'environnement. Ces commandes vont être envoyées à la représentation physique du système conatif par le biais du lien de dépendance bidirectionnel.

Enfin, le système conatif est composé de deux entités : une entité d'autonomie et une entité de représentation de l'environnement. C'est dans cette première entité que l'on va pouvoir modéliser le comportement de l'agent et donc, par la même, intégrer plus ou moins « d'intelligence » dans ses prises de décisions. Dans la deuxième entité, nous allons trouver la manière dont l'agent se représente l'environnement dans lequel il évolue. Cette vision est, évidemment, fonction de ce qu'il a pu percevoir durant les étapes précédentes de simulation et également fonction des changements liés aux actions qui ont été réalisées sur l'environnement. C'est pourquoi, cette représentation n'est pas forcément le reflet exact de l'environnement tel qu'il est réellement. De plus, cette représentation est amenée à varier au cour de la simulation. Par exemple, dans le cas d'une simulation d'un système multi-agents purement communiquant, un agent aura connaissance des agents avec lesquels il communique, mais il n'aura pas forcément connaissance de liens d'accointances qui lient les autres agents entre eux ; par contre, il pourra éventuellement le déduire. Un autre exemple nous est fourni par A. Drogoul dans son projet MICRobES [DP99]. Dans ce projet lié à la robotique, les robots (*e.g* les agents) possèdent une cartographie des lieux (*e.g* le laboratoire LIP6 de Jussieu), mais au début de la vie des robots, cette carte est vide et donc elle va être mise à jour au fur et à mesure de l'évolution du robot dans le laboratoire.

La figure 3.5 page suivante illustre le système conatif de l'agent.

Dans la prochaine section nous allons nous attacher à définir la terminologie liée à l'environnement.

[DP99] Alexis Drogoul et Sébastien Picault. – Microbes : vers de collectivités de robots socialement situés. *In : Actes des 7ème Jounées Francophones en Intelligence Artificielle Distribuée et Systèmes Multi-Agents*, éd. par Marie-Pierre Gleizes et Pierre Marcenac. pp. 265–277. – Hermes, Saint Gilles, Ile de La Réunion, novembre 1999.

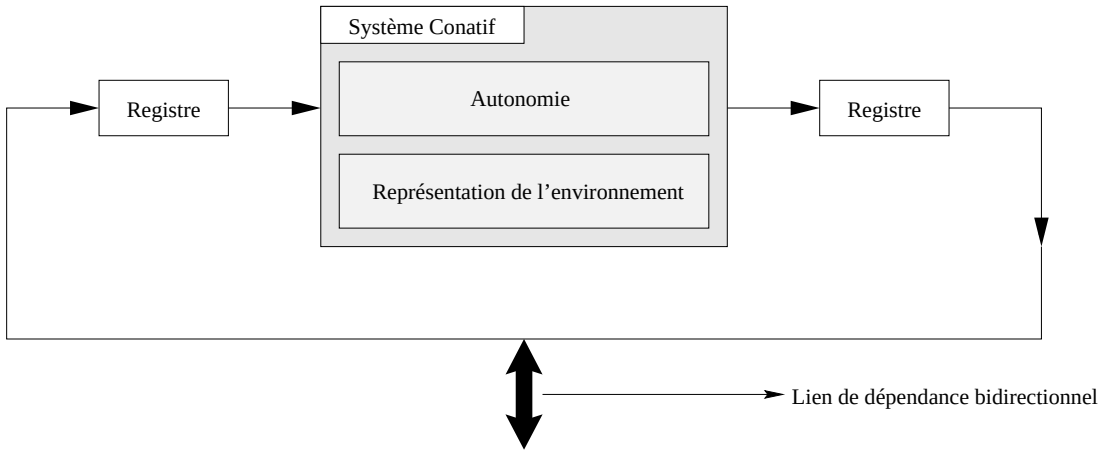


FIG. 3.5 – *Le système conatif de l'agent*

3.1.2 Terminologie liée à l'environnement

Conformément à la définition donnée par J. Ferber dans [Fer95] et détaillée dans la section 1, l'environnement est un espace quelconque. Cet espace pourra être une représentation fidèle de l'environnement de simulation, une abstraction de l'environnement de simulation ou bien encore juste un réseau d'accointances (ou bien encore un graphe de relation) dans le cas de système multi-agents purement communiquant. Comme nous l'avons présenté en section 3.1.1, notre modèle et notre décomposition supporte tous ces types de simulation. Donc pour une simulation donnée, l'environnement contiendra au minimum toutes les représentations physiques des systèmes conatifs. Ces représentations physiques sont contenues dans un vecteur appelé *vecteur représentations physiques* et noté $\mathcal{VRP}$. Ce vecteur est initialisé au début de la simulation et il peut, éventuellement, être mis à jour au cours de la simulation si de nouveaux agents apparaissent. De plus, l'environnement va contenir tous les objets (éventuellement aucun) qui seront passifs, mais que les agents doivent pouvoir prendre en compte (par exemple un mur, une porte, une rivière ou bien encore une route). Ces objets sont appelés des *objets situés* et ils sont notés $\mathcal{OS}$. Le vecteur qui va les contenir s'appelle *vecteur objets situés* et il est noté $\mathcal{VOS}$. Enfin, l'environnement peut, éventuellement, être soumis à des phénomènes qui lui sont propres et intimement liés. Ces phénomènes peuvent prendre diverses formes : par exemple le vecteur d'accélération de la gravité $\vec{g}$ que l'on pourra définir selon la volonté de la personne qui va concevoir le système, l'altération que peuvent subir des objets situés (l'érosion liée à l'écoulement d'une rivière par exemple) ou bien encore des constantes d'accroissement pour des objets situés (par exemple pour des arbre, des plantes, ...). Ces phénomènes appelés la *phénoménologie* de l'environnement sont stockés dans un vecteur appelé *vecteur phénomènes* et il est noté $\mathcal{VP}$. Bien entendu, de manière analogue à l'autre vecteur $\mathcal{VOS}$, ce vecteur peut être vide et donc on pourra effectuer des simulations sans phénoménologie associée.

De plus, on peut trouver dans la littérature et selon les différentes simulations effec-

tuées, un grand nombre de possibilité dans la manière de concevoir et de définir l'environnement.

La première manière, et la plus utilisée, et de voir l'environnement comme une surface divisée en cellules. Cette vision vient essentiellement du fait que beaucoup de chercheurs utilisent des données environnementales provenant du monde des systèmes d'information géographiques (SIG)[8] et que c'est de cette manière que les SIGs modélisent les données. En fait, on peut, dans ce cas, modéliser l'environnement de la manière suivante : l'environnement est composé de n ($n \geq 1$) cellules et ces cellules sont d'une taille définie (dans le cas d'un environnement en 2 dimensions la cellule sera définie par sa *hauteur* et sa *largeur* et dans se cas, la taille t de l'environnement sera la suivante : $t = (n \times hauteur) \times (n \times largeur)$. Dans le cas d'un environnement en 3 dimensions la cellule sera définie par sa *profondeur* et la taille t de l'environnement sera la suivante : $t = (n \times hauteur) \times (n \times largeur) \times (n \times profondeur)$. Cette méthode est notamment utilisée par CORMAS de F. Bousquet [BBaPP98] et par MOBIDYC de V. Ginot et C. Le-Page [GL98]. Dans ce cas, le déplacement se fera de deux manières : d'une façon continue à l'intérieur de la cellule et de façon discontinue d'une cellule à une autre.

La deuxième méthode pour modéliser l'environnement revient à définir celui-ci par sa hauteur h et sa largeur l et dans ce cas, la taille t de l'environnement est calculée de la manière suivante : $t = l \times h$. Cette manière de modéliser l'environnement n'est ni plus ni moins qu'une généralisation de la première, car il suffit de prendre une dimension de 1 pour la cellule et on obtient un environnement continu. Dans ce dernier cas, les déplacements se feront toujours de manière continue. Une des plus fameuses utilisation des environnements continus est décrites dans les travaux de E. Bonabeau, M. Dorigo et G. Theraulaz sur les « algorithmes fourmis » [BDT99].

La troisième manière est d'utiliser un environnement infini. Dans ce cas on peut isoler deux configurations bien précises :

L'environnement purement infini Dans ce cas, il n'y a aucune borne pour l'environ-

8. Le lecteur désirant opérer une lecture plus approfondie sur le sujet de SIGs pourra consulter les ouvrages de J. Denèche et F. Salgé [DS96] et de C. Jones [Jon97]

[BBaPP98] François BOUSQUET, Innocent BAKAM, Hubert PROTON et Christophe Le PAGE. – Cormas: Common-pool ressources and multi-agents system. *In : Proceedings of the* 11^{th} *International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, éd. par A.P del POBIL et M. ALI, *Lecture Notes in Artificial Intelligence*, volume 1416. pp. 826–838. – Springer Verlag, juin 1998.

[GL98] Vincent GINOT et Christophe LE PAGE. – Mobidyc, a generic multi-agents simulator for modeling populations dynamics. *In : Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, éd. par A.P del POBIL et M. ALI, *Lecture Notes in Artificial Intelligence*, volume 1416. pp. 805–814. – Springer Verlag, juin 1998.

[BDT99] Eric BONABEAU, Marco DORIGO et Guy THERAULAZ. – *From Natural to Artificial Swarm Intelligence.* – Oxford University Press, 1999.

nement et donc l'agent pourra se déplacer de manière infinie au sein de celui-ci. Il conviendra toutefois de mettre en place un système de visualisation suffisamment puissant qui permette de pouvoir suivre l'évolution des agents au sein de l'environnement.

L'environnement borné toroïdal Dans ce cas, l'environnement est effectivement borné, mais lorsque l'agent doit effectuer un déplacement qui doit le faire sortir de l'environnement, celui-ci se retrouve de l'autre côté de l'environnement. Par exemple, dans le cas d'un environnement en 2 dimensions (où x représente une valeur de l'abcisse et y représente une valeur de l'ordonnée), si un agent doit effectuer un déplacement $x + 1$ (resp. $y + 1$) et qu'on a déjà $x = largeur$ (resp. $y = hauteur$), alors on aura après le déplacement : $x = 0$ (resp. $y = 0$). De la même manière, si un agent doit effectuer un déplacement $x - 1$ (resp. $y - 1$) et qu'on a déjà $x = 0$ (resp. $y = 0$), alors on aura après le déplacement : $x = largeur$ (resp. $y = hauteur$). Cette manière de modéliser l'environnement est très utilisée en robotique par exemple et en vie artificielle. Par exemple, S. Calderoni avec sa plate-forme MUTANT utilise ces deux types d'environnements [CM98, CM99].

Pour en finir avec ces manières de représenter l'environnement, la dernière façon de le représenter est lorsque l'on n'utilise pas d'environnement réel, mais que le système multi-agents est purement communiquant. Dans ce cas, le réseau d'accointances sera un graphe et donc l'environnement sera défini par ce graphe. Une mesure classique liée à ce type d'environnement est le nombre d'arcs qu'il existe entre les agents (*e.g* les communications entre les agents au travers du réseau d'accointances), qui permet de mesurer le degré de communication du système. Cette modélisation est très utilisée dans les simulations qui n'utilisent pas d'environnement « réel » en tant que représentation ou abstraction d'un environnement physique. La plate-forme GEAMAS de P. Marcenac est un très bon exemple d'environnement pour des systèmes multi-agents purement communiquant [MG98, MC99, SMCC98].

[CM98] Stéphane CALDERONI et Pierre MARCENAC. – MUTANT: A multiagent toolkit for artificial life simulation. *In : Proceedings of the* 26^{th} *International Conference on Technologies of Object-Oriented Languages and Systems*, éd. par M. SINGH, B. MEYER, J. GIL et R. MITCHELL. pp. 218–229. – IEEE Computer Society Press, Santa Barbara, California, USA, août 1998.

[CM99] Stéphane CALDERONI et Pierre MARCENAC. – Mutant: a Genetic Learning System. *In : Proceedings of the* 12^{th} *Australian Joint Conference on Artificial Intelligence (AI-99)*, éd. par Norman FOO, *Lecture Notes in Artificial Intelligence*, volume 1747. – Springer, Berlin, décembre 1999.

[MG98] Pierre MARCENAC et Sylvain GIROUX. – *GEAMAS: A Generic Architecture for Agent-Oriented Simulations of Complex Processes.* – International Journal of Applied Intelligence, Kluwer Academic Publishers, 1998.

[MC99] Pierre MARCENAC et Rémy COURDIER. – *Java Agent-Oriented Development Environment.* – John Wiley & Sons Books, 1999, *Journal of Object-Oriented Application Frameworks, M. Fayad.*

[SMCC98] Jean-Christophe SOULIÉ, Pierre MARCENAC, Stéphane CALDERONI et Rémy COURDIER. – GEAMAS v2.0: An object oriented platform for complex systems simulations. *In : Proceedings of the* 26^{th} *International Conference on Technology of Object-Oriented Languages and Systems – TOOLS USA'98*, éd. par M. SINGH, B. MEYER, J. GIL et R. MITCHELL. pp.

Après avoir défini tous les termes que nous allons utiliser dans ce document, nous allons maintenant nous attacher à montrer comment toutes ces entités interagissent et comment le modèle est construit.

3.2 Comment toutes ces entités interagissent ?

En accord avec notre terminologie présentée dans la section précédente, nous allons concentrer notre étude sur deux principales entités : l'environnement car c'est cette entité qui sert de médium de communication et enfin l'agent car son comportement est contraint par l'environnement ; et en retour de part ses actions sur l'environnement, l'agent agit sur celui-ci.

3.2.1 Au niveau de l'agent

Dans un premier temps, nous allons montrer comment l'agent prend en compte ses perceptions, comment il agit sur l'environnement et enfin, comment il interagit avec son instance dans l'environnement.

Dans le système conatif

Le système conatif contient deux entités bien distinctes : une entité d'autonomie et une entité de représentation de l'environnement. La première doit être capable de prendre en compte des perceptions provenant de l'environnement, de réagir à ces perceptions, de prendre des décisions en fonction de ses tendances propres et, éventuellement de donner naissance à un clone ou à un agent d'un autre type. La deuxième entité est tenue à jour par l'entité d'autonomie tout au long de la simulation.

On peut séparer deux parties bien distinctes dans ce qui se passe au niveau du système conatif : la partie perception et la partie action. Dans la section suivante nous allons nous attacher à présenter ce qu'il se passe au niveau des perceptions et dans la suivante nous allons présenter ce qu'il se passe au niveau des actions.

La perception

Dans la section 3.1.1 page 33, nous avons présenté le système conatif comme étant équipé d'un registre permettant de stocker les perceptions provenant de l'environnement.

Ce registre est appelé *registre de perceptions* et il est noté $\mathcal{RP}$. Nous avons donc à un instant t de la simulation :

$$\mathcal{RP} = \{p_1, \ldots, p_n\}$$

où n est le nombre de capteurs qui ont effectivement perçu quelque chose[9]. Ce registre, quand il contient des éléments est ensuite pris en charge par le *gestionnaire de priorités*. Ce gestionnaire de priorités, qui est en liaison constante avec l'entité d'autonomie, est capable à tout moment de donner une priorité à une perception p_i dans $\mathcal{RP}$. Plus concrètement, lorsque que $\mathcal{RP}$ contient des perceptions p_i, le gestionnaire de priorités est capable de déterminer de quel type t_i est cette perception et ensuite il demande à l'entité d'autonomie quelle est la priorité pr_i associée à t_i et donc à p_i. Ensuite, lorsque le gestionnaire de priorités a construit une table avec toutes les perceptions et les priorités associées comme présenté dans la table 3.1, il les trie dans un nouveau registre appelé *registre de perceptions*

Perceptions	Types	Priorités
p_1	$t_{a'}$	pr_a
p_2	$t_{b'}$	pr_b
$\vdots$	$\vdots$	$\vdots$
p_n	$t_{m'}$	pr_m

TAB. 3.1 – *Une table de perceptions générée par le gestionnaire de priorités*

ordonnées est noté $\mathcal{RPO}$. Et nous avons donc :

$$\mathcal{RPO} = \{p_m, \ldots, p_1\}; \forall i,j \in [1, \ldots, m] \text{ avec } i < j \text{ alors } p_i \geq p_j$$

où m est la priorité maximum qui aura été définie par le concepteur lors de la phase de mise en place de l'agent. Finalement, ce registre est ensuite directement couplé à l'entité d'autonomie qui va pouvoir lire une à une toutes les perceptions d'un instant t et donc pouvoir les prendre en compte automatiquement au niveau de sa capacité de raisonnement. Comme nous codons la priorité sur un entier, on a pour donc, toutes les priorités p_i, $p_i \in \mathbb{N}, \forall i \in [1, \ldots, m]$, où m est la priorité maximale fixée par le concepteur de l'agent.

Le fait qu'une priorité soit donnée à une perception nous permet de représenter l'importance d'une perception par rapport à une autre. Pour un agent, en effet, une perception est quelque chose qu'il ressent et ce ressentiment peut être complètement différent selon son type et donc posséder un sens plus ou moins important selon ce type. Selon la manière dont il aura été modélisé, une peine pourra voir un sens beaucoup plus important qu'une joie et donc la perception de la peine aura une priorité beaucoup plus forte que la perception de la joie. Par exemple, dans le cadre de simulation de comportement de bancs d'espadons dans la zone de l'océan Indien [SM99, GSML99] si un espadon perçoit

9. si a est le nombre de capteurs pour un agent donné, on a alors $n \leq a$

[GSML99] David GUYOMARD, Jean-Christophe SOULIÉ, Pierre MARCENAC et Michel LARUE. – Mise

un fort gradient de température et en même temps un prédateur (un requin par exemple), la plus forte priorité sera donnée à cette deuxième perception car c'est la plus vitale pour lui. Il doit, en effet, réagir en premier lieu à cette menace.

En plus de ce premier dispositif destiné à gérer les priorités des perceptions, nous ajoutons un autre registre à l'entité d'autonomie. Ce registre, appelé *registre de perceptions urgentes* est noté $\mathcal{RPU}$. Ce registre est lui aussi géré par le gestionnaire de priorités. S'il détecte dans le type de la perception un caractère urgent, il va la mettre dans ce registre. Cela va nous permettre de prendre en compte une perception qui aura une priorité sans aucune mesure par rapport aux autres. Quand le gestionnaire de priorité ajoute une ou plusieurs perceptions dans ce registre, l'entité d'autonomie est automatiquement prévenue et elle cesse toutes ses autres activités afin de pouvoir traiter en urgence cette (ou ces) perception(s). Lorsque l'entité d'autonomie aura fini de traiter ces perceptions urgentes, elle va reprendre l'exécution des tâches suspendues et ensuite elle va prendre en compte ce qui va être contenu dans le registre $\mathcal{RPO}$. Bien évidemment, ceci peut induire une sorte de décalage lors des traitements. Par exemple, l'entité d'autonomie, si elle prend du retard lors du traitement des informations contenues dans $\mathcal{RPU}$, les informations contenues dans le registre $\mathcal{RPO}$ deviennent alors obsolètes dans la mesure où de nouvelles perceptions sont parvenues dans le registre $\mathcal{RP}$ et donc l'entité d'autonomie risque d'avoir à traiter des informations qui ne sont plus d'actualité au niveau de l'environnement. Ce problème va être réglé par le gestionnaire de priorité. En effet, lorsqu'il va commencer sa tâche de tri, il va commencer par comparer toutes les perceptions contenues dans $\mathcal{RP}$ et éliminer toutes celles qui sont les plus anciennes et ne garder que celles qui sont actuelles. Néanmoins, on peut se demander qu'elles vont être les conséquences de telles pertes d'informations aux profits de plus récentes. Nous pensons ici que ces pertes d'informations ne sont pas importantes dans la mesure où l'entité qui n'aura pas réagi à une perception à un instant t va se retrouver, à l'instant $t+1$, dans deux cas de figure :

- La perception relative au temps t éliminée à $t+1$ par le gestionnaire de priorités est de nouveau présente dans $\mathcal{RP}$ et donc elle sera traitée à $t+1$ par le gestionnaire de priorités et donc par conséquent par l'entité d'autonomie. Dans ce cas, on peut considérer qu'il n'y a pas eu une perte d'information, mais seulement un décalage dans le temps de cette information.
- La perception relative au temps t éliminée à $t+1$ par le gestionnaire de priorités n'est plus présente dans $\mathcal{RP}$ et donc on a perdu la trace de cette perception. Dans ce cas, on peut considérer que, même si cette perte d'information est préjudiciable, elle est minime car si le phénomène qui a déclenché cette perception à l'instant t n'est plus présent à l'instant $t+1$, son influence lors de la simulation dans son ensemble est minimale. C'est pourquoi cette perte d'information, même si on ne peut en occulter sa présence, est minimisée.

---

en place d'un système multi-agents destiné à la simulation de dynamiques comportementales spatiales environnement/ressource, appliqué à l'espadon (xiphias gladius) dans le sud-ouest de l'océan indien. *In : Actes du 4ème Forum de l'Association Française d'Halieumétrie – Les Espaces de l'Halieutique.* – Edition IRD, Rennes, juin 1999.

Néanmoins, on peut se demander si le retard dû à des perception urgentes à un instant t ne peut pas se répéter à un instant $t+1$ et donc générer un nouveau retard et de nouvelles éliminations de perceptions à $t+1$. Dans ce cas de figure, des perceptions de priorités « normales » ne peuvent être jamais prises en compte. Afin de pouvoir prendre en compte ce problème, deux options sont envisageables :

- Ces perceptions sont effectivement éliminées et le gestionnaire de priorités attend les perceptions suivantes. Mais ce phénomène peut, éventuellement se répéter tout au long de la simulation et dans ce cas la perte d'information peut se révéler très préjudiciable pour l'entité d'autonomie car elle ne va pas pouvoir prendre en compte toutes ces informations.
- On essaye de mettre en place un mécanisme qui force la prise en compte, même tardive, des perceptions qui ont été éliminées ou qui vont l'être.

Dans notre modèle, nous avons choisi de privilégier cette seconde approche plutôt que la première. C'est pourquoi nous ajoutons un registre appelé *registre de perceptions retardées* et noté $\mathcal{RPR}$. Ce registre va contenir les perceptions qui ont été éliminées lors d'étapes précédentes de simulation. On aura donc :

$$\mathcal{RPR} = \{p_1^{t-1}, p_2^{t-1}, \ldots, p_n^{t-1}, \ldots, p_1^{t-l}, p_2^{t-l}, \ldots, p_{n'}^{t-l}\}$$

où p_i^j représente une perception i donnée à instant j, n est le nombre de perceptions à l'instant $t-1$, l est la profondeur du registre et n' est le nombre de perceptions à l'instant $t-l$. A l'étape courante (*e.g* à l'instant t), si le gestionnaire de priorités détecte qu'une perception p_i est susceptible d'être éliminée et qu'elle existe déjà dans $\mathcal{RPR}$ depuis k temps de simulation, alors le gestionnaire de priorités va décider de garder cette perception quand même dans $\mathcal{RPO}$, mais avec une priorité la plus haute possible afin qu'elle soit traitée avant les autres. Et finalement le gestionnaire de priorités enlève toutes les perceptions du type p_i^k (avec $k \in [t-1, \ldots, t-l]$) dans $\mathcal{RPR}$. Donc l'entité d'autonomie aura donc pris en compte cette perception et la simulation pourra se dérouler avec cette perception.

Grâce à ces différents mécanismes, qui ont tous comme point commun d'être reliés de près ou de loin à l'entité d'autonomie, on peut donc simuler l'importance d'une perception par rapport à une autre et aussi prendre en compte le fait qu'il peut y avoir des perceptions qui sont beaucoup plus importantes que les autres et les communiquer en priorité à l'entité d'autonomie. De plus, comme le gestionnaire de priorités est en liaison constante avec l'entité d'autonomie, celle-ci peut à tout moment revoir ou modifier ses règles de priorités et donc, par la même, des perceptions qui auraient été dotées d'une priorité moindre peuvent se retrouver avec une priorité beaucoup plus forte et donc passer devant d'autres. On peut donc dire que grâce à ce mécanisme, l'entité autonome est à même de prendre la décision de s'adapter aux résultats de ses actions et aussi en fonction de ses perceptions.

L'action

La partie action du système conatif se situe à l'opposé de la partie perception. Bien que son rôle soit complètement différent, on peut noter une certaine analogie entre ces deux parties. Mais, néanmoins, nous allons présenter dans cette section tout son fonctionnement.

De manière analogue à la partie liée à la perception, les actions décidées par l'entité d'autonomie sont stockées dans un registre appelé *registre de commandes* et noté $\mathcal{RC}$. On a donc à instant t de la simulation :

$$\mathcal{RC} = \{c_1, \ldots, c_m\}$$

où m représente le nombre de commandes à effectuer sur l'environnement. Lorsque ce registre est rempli de commandes, il est alors pris en charge par le gestionnaire de priorités. Celui-ci, en liaison avec l'entité d'autonomie, va pouvoir donner une priorité à toutes les commandes contenues dans $\mathcal{RC}$ comme illustré ci-dessous :

Commandes	Types	Priorités
c_1	$t_{a'}$	pr_a
c_2	$t_{b'}$	pr_b
$\vdots$	$\vdots$	$\vdots$
c_n	$t_{m'}$	pr_m

TAB. 3.2 – *Une table de commandes générée par le gestionnaire de priorités*

Ensuite, le gestionnaire de priorités dépose ces commandes triées par priorités dans un nouveau registre appelé *registre de commandes ordonnées* et noté $\mathcal{RCO}$. On a donc, à un instant t de la simulation :

$$\mathcal{RCO} = \{c_{m'}, \ldots, c_1\}; \forall i,j \in [1, \ldots, m'] \text{ avec } i < j \text{ alors } c_i \geq c_j$$

où m' est la priorité maximum qui aura été définie par le concepteur lors de la phase de mise en place de l'agent. Comme la priorité est codée sur un entier, on a donc pour toutes les priorités p_i, $p_i \in \mathbb{N}$ $(\forall i \in [1, \ldots, m'])$, où m' est la priorité maximale fixée par le concepteur de l'agent. Mais le gestionnaire de priorités est aussi capable de pouvoir détecter des commandes qui ont un caractère urgent. Dans ce cas, il range ces commandes dans un registre spécial appelé *registre de commandes urgentes* et noté $\mathcal{RCU}$. Toutes les commandes contenues dans $\mathcal{RCU}$ doivent être évidemment envoyées en priorité par rapport à celles contenues dans $\mathcal{RCO}$, c'est pourquoi nous ajoutons une autre entité appelée *ordonnanceur de commandes* et noté $\mathcal{OC}$ à notre modèle qui va avoir en charge l'envoi, dans le bon ordre et selon les bonnes priorités, des commandes présentes dans $\mathcal{RCO}$ et $\mathcal{RCU}$. Concrètement, après être passées au travers de $\mathcal{OC}$, les informations qui vont être contenues dans le message vont transiter le long du lien de dépendance bidirectionnel. Ces commandes vont être formatée comme ci-dessous :

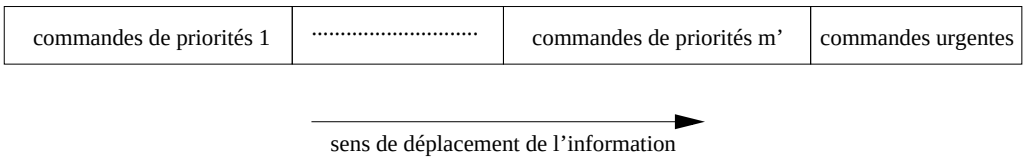


FIG. 3.6 – *Un exemple de message*

Le fait que les commandes qui vont être envoyées à l'environnement soient ordonnées selon des priorités bien précises représente le fait que le système conatif souhaite pouvoir être sûr que des actions vont être appliquées bien avant les autres au niveau de l'environnement et donc qu'il existe une hiérarchie dans les commandes. Par exemple, dans le cas de la simulation de comportement de bancs d'espadons [SM99], le système conatif d'un agent (*e.g* un espadon) peut vouloir qu'une action liée à une mesure de prise de fuite soit prise en compte bien avant les autres car cela représente un caractère vital pour lui de ne pas être capturé par un prédateur. Même si le système conatif souhaite ensuite effectuer d'autres actions sur l'environnement (par exemple, laisser un trace de son passage par un déplacement d'eau, ou changer un paramètre de l'environnement), il faut absolument que les actions urgentes soient prises en compte avant. Nous illustrerons comment ceci est réalisé au niveau de l'environnement dans la section 3.2.2 page 65.

Dans cette partie concernant les mécanismes d'actions, nous avons montré comment le système conatif, par le biais du gestionnaire de priorités, est capable d'influer sur les priorités des actions qui vont être effectuées sur l'environnement. De manière analogue à la partie perception, le système conatif peut à tout moment changer les règles pour le calcul des priorités et donc s'adapter à ce qu'il a pu déduire des précédentes expériences effectuées lors de la simulation.

La représentation de l'environnement

Comme nous l'avons présenté précédemment, le système conatif est capable de percevoir et d'agir sur l'environnement. Mais il construit, en même temps, et au fur et à mesure de la simulation, une représentation de l'environnement qui lui est propre. Dans cette partie, nous allons nous attacher à décrire comment l'entité d'autonomie fait pour construire cette représentation de l'environnement.

Au tout début de la simulation, la représentation locale de l'environnement est, bien entendu, vide dans la mesure où le système conatif n'a rien encore perçu. La première chose que fait l'entité d'autonomie est d'envoyer un message à son instance dans l'environnement, qui elle même va répercuter ce message dans l'environnement, afin d'obtenir une information sur le type de l'environnement : en cellule, continu, ... (cf 3.1.2 page 38) Une fois cette information reçue, l'entité d'autonomie va donc être capable de pouvoir comprendre et gérer les informations qu'elle va recevoir. Puis lors du déroulement de la simulation, le système conatif (et donc l'entité d'autonomie) va recevoir des perceptions

qui vont être liées à des capteurs bien précis et donc il va pouvoir construire au fur et à mesure sa propre représentation de l'environnement. Ces perceptions sont, en effet, une description de l'environnement à un instant t donné car grâce aux capteurs, elles contiennent une information liée à l'environnement donné. Par contre, on s'aperçoit bien que cette description est évidemment locale et non exhaustive car on peut rencontrer deux cas de figures :

- L'agent ne va pas se déplacer, par exemple, dans tout l'environnement. Dans ce cas, toute la zone non explorée reste inconnue pour l'agent et donc il ne va pas pouvoir tenir compte de ce qu'il peut éventuellement se passer dans ces zones. Toutefois, si l'agent « ressent » le désir d'explorer ces zones, il pourra faire le nécessaire au niveau de ses commandes, et donc au niveau de ses effecteurs, pour aller dans ces zones. Dans la cas d'un système multi-agents purement communiquant, si l'agent ne rentre pas directement en communication avec un autre, il ignorera complètement son existence. Par contre, l'entité d'autonomie pourra éventuellement déduire la présence d'un autre agent et essayer de rentrer en contact avec lui.
- L'agent n'est pas équipé des capteurs nécessaires pour lui fournir une description exhaustive de l'environnement et donc certains paramètres de celui-ci vont rester inconnus pour lui. Par exemple, si l'agent n'est pas équipé d'un capteur détectant la présence d'eau, il ne va pas percevoir une rivière devant lui et va éventuellement se retrouver au milieu sans même s'en apercevoir. Néanmoins, on peut se demander si cela présente un caractère de gravité pour lui dans la mesure où il n'a pas été équipé de ce capteur. Donc comme l'agent ne peut strictement percevoir que les phénomènes pour lesquels il possède des capteurs, cette perception de l'environnement est forcément incomplète, sauf dans le cas où les capteurs dont il est muni lui permettent d'accéder à toute l'information disponible dans l'environnement.

De plus, au fur et à mesure de la simulation, l'entité d'autonomie va pouvoir (si le concepteur l'a doté de telles fonctions, dans le cas d'un agent cognitif par exemple) analyser quelle sont les conséquences de ses commandes sur l'environnement quand il reçoit des perceptions qu'il pourra associer à ses commandes.

Pour illustrer cette notion de représentation, reprenons l'exemple de robots utilisé dans le LIP6 dans le cadre du projet MICRobES [DP99]. Ces robots évoluent dans les bureaux du LIP6 et à l'initialisation de leur système conatif, ils ne possèdent aucune représentation (aucune carte) des bureaux. Par contre, ils savent qu'ils vont évoluer dans un environnement de type continu avec des obstacles. A chaque fois qu'ils se déplacent, ils mettent à jour leur carte et donc à chaque fois ils peuvent ajouter des informations du type : au point p_1 de coordonnées (x_1,y_1), je peux me déplacer, au point p_2 de coordonnées (x_2,y_2), je ne peux pas me déplacer et donc je suis en présence d'un mur ou d'une porte, ... Puis, si au hasard de ses déplacements, le robot se retrouve de nouveau au point p_2 et qu'il peut se déplacer, le robot va pouvoir (éventuellement) déduire qu'à ce point il se trouve en présence d'une porte, car selon ses deux expériences dans un cas il ne peux pas passer

et dans l'autre oui. Donc il se trouve en présence d'un mur qui permet, de temps en temps de passer, donc une porte.

La figure 3.7 illustre notre modèle pour le système conatif.

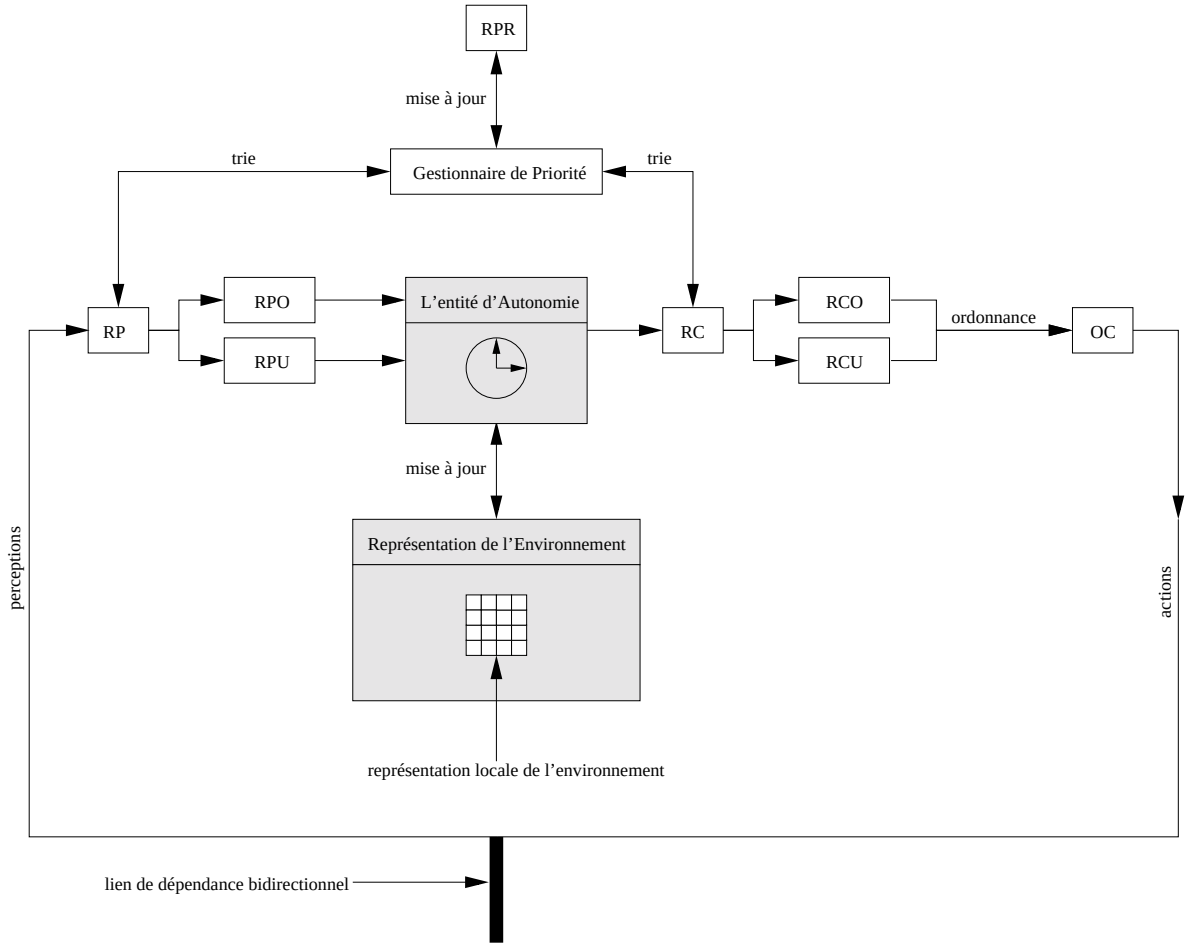


FIG. 3.7 – *Notre modèle pour le système conatif*

Dans la prochaine section nous allons présenter comment l'instance physique du système conatif, qui est plongée dans l'environnement, peut être modélisée.

Dans son instance environnementale

L'instance environnementale du système conatif, de part sa caractéristique principale, est au contact direct de l'environnement, elle aura donc la charge de percevoir les phénomènes de l'environnement afin d'en informer son système conatif et d'agir sur l'environnement par l'intermédiaire des commandes qui lui parviennent du système conatif.

De manière analogue à la partie concernant le système conatif, on peut séparer les tâches à réaliser en deux parties : la perception et l'action.

La perception

Dans le but de percevoir l'environnement, l'instance physique est équipée de capteurs. Ceux-ci sont dédiés à un certain type de phénomène. On aura, par exemple, des capteurs de chaleur, d'obstacles, de lumière ou bien encore un capteur de message dans le cas d'un système multi-agents purement communiquant. Les types d'informations récupérés par ces capteurs peuvent être très différents : numérique, symbolique ou bien encore une énumération par exemple. Quand un capteur a reçu une information (à chaque pas de simulation), il la transmet à son instance environnementale associée. Or, on a pu voir dans la partie concernant le système conatif que celui-ci recevait des messages de haut niveau provenant de son instance environnementale. Malheureusement, les informations perçues par les capteurs sont bien souvent des informations de très bas niveaux et inutilisables en l'état par le système conatif. C'est pourquoi nous avons ajouté une entité appelée *transcodeur de perceptions* qui va avoir la charge de réaliser ce transcodage d'une information brute en une information de plus haut niveau pour le système conatif.

Enfin, ce transcodeur possède une capacité de reconnaissance de perceptions suffisamment développée de sorte qu'il peut décider à tout moment si la perception p_i qu'il est en train de traiter doit être typée normalement ou dans un type « urgent ». Si tel est le cas, cette perception va donc être codée selon le type « urgent » et va être mise en tête des informations qui vont transiter sur le lien de dépendance bidirectionnel. Les autres informations seront normalement codées selon leur type. Néanmoins, il faut noter qu'à une perception p_i récupérée sur un capteur c_i, ne correspond pas forcément une perception $p'_i = p_i$ identique prête à transiter sur le lien de dépendance bidirectionnel. Par exemple, si un agent détecte dans un de ses capteurs qu'il y a un objet en face de lui et que dans un autre capteur, il détecte que sa vitesse est nulle, l'information à faire remonter est qu'il est à l'arrêt dans son déplacement à cause de la présence d'un objet. C'est pourquoi le transcodeur de perceptions contient des règles du type : **si** $perception_1$ **et/ou** $perception_2$ **et/ou** ... **et/ou** $perception_n$ **alors** $perception'_i$. De cette manière on pourra composer un certain nombre de perceptions primitives en d'autres de plus haut niveaux. Enfin, cette composition de perceptions primitives peut être simplement la perception primitive à laquelle on va ajouter des informations descriptives en plus. Par exemple, dans un capteur de vitesse, on va juste obtenir une valeur et le transcodeur de perception va simplement ajouter l'unité de cette vitesse de manière à ce que le système conatif, et plus précisément l'entité d'autonomie, puisse analyser le mieux possible cette perception.

La figure 3.8 page suivante illustre le fonctionnement du transcodeur de perceptions à l'aide de règles.

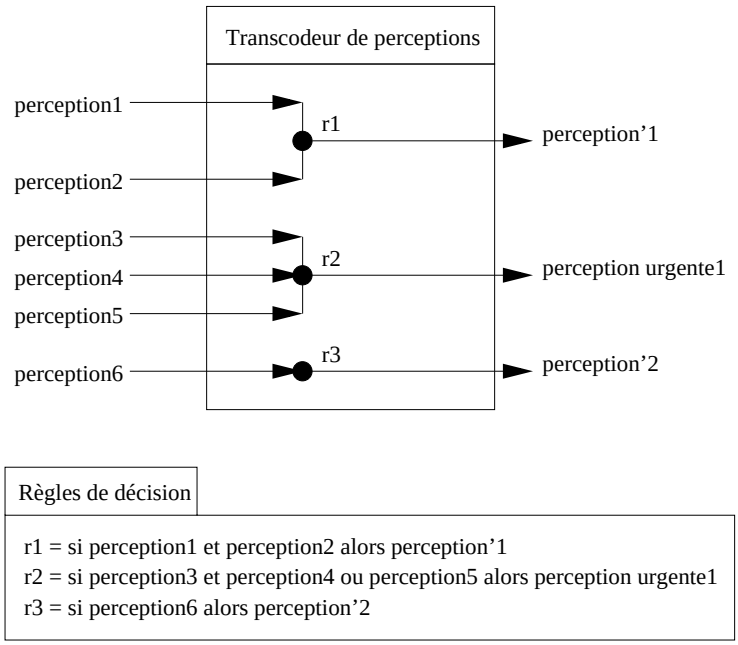


FIG. 3.8 – *Un transcodeur de perceptions*

L'action

De manière symétrique à la perception, l'instance environnementale est équipée d'un ensemble d'effecteurs. Ces effecteurs sont dédiés à un certain type d'action. Par exemple, nous pourrons avoir des effecteurs de mouvement, de rotation ou de message dans le cas d'un système multi-agents purement communiquant. De la même manière, comme le système conatif peut envoyer des messages de type « urgent », nous avons ajouté la possibilité d'équiper l'instance environnementale avec des *effecteurs d'urgences*. C'est dans ceux-ci que l'on va mettre les actions urgentes. De plus, comme les commandes qui arrivent le long du lien de dépendance bidirectionnel sont sous la forme d'un seul et unique message, celles-ci doivent pouvoir être découpées correctement en un ensemble de commandes. C'est pourquoi nous avons ajouté une nouvelle entité dans l'instance environnementale appelée *transcodeur de commandes*. Ce transcodeur aura la charge de séparer le message provenant du système conatif en plusieurs commandes et de les répartir dans les bons effecteurs. A chaque cycle, le transcodeur de commandes va vérifier s'il n'y a pas de commandes urgentes dans le message qu'il a reçu. Si c'est le cas, il va mettre ces commandes dans les effecteurs d'urgences et va immédiatement prévenir l'environnement qu'il doit réaliser des actions urgentes. Puis, il va ensuite traiter les actions qui ont une priorité moins forte et à chaque fois qu'il a terminé de déposer des commandes d'un certain type, il va prévenir l'environnement qu'il y a des actions à traiter. Et ceci jusqu´à ce qu'il n'y ai plus d'actions à traiter. De cette manière, les commandes avec de fortes priorités vont être effectuées avant celles qui ont une priorité moins forte. Néanmoins, on peut se demander ce qu'il se passe au niveau du temps lorsque ces actions sont activées les unes après les autres en fonction de leurs priorités. Il est clair que nous introduisons ici un décalage,

mais celui-ci est voulu du fait de la prise en compte des priorités. Par contre, le cas qui pourrait poser un problème serai si l'environnement ne pouvait pas avoir pris en compte toutes les commandes avant de faire une transition d'un instant t à un instant $t+1$. C'est pourquoi nous avons décidé que ce serait l'instance environnementale, et sur ordre du transcodeur de commandes, qui préviendrait l'environnement que le dépôt de commandes est terminé. Plus précisément, quand le transcodeur de commandes a terminé de déposer toutes les commandes, il va prévenir son instance environnementale. Celle-ci va ensuite prévenir l'environnement qu'il peut faire une transition temporelle. Bien entendu, si l'environnement n'a pas fini le traitement, il va pouvoir continuer ses activités avant de faire une transition. De cette manière, on est assuré que l'environnement ne va pas passer au un pas de simulation suivant sans avoir de perte d'intégrité dans les aspects temporels des actions. Et aussi, on est sûr d'avoir effectivement transmis toutes les actions a effectuer sur l'environnement à celui-ci. Étant bien entendu que certaines actions ne pourront éventuellement pas être prises en compte. En effet, par exemple, dans le cas de la simulation du déplacement des espadons dans l'océan Indien, si un espadon (*e.g* un agent et donc un système conatif) prend la décision (et donc envoie la commande associée) de se déplacer dans une direction précise, il se peut très bien que l'environnement ne puisse pas lui donner satisfaction. Par exemple, si l'espadon se trouve dans une position qui ne lui permet d'effectuer ce déplacement (par exemple, il est près d'un haut fond), l'environnement ne pourra lui donner satisfaction et donc ce sera à la charge du système conatif de s'en apercevoir pour corriger cette commande.

Finalement, on peut noter que, comme dans le cas de la perception, à une commande demandée par le système conatif, ne correspond pas forcément une action réelle sur l'environnement. Il est possible, en effet, qu'une commande soit de trop haut niveau pour qu'elle puisse être effectuée par des actions de bas niveaux. Par exemple, si le système conatif envoie une commande de changement de direction, les actions qui vont être déclenchées sur l'environnement vont être les suivantes :

1. une action de rotation
2. puis, une action d'accélération

C'est pourquoi le transcodeur de commandes va contenir des règles de transformations qui vont lui permettre de réaliser ces décompositions de commandes de haut niveaux en actions de bas niveaux. Ces règles vont être basées sur des branchements conditionnels du type : **si** $commande_i$ **alors** $action_1$,$action_2$,...,$action_n$. De cette manière on pourra décomposer toutes les commandes en actions. De plus, on pourra noter que le séquencement des actions en $action_1$,$action_2$,...,$action_n$ est réalisé de manière séquentielle. De cette manière, on pourra de nouveau introduire une priorité de moindre importance que les autres dans les actions que l'on va déclencher. Celà va être au concepteur de l'agent de décider quelle va être l'action à déclencher en premier, en deuxième, ... Et c'est à chaque fois que le transcodeur de commandes va être sollicité qu'il va réaliser ces opérations. Néanmoins, on pourra remarquer que ces règles sont complètement liées à la simulation qui a été mise en place. C'est pourquoi, d'un point de vue générique, on ne peut que

mettre en place les services liés a ces mécanismes.

La figure 3.9 illustre le fonctionnement du transcodeur de commandes à l'aide de règles.

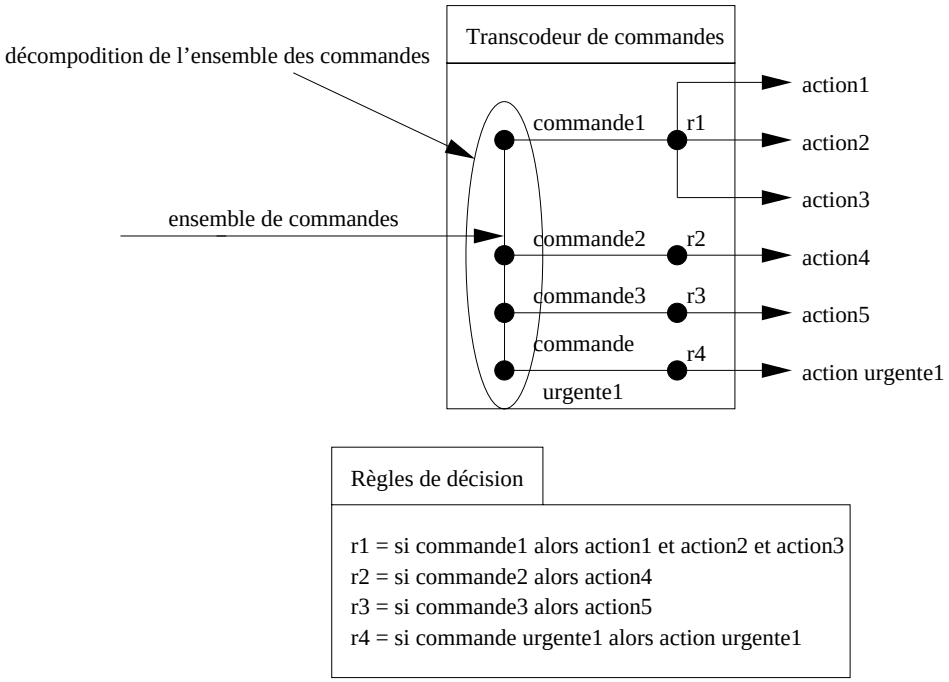


FIG. 3.9 – *Un transcodeur de commandes*

Les paramètres physiques

Comme l'instance environnementale est directement au contact de l'environnement, elle va contenir toutes les informations liées à l'agent, mais qui sont aussi liées à l'environnement. En effet, l'agent peut être amené à contenir des informations qui sont en rapport direct avec l'environnement. Cela pourra être, par exemple, un âge, une taille, un poids, une envergure ou bien encore une vitesse. Ces informations appelées *paramètres physiques* sont contenues dans une entité appelée *informations physiques*. Tous ces paramètres sont contenus dans un registre appelé *registre de paramètres physiques* et noté $\mathcal{RPP}$.

Ces paramètres contenus dans $\mathcal{RPP}$ vont être modifiés tout au long de la simulation soit par (1) le système conatif soit (2) par l'environnement.

1. Le cas d'une modification par le système conatif peut se présenter quand celui-ci prend une décision (*e.g* qui se traduit par l'envoi d'une commande) qui va entraîner une modification de ces paramètres. Dans ce cas, la commande qui va transiter le long du lien de dépendance bidirectionnel va être d'un type spécial qui va permettre à l'instance environnementale de le prendre en compte et donc d'effectuer la mo-

dification. Cette modification peut être vue comme une modification *intentionnelle* dans la mesure où elle découle d'une volonté réelle du système conatif.

2. Le cas d'une modification par l'environnement peut se présenter lorsque celui-ci doit modifier ces valeurs à cause de différents phénomènes liés à l'environnement. Dans ce cas, l'environnement va directement accéder à ces valeurs par le biais de l'entité d'informations physiques qui elle même contient $\mathcal{RPP}$. Et donc cette modification pourra être vue comme une modification *autoritaire* dans la mesure où le système conatif n'a pas pris la décision de cette modification et donc il la subit.

De la même manière que pour la modification, ces paramètres physiques peuvent être consultés par le système conatif ou par l'environnement.

- Dans le cas d'une consultation par le système conatif, cette opération va se faire par le biais d'une commande spéciale qui va être envoyée par celui-ci. Et donc l'instance environnementale va donc pouvoir comprendre ce message et finalement répondre à cette requête.
- Dans le cas d'une consultation par l'environnement, celui-ci va utiliser son accès à l'entité d'informations physiques pour accéder à $\mathcal{RPP}$ et donc aux paramètres.

La figure 3.10 page suivante illustre notre modèle d'instance environnementale.

Dans la prochaine section nous allons montrer quels sont les types des messages envoyés entre le système conatif et l'instance environnementale.

Le long du lien de dépendance

Tout au long de la présentation des diverses entités de notre modèle, nous avons souvent fait allusion au type de messages échangés entre le système conatif et l'instance environnementale (*cf* sections 3.2.1 page 48 et 3.2.1 et figures 3.10 page suivante et 3.7 page 48). Notre but dans cette section sera d'isoler et de mettre en évidence une typologie des messages qui sont échangés entre ces entités dans le but de pouvoir fournir à un concepteur, par le biais de notre architecture, une bibliothèque de messages prédéfinis pour des systèmes multi-agents classiques. Ceci étant bien entendu qu'un concepteur pourra toujours se créer ses propres messages et ceci grâce au système d'héritage [Mul97, RBEL91] du paradigme objet.

Tout d'abord, nous allons présenter les messages échangés dans le sens : système conatif $\longrightarrow$ instance environnementale. Et ensuite les messages échangés dans le sens : instance environnementale $\longrightarrow$ système conatif.

[Mul97] Pierre-Alain Muller. – *Modélisation objet avec UML.* – Eyrolles, 1997.

[RBEL91] J. Rumbaugh, M. Blaha, F. Eddy et W. Lorensen. – *Object-Oriented Modeling and Design.* – London, England, Prentice All International, 1991.

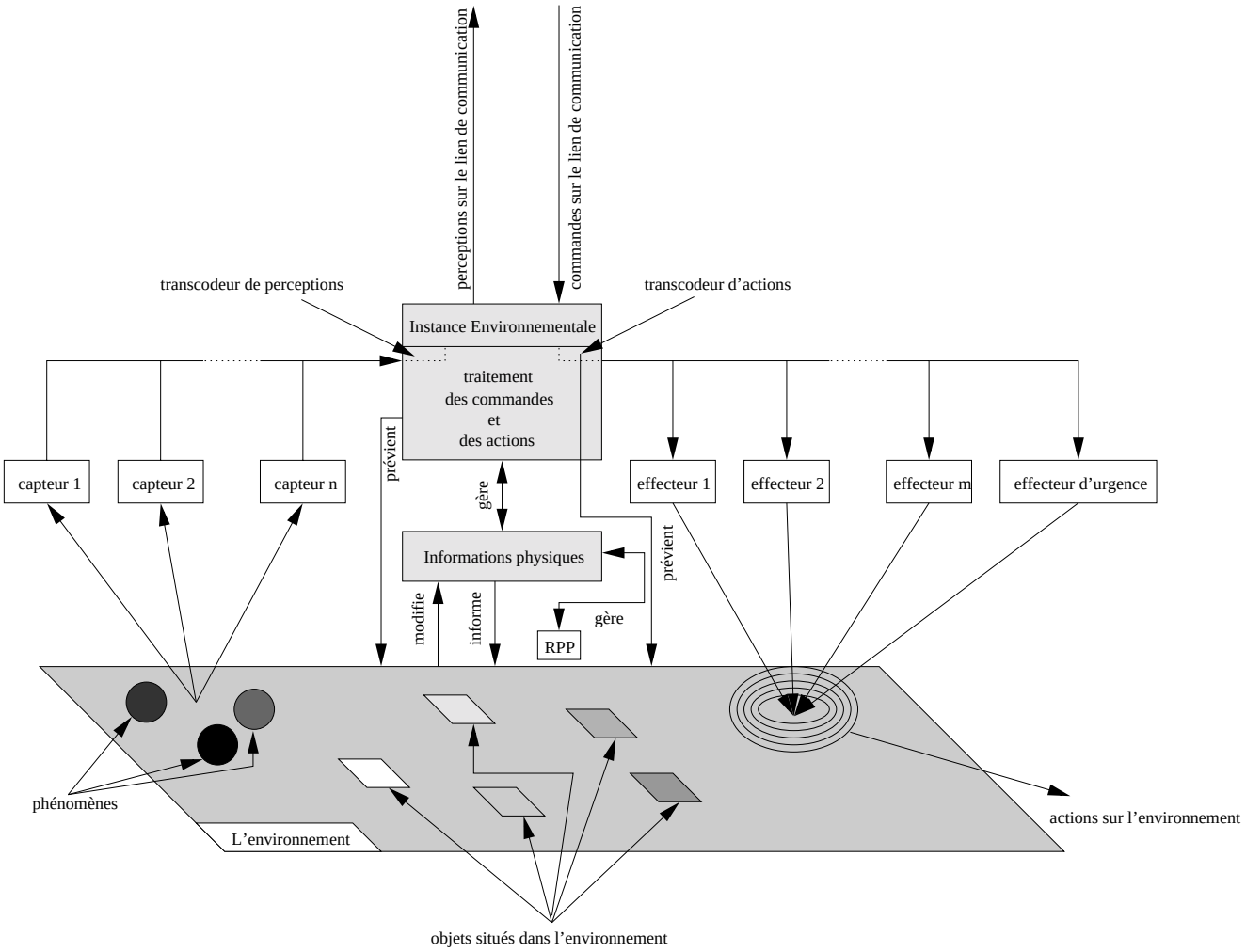


FIG. 3.10 – *Le modèle de l'instance dans l'environnement*

Système conatif $\longrightarrow$ instance environnementale

En reprenant la lecture des informations ci-dessus, on peut voir que le premier message qui est échangé entre le système conatif et l'instance environnementale est un message de demande de description de l'environnement. C'est donc un *message de consultation environnementale*. Ce type de message va être aussi utilisé quand le système conatif veut connaître des valeurs contenues dans $\mathcal{RPP}$. De la même manière, on peut isoler deux autres types de messages de haut niveau :

– Un *message de commande*. Celui-ci est généré lorsque le système a décidé (à la suite d'un processus de raisonnement plus ou moins long ou à la suite de stimulus extérieur ou bien encore à cause de sa fonction de satisfaction) d'envoyer une commande à l'environnement. Et dans ce cas, il va donc déposer le message dans $\mathcal{RC}$ et le gestionnaire de priorités va le ranger dans $\mathcal{RCO}$. Le type de ce genre message sera

donc : `commande`.

- Un *message de commande urgente*. Celui-ci est généré lorsque le système conatif désire qu'une commande soit réalisée le plus vite possible dans l'environnement. Dans ce cas, le système conatif va déposer le message dans $\mathcal{RC}$ et le gestionnaire de priorités va le ranger dans $\mathcal{RCU}$. Le type de ce genre de message sera donc : `commande urgente`.

Dans tous ces cas, les messages peuvent contenir et/ou renvoyer des types différents en plus de leurs types principaux. On peut, au niveau le plus générique, isoler deux principaux types utilisés : un type numérique ou un type symbolique.

Concrètement, les messages sont, dès le départ, typés par leur type de base : `commande`. Ensuite, on va trouver dans le message une description de la commande : par exemple, une commande de déplacement, une commande de rotation ou bien encore une commande d'envoi de message. Puis lorsque la commande va être traitée par le gestionnaire de priorités, on va ajouter les informations concernant la priorité dans ce message (et éventuellement une information concernant son caractère urgent). Ensuite, l'ensemble des messages vont être ordonnés selon leurs priorités et mis bout à bout. Puis finalement, cet ensemble de messages va arriver au transcodeur de commandes après avoir transité le long du lien bidirectionnel de dépendance. Le transcodeur de commandes va extraire une à une toutes les commandes c_i et ensuite il va les transformer en des action a_1,a_2,...,a_{n-1},a_n qui vont être directement appliquées sur l'environnement. On peut remarquer que ce système d'enrichissement successifs des paquets d'informations peut être très facilement rapproché des techniques bien connues dans le monde des réseaux et plus particulièrement le fonctionnement du modèle OSI [Lem96a].

Dans le cas d'un message de consultation environnementale, le fonctionnement de ce mécanisme est le même, ceci à la différence près que ce type de message implique une réponse immédiate de la part de l'instance environnementale et donc le système conatif va attendre cette réponse. Dans le cas où cette réponse n'arriverai pas, le système conatif va générer à chaque pas de simulation la même commande jusqu'à ce que sa demande soit satisfaite. C'est pourquoi ce genre de message possède un type différent des autres : `consultation env`. Lorsque le gestionnaire de priorités détecte un message de ce type, il va quand même lui donner une priorité, mais afin de pouvoir les distinguer, il va lui donner comme priorité : -1. De ce fait notre ensemble des priorités défini en section 3.2.1 page 41 se trouve enrichie par une nouvelle valeur. Donc l'ensemble des priorités potentielles est : $\mathbb{N}^* \cup \{-1\}$ et plus précisément : $[1,\ldots,m] \cup [1,\ldots,m'] \cup \{-1\}$. Finalement, le gestionnaire de priorités va ajouter cette commande à la fin de l'ensemble des messages possédant une priorité $p \geq 1$ ou étant urgent. De ce fait, le transcodeur de commandes va pouvoir facilement identifier les commandes à transformer en actions des messages de consultation environnementale : lorsque celui-ci va pouvoir lire une priorité $p = -1$ en entête d'un message, il va comprendre que le reste des messages à lire sont

[Lem96a] Jean-Paul LEMAIRE. – *Les protocoles de réseaux.* – Hermès, 1996.

des messages de consultation environnementale et donc qu'il va falloir y répondre le plus rapidement possible.

Le figure 3.11 illustre ce mécanisme.

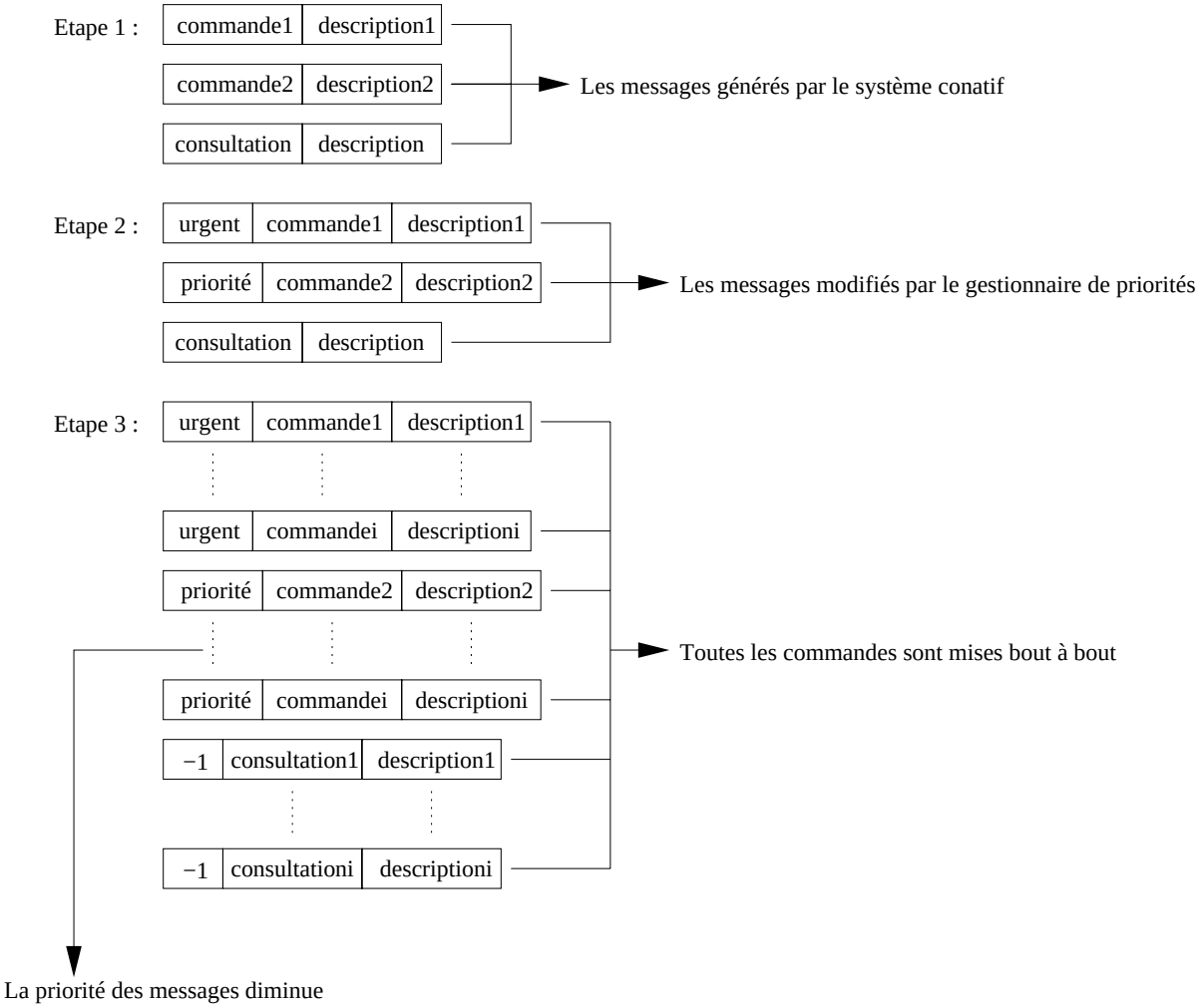


FIG. 3.11 – *Le mécanisme d'enrichissement et de traitement des messages*

De plus, les tables du gestionnaire de priorités doivent contenir ces nouveaux messages avec une priorité à -1 comme illustré dans la table 3.3 page ci-contre.

Instance environnementale $\longrightarrow$ système conatif

De manière symétrique, la première sollicitation à laquelle doit répondre l'environnement est un message provenant du système conatif lui demandant de lui décrire l'environnement dans lequel il se situe afin de pouvoir commencer de construire sa représentation locale de l'environnement. Le message renvoyé va donc être une réponse à un message de

Commandes	Types	Priorités
c_1	$t_{a'}$	pr_a
c_2	$t_{b'}$	pr_b
$\vdots$	$\vdots$	$\vdots$
c_n	$t_{m'}$	pr_m
$consultation_i$	t_i	-1
$\vdots$	$\vdots$	$\vdots$
$consultation_{i+\alpha}$	$t_{i'}$	-1

TAB. 3.3 – *Une table générée par le gestionnaire de priorités*

consultation environnemental. Ce message contenant la description de l'environnement va donc être stocké dans $\mathcal{RP}$. De plus, on va pouvoir isoler deux types principaux de messages :

- Un *message de perception*. Celui-ci est généré lorsque l'environnement doit informer le système conatif qu'il y a eu une perception au travers des capteurs et donc il doit être tenu au courant de cette information. Ce message est déposé dans $\mathcal{RP}$, puis il est pris en charge par le gestionnaire de priorités du système conatif qui va le ranger dans $\mathcal{RPO}$. Le type de ce message sera donc : `perception`.
- Un *message de perception urgente*. Celui-ci est généré lorsque l'environnement doit informer le système conatif qu'il y a eu une perception par le biais des capteurs et donc il doit être tenu au courant de cette information. Ce message est déposé dans $\mathcal{RP}$, puis il est pris en charge par le gestionnaire de priorités du système conatif qui va le ranger dans $\mathcal{RPU}$. Le type de ce message sera donc aussi de type `perception`, mais c'est les gestionnaire de priorités qui va décider de son type final, à savoir `perception urgente`.

De la même manière que dans le sens système conatif $\longrightarrow$ instance environnementale, ces messages peuvent contenir et/ou renvoyer des types différents. A savoir au niveau le plus générique : un type numérique ou un type symbolique.

De manière plus concrète, les perceptions, dès qu'elles sont perçues par les capteurs au niveau de l'environnement, sont tout d'abord traitées par le transcodeur de perceptions qui les transforme chaque séquence de perceptions p_1,p_2,...,p_{z-1},p_z avec $z \in [1,\ldots,n]$ en une seule perception p_i. Ensuite l'instance environnementale va rajouter le type auxquelles elles appartiennent en entête du message : `perception`. Ensuite, toujours grâce à l'instance environnementale, ces commandes sont mises bout à bout. Et finalement l'instance environnementale va aussi prendre en charge le fait d'envoyer ce paquet de perceptions le long du lien bidirectionnel de dépendance. Ce paquet va être finalement déposé dans le registre $\mathcal{RP}$ du système conatif. Et ensuite, le gestionnaire de priorités va lire une à une les perceptions contenues dans le message et en fonction de ses tables et du système conatif, il va ajouter les priorités adéquates. Si le message prend un caractère urgent, il

va rajouter le type `urgent` dans l'entête du message. Enfin, le gestionnaire de priorités va réordonner les perceptions en fonction de leur caractère urgent et en fonction de leur priorité. De cette manière le système conatif va pouvoir interpréter les perceptions qu'il a reçu en tenant compte de toutes les priorités.

De plus, dans le cas d'une réponse à un message de consultation environnementale, l'instance environnementale va créer de nouveaux messages qui sont fonctions des requête envoyées par le système conatif. Puis lorsque cet ensemble de messages a été créé, ils vont être rajoutés à la fin du paquet de messages contenant des perceptions. De la même manière que pour le sens inverse, en entête de chaque réponses, le transcodeur de perceptions va ajouter une priorité de -1. De cette manière, lorsque le gestionnaire de priorités va lire le registre $\mathcal{RP}$, il va d'abord lire les perceptions urgentes puis celles possédant des priorités inférieures et dès qu'il va détecter une valeur à -1 dans le champ priorité, il va savoir que ce qui suit concerne des réponses à des messages de consultation environnementale.

Le figure 3.12 illustre ce mécanisme.

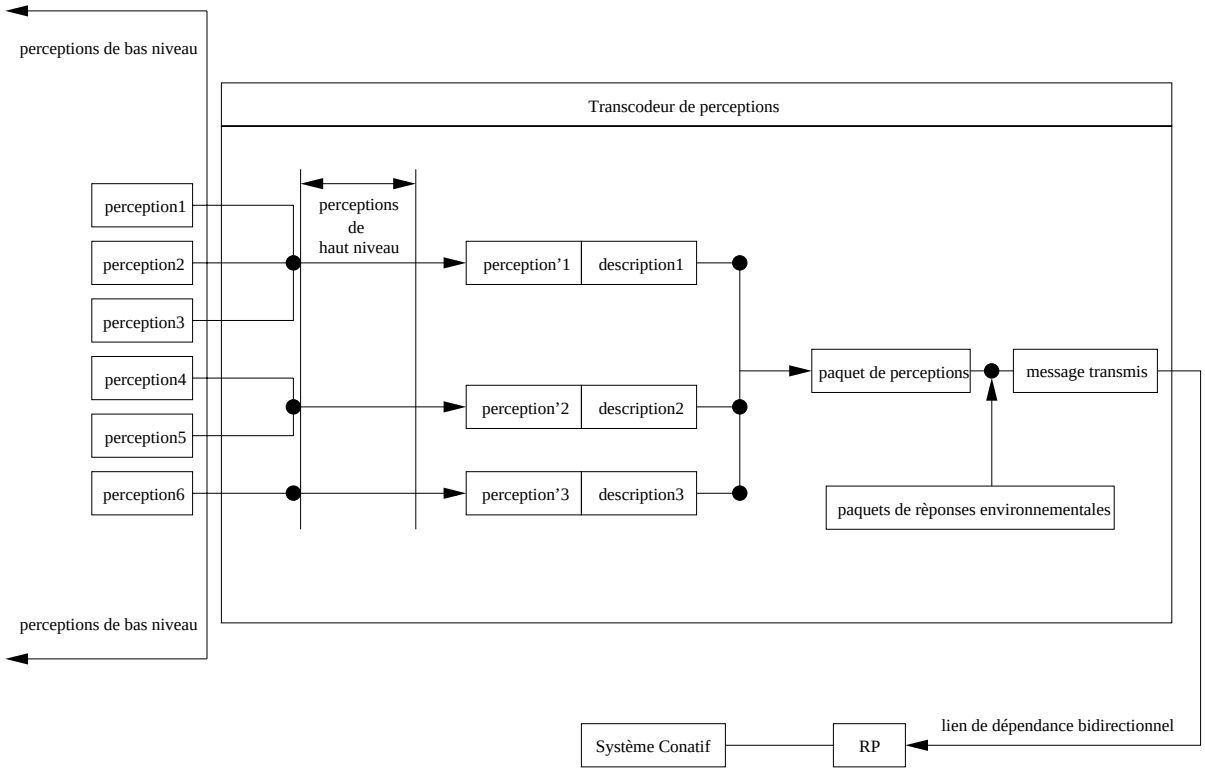


FIG. 3.12 – *Le mécanisme d'enrichissement et de traitement des messages*

Enfin, les tables du gestionnaire de priorités doivent contenir en plus ces nouveaux messages avec une priorité à -1 comme illustre dans la table 3.4 page ci-contre.

Cette manière de coder les messages nous amène à mettre en place une hiérarchie pour les messages échangés. Il est, en effet, possible de pouvoir dégager des caractéristiques communes à des niveaux plus ou moins élevés de granularité. Tout au sommet de

Perceptions	Types	Priorités
p_1	$t_{a'}$	pr_a
p_2	$t_{b'}$	pr_b
$\vdots$	$\vdots$	$\vdots$
p_n	$t_{m'}$	pr_m
$perception_i$	t_i	-1
$\vdots$	$\vdots$	$\vdots$
$perception_{i+\beta}$	$t_{i'}$	-1

TAB. 3.4 – *Une table générée par le gestionnaire de priorités*

la hiérarchie, nous allons trouver un objet de type `message`. Cet objet est le plus général possible et ne possède qu'un seul attribut : `type`. Cet attribut va désigner le type du message qui va transiter le long du lien bidirectionnel de dépendance. Il va donc pouvoir contenir qu'un ensemble bien précis de valeur : {*perception, perception urgente, réponse consultation env, commande, commande urgente, consultation env, modification, réponse modification env*}. Quand une commande va être déposée dans $\mathcal{RC}$, cet attribut n'est rempli qu'avec des valeurs de type `commande`. Et c'est les gestionnaire de priorités qui va éventuellement modifier ces valeurs selon la connaissance qu'il possède grâce à sa table interne de priorité ; c'est donc lui qui va changer le type `commande` en type `commande urgente` ou bien encore en `consultation env`. De la même manière, quand un ensemble de perceptions vont être captées par l'instance environnementale, c'est le transcodeur de perceptions qui va mettre dans l'attribut `type` la valeur `perception`. Mais à ce niveau, donc avant que l'ensemble des perceptions aient été déposées dans $\mathcal{RP}$, on ne peut pas encore savoir de quel type précisément est cette perception. C'est pourquoi, le transcodeur de perceptions ne va pas pouvoir faire la différence entre une `perception`, une `perception urgente` ou bien encore un `réponse consultation env`. C'est le gestionnaire de priorités qui va détecter, lorsque le transcodeur de perception va avoir déposé dans $\mathcal{RP}$ les perceptions, et en fonction de ses tables internes, quelles sont les valeurs réelles qui vont être attribuées à chaque perceptions. On peut donc voir le rôle essentiel que va jouer le gestionnaire de priorités dans le typage de ces messages.

Mais le gestionnaire de priorités ne va pas seulement donner des types aux messages qui vont ainsi transiter le long du lien de dépendance bidirectionnel. Il va aussi, en effet, changer le type de l'objet. Quand un objet va être d'un type plutôt qu'un autre, le contenu des informations qu'il va transporter ne va pas être le même et c'est pourquoi nous devons prévoir un ensemble d'objets messages qui puissent jouer ce rôle. Dans ce but, nous avons créé sept sous-objets à l'objet `message` :

- Un objet `consultation env`. Cet objet va contenir des informations nécessaires à la consultation du paramètre environnemental voulu. Il va donc contenir une méthode qui va être appelée `getParam`. Celle-ci va prendre en paramètre le type du paramètre que l'on va désirer consulter. De cet objet, nous pouvons isoler trois nouveaux objets

qui sont de granularités inférieures :

- Un objet appelé `consultation env symbolique`. Cet objet va être créé lorsque la requête émise par le système conatif pour son instance environnementale associée concerne une valeur qui va être de type symbolique. Ceci afin de permettre au transcodeur de commandes de reconnaître plus facilement l'opération qu'il va devoir effectuer.
- Un objet appelé `consultation env numérique`. Cet objet va être créé lorsque la requête émise par le système conatif pour son instance environnementale associée concerne une valeur qui va être de type numérique. Ceci afin de permettre au transcodeur de commandes de reconnaître plus facilement l'opération qu'il va devoir effectuer.
- Un objet appelé `consultation env objet`. Cet objet va être créé lorsque la requête émise par le système conatif pour son instance environnementale associée concerne un objet composé dont on ne connaît pas encore le type. Cet objet va être utilisé comme base pour tous les messages plus spécifiques que l'on pourra trouver dans une simulation réelle. Ainsi, grâce au mécanisme d'héritage, on pourra créer des objets plus spécifiques en fonction des objets que l'on va manipuler lors de la simulation considérée.

- Un objet `modification env`. Cet objet va contenir des informations nécessaires à la modification d'un paramètre contenu dans l'instance environnementale associée au système conatif. Cet objet va contenir une méthode appelée `setParam`. Celle-ci va prendre en paramètre le type du paramètre environnemental que l'on va désirer modifier. De manière analogue à ci-dessus, on va pouvoir isoler trois nouveaux objets qui sont de granularités inférieures :

  - Un objet appelé `modification env symbolique`. Cet objet va être créé lorsque la requête émise par le système conatif pour son instance environnementale associée concerne une valeur qui va être de type symbolique. Ceci afin de permettre au transcodeur de commandes de reconnaître plus facilement l'opération qu'il va devoir effectuer.
  - Un objet appelé `modification env numérique`. Cet objet va être créé lorsque la requête émise par le système conatif pour son instance environnementale associée concerne une valeur qui va être de type numérique. Ceci afin de permettre au transcodeur de commandes de reconnaître plus facilement l'opération qu'il va devoir effectuer.
  - Un objet appelé `modification env objet`. Cet objet va être créé lorsque la requête émise par le système conatif pour son instance environnementale associée concerne un objet composé dont on ne connaît pas encore le type. Cet objet va être utilisé comme base pour tous les messages plus spécifiques que l'on pourra trouver dans une simulation réelle. Ainsi, grâce au mécanisme d'héritage, on pourra créer des objets plus spécifiques en fonction des objets que l'on va manipuler lors de la simulation considérée.

- Un objet `urgent`. Cet objet va contenir les informations éventuellement nécessaires pour définir le caractère urgent du message. Entre autres caractéristiques, cet objet va contenir un attribut de type booléen appelé `urgent` qui va être mis à vrai par le gestionnaire de priorités dans le cas d'une perception urgente (*e.g* lors du passage de $\mathcal{RP}$ à $\mathcal{RPU}$) ou d'une commande urgente (*e.g* lors du passage de $\mathcal{RC}$ à $\mathcal{RCU}$) et à faux dans tous les autres cas.
- Un objet `perception`. Cet objet va contenir une information supplémentaire appelée `priorité` qui va permettre de définir la priorité associée à cette perception. Comme nous l'avons précisé ci-dessus, c'est le gestionnaire de priorités qui va avoir la charge de donner cette priorité et c'est donc lui qui va créer l'objet `perception`.
- Un objet `commande`. Cet objet va contenir une information supplémentaire appelée `priorité` qui va permettre de définir la priorité associée à cette commande. Comme nous l'avons précisé ci-dessus, c'est le gestionnaire de priorités qui va avoir la charge de donner cette priorité et c'est donc lui qui va créer l'objet `commande`.
- Un objet `réponse consultation env`. Cet objet va contenir des informations nécessaires pour répondre à une commande de consultation environnementale. Il va donc contenir une méthode qui va être appelée `paramValeur:Valeur` qui, lorsqu'elle va être invoquée sur cet objet, va lui renvoyer la valeur associée au paramètre. De cet objet, nous pouvons isoler trois nouveaux objets qui sont de granularités inférieures :

  - Un objet appelé `réponse consultation env symbolique`. Dans ce cas, le type de la valeur retournée par `paramValeur` et qui correspond à une commande de type `consultation env symbolique` émise à un instant $t-1$ de la simulation, va être une valeur symbolique. Ceci va permettre au système conatif de pouvoir savoir dès la réception du message que la commande qu'il a émis à $t-1$ a été bien effectuée et que ce n'est pas la peine de réitérer l'envoi de cette commande à l'instant t.
  - Un objet appelé `réponse consultation env numérique`. Dans ce cas, le type de la valeur retournée par `paramValeur` et qui correspond à une commande de type `consultation env numérique` émise à un instant $t-1$ de la simulation, va être une valeur numérique. Ceci va permettre au système conatif de pouvoir savoir dès la réception du message que la commande qu'il a émis à $t-1$ a été bien effectuée et que ce n'est pas la peine de réitérer l'envoi de cette commande à l'instant t.
  - Un objet appelé `réponse consultation env objet`. Dans ce cas, le type de la valeur retournée par `paramValeur` et qui correspond à une commande de type `consultation env objet` émise à un instant $t-1$ de la simulation, va être une valeur de type objet. Ceci va permettre au système conatif de pouvoir savoir dès la réception du message que la commande qu'il a émis à $t-1$ a été bien effectuée et que ce n'est pas la peine de réitérer l'envoi de cette commande à l'instant t. De plus, cet objet va être utilisé comme base pour tous les messages plus spécifiques que l'on pourra trouver dans une simulation réelle. Ainsi, grâce au mécanisme d'héritage, on pourra créer des objets plus spécifiques en fonction des objets que l'on va manipuler lors de la simulation considérée.

- Un objet `réponse modification env`. Cet objet va contenir juste une valeur booléenne qui va indiquer si la modification a bien eu lieu au niveau de l'instance environnementale associée au système conatif. Ainsi, celui-ci va pouvoir comprendre si la modification a bien été effectuée. Si c'est le cas, le système conatif ne va pas répéter cette demande et si ce n'est pas le cas, il va réitérer sa demande de modification. Comme dans les cas précédents, nous pouvons isoler trois nouveaux objets qui sont de granularités inférieures :
  - Un objet appelé `réponse modification env symbolique`. Comme décrit ci-dessus, ce message va contenir une valeur booléenne qui sera a vrai dans le cas ou la modification a bien été réalisée et faux sinon. Ceci va permettre au système conatif de pouvoir savoir à quelle requête cette réponse est liée.
  - Un objet appelé `réponse modification env numérique`. Comme décrit ci-dessus, ce message va contenir une valeur booléenne qui sera a vrai dans le cas ou la modification a bien été réalisée et faux sinon. Ceci va permettre au système conatif de pouvoir savoir à quelle requête cette réponse est liée.
  - Un objet appelé `réponse modification env objet`. Comme décrit ci-dessus, ce message va contenir une valeur booléenne qui sera a vrai dans le cas ou la modification a bien été réalisée et faux sinon. Ceci va permettre au système conatif de pouvoir savoir à quelle requête cette réponse est liée.

Par contre, on peut noter que les tables du gestionnaire de priorités ne sont pas encore capables de gérer tous ces types de messages, c'est pourquoi nous devons les modifier comme présenté dans la table 3.5 page suivante afin qu'elles puissent réaliser cela.

Enfin, pour finir cette partie concernant les types et les objets que l'on crée lorsque de l'information circule sur le lien de dépendance bidirectionnel, on peut noter que cet ensemble de messages pré-définis peut être aisément enrichi pour des cas bien particuliers d'environnements précis. En effet, nous pouvons créer une bibliothèque de messages pour des environnements planaire (*e.g* en deux dimensions) ou bien encore des environnements dans l'espace (*e.g* en trois dimensions) ou finalement pour des environnements purement communiquants.

Par exemple, prenons la cas d'un environnement planaire. On peut facilement isoler des actions et des perceptions communes à ce genre d'environnement. Par exemple, nous pouvons avoir les messages suivants :

- `getPosition`. Dans ce cas, ce message va être créé en s'appuyant sur le message de base `consultation env` avec comme type `objet` car ce message va renvoyer un objet composé (typiquement un point composé de deux attributs : x pour l'abscisse du point dans le plan et y pour l'ordonnée du point dans le plan). Et sa priorité sera de -1 (*cf* la table 3.5 page ci-contre).
- `positionValue`. Ce message est celui associé au précédent. Lorsque l'instance environnementale (et plus particulièrement le transcodeur de commandes) va avoir pris

Messages	Types	Priorités
consultation env_1	{symbolique, numérique, objet}	-1
$\vdots$	$\vdots$	$\vdots$
consultation env_n	{symbolique, numérique, objet}	-1
modification env_1	{symbolique, numérique, objet}	-1
$\vdots$	$\vdots$	$\vdots$
modification env_m	{symbolique, numérique, objet}	-1
perception_1	{urgent}	
$\vdots$	$\vdots$	
perception_o	{urgent}	
perception_1	$\{\emptyset\}$	1
$\vdots$	$\vdots$	$\vdots$
$\text{perception}_{o'}$	$\{\emptyset\}$	o'
commande_1	{urgent}	
$\vdots$	$\vdots$	
commande_p	{urgent}	
commande_1	$\{\emptyset\}$	1
$\vdots$	$\vdots$	$\vdots$
$\text{commande}_{p'}$	$\{\emptyset\}$	p'
réponse consultation env_1	{symbolique, numérique, objet}	-1
$\vdots$	$\vdots$	$\vdots$
réponse consultation env_q	{symbolique, numérique, objet}	-1
réponse modification env_1	{symbolique, numérique, objet}	-1
$\vdots$	$\vdots$	$\vdots$
réponse modification env_r	{symbolique, numérique, objet}	-1

TAB. 3.5 – *La table principale du gestionnaire de priorités*

en compte celui-ci, elle va répondre à son système conatif en mettant la valeur du point dans ce message et en créant le message `positionValue` en s'appuyant sur le message de base `réponse consultation env` et avec un type `objet` car le point est un objet. Comme ci-dessus, sa priorité sera de -1.

– `setPosition`. Ce message va être créé en s'appuyant sur le message de base `modification env` avec comme type `objet` car ce message va prendre en paramètre un objet de type point. Sa priorité sera de -1.

– `assignementPosition`. Ce message est celui associé au précédent. Ce message va être créé en s'appuyant sur le message de base `réponse modification env` avec comme type `objet`. Comme nous l'avons décrit précédemment, ce message va juste contenir une valeur qui sera à `vrai` ou à `faux` selon que l'opération d'assignation a été effectuée avec succès ou non.

- `bouge`. Ce message va être vu comme une action à effectuer sur l'environnement. Si, en effet, le système conatif pour une raison précise décide d'effectuer un déplacement au sein de l'environnement, il va déposer cette commande dans le registre $\mathcal{RC}$. Puis le gestionnaire de priorités en fonction de sa table de priorités va lui assigner une priorité numérique ou alors urgente, et enfin celui-ci va se charger de faire transiter cette commande sur le lien bidirectionnel de dépendance. Comme c'est une commande, ce message va être construit grâce au type de message `commande`.
- `valeurPhysique`. Ce message va être vu comme une perception, mais seulement dans le cas où la valeur contenue dans le message n'est pas une valeur contenue dans l'instance environnementale de l'agent. Dans ce cas donc, ce message sera créé en s'appuyant sur le type de base `perception`. Selon la simulation considérée, ce message pourra être urgent ou non et donc pourra avoir une priorité ou non. Comme nous l'avons décrit plus haut, ce message va être géré dans un premier temps par le transcodeur de perceptions, puis par le gestionnaire de priorités et ensuite par le système conatif.

La figure 3.13 est un diagramme illustrant de notre hiérarchie de messages.

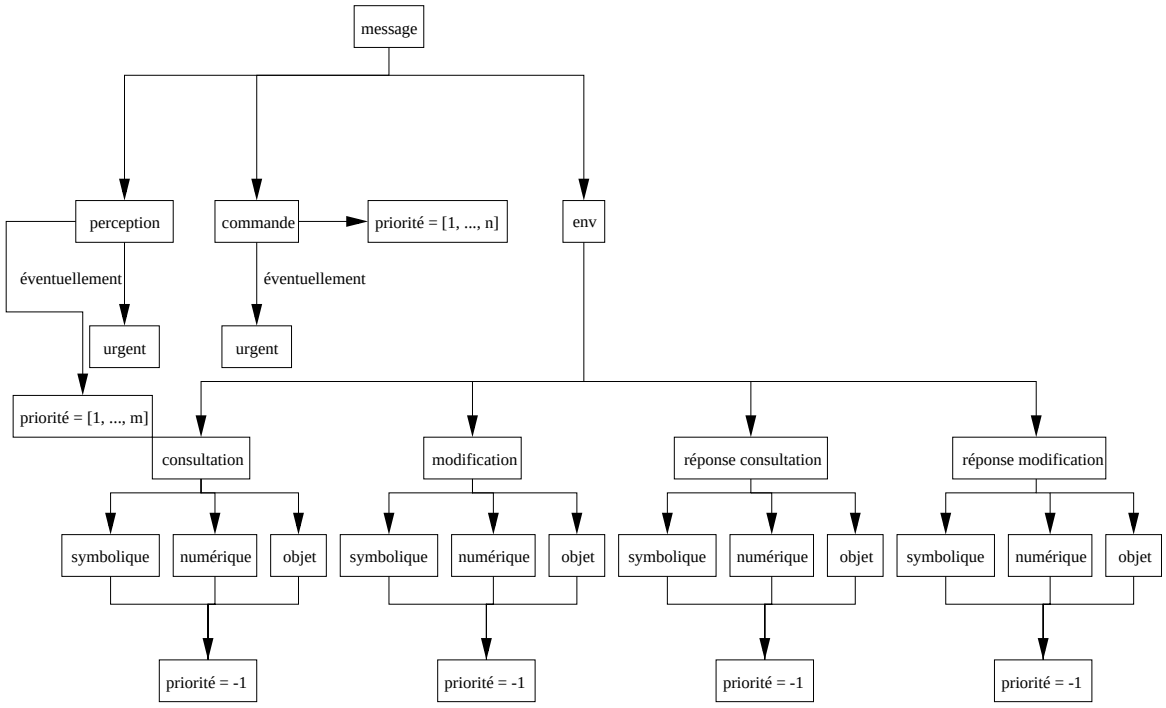


FIG. 3.13 – *La hiérarchie de message*

Dans la prochaine section nous allons présenter comment l'environnement peut être représenté dans notre architecture et comment sont pris en compte tous les différents paramètres liés à celui-ci.

3.2.2 Au niveau de l'environnement

Comme nous l'avons présenté dans la section 3.1 page 31, l'environnement est très étroitement lié aux agents dans le système multi-agents. Pour s'en convaincre, il suffit de voir notre représentation de l'agent où celui-ci est décomposé en deux entités, dont une appelée instance environnementale est directement plongée dans l'environnement. C'est pourquoi, dans cette section nous allons nous attacher à présenter comment on peut modéliser cet environnement et comment on peut prendre en compte tous les différents phénomènes qui peuvent jouer un rôle dans celui-ci.

Dans un premier temps, il faut isoler les deux principaux types d'environnement que nous pouvons rencontrer lors d'une simulation multi-agents : un environnement de type graphe dans le cas d'un système multi-agents purement communiquant et un environnement de type représentation d'un environnement réel ou virtuel dans le cas d'un système multi-agents situé. Nous allons donc étudier comment ces deux types d'environnements peuvent être modélisés dans notre étude.

L'environnement de type graphe

Dans cette partie, nous utiliserons les notations classiques liées aux graphes [BB83, Lee92], à savoir :

Définition 3 *On appelle* ***graphe*** *:*

- *un couple* $< X, \Gamma >$*, où* X *est un ensemble quelconque, et où* Γ *est un sous-ensemble de* $X \times X$ *;*
- X *est appelé ensemble des* ***sommets*** *du graphe ;*
- Γ *est appelé ensemble des* ***arcs*** *du graphe ;*
- *Etant donné un arc* (x,y)*,* x *est l'****origine*** *et* y *est l'****extrêmité*** *de l'arc ;*
- *Un* ***chemin*** *est une suite de sommets* $x_0,x_1,\ldots,x_k$*, avec* $k \geq 0$*, telle que* $\forall i,\ 0 \leq i \leq k-1$*, il existe un arc d'origine* x_i *et d'extrêmité* x_{i+1}*. Et dans ce cas, la valeur de* k *représente la* ***longueur*** *du chemin pour le graphe considéré ;*
- *Un graphe orienté est un graphe où pour tout arc* (x,y)*,* $(x,y) \neq (y,x)$*.*

Dans notre modèle, nous utilisons des graphes orientés car le fait qu'il existe un arc entre deux instances environnementales x et y (*e.g* x et y sont en communication entre

[BB83] Pierre Berlioux et Philippe Bizard. – *Algorithmique – Construction, preuve et évaluation de programmes.* – Dunod informatique, 1983.

[Lee92] Jan Van Leeuwen. – *Graphs Algorithms*, chap. 10, pp. 525–631. – Elsevier Sciences, 1992.

eux) implique que x communique (ou a communiqué selon le degré de persistence) avec y. Mais l'inverse (*e.g* que y communique avec x) n'est pas forcément vrai.

Dans le cas d'un environnement de type graphe, les instances environnementales des systèmes conatifs ne sont munies que de capteurs et d'effecteurs de messages. L'environnement va donc posséder de manière évidente un registre qui va contenir toutes le instances environnementales. Ce registre appelé *registre d'instances environnementales* et noté $\mathcal{RIE}$. Ce registre va donc former l'ensemble des sommets du graphe, et donc $\mathcal{RIE} = X$ selon la définition ci-dessus. De plus, l'environnement va posséder l'ensemble des liens de communication qui peuvent exister entre les différentes instances environnementales dans un registre particulier appelé *registre d'accointances* et noté $\mathcal{RA}$. Ce registre va donc représenter le réseau d'accointances qu'il existe entre les agents. Hors de part la définition même des agents et des systèmes multi-agent, cette représentation peut-être amenée à évoluer au fil de la simulation au grès des différentes décisions prises par les agents et au grès des différentes perceptions qu'ils ont pu avoir. Le problème qui se pose ici peut être formulé par la question suivante : « *faut-il conserver toutes les traces de communication entre les agents tout au long de la simulation ?* ». Notre réponse dans ce cas est de laisser la possibilité au concepteur du système considéré de définir lui même ce qu'il veut éventuellement pouvoir modifier au cours de la simulation. C'est pourquoi l'environnement est muni d'un paramètre numérique appelé *persistence* qui va définir si les arcs du graphe doivent être persistant et si oui pendant combien de temps. On va pouvoir isoler trois types de valeurs bien distinctes pour ce paramètre :

- $persistence = -1$. Dans ce cas, la persistence des liens d'accointances entre les agents va être maintenue tout au long de la simulation.
- $persistence = 0$. Dans ce cas, la persistence des liens d'accoitances ne va pas être maintenue tout au long de la simulation. Et donc à chaque pas de simulation, ces liens vont être détruits par l'environnement.
- $persistence = i$ avec $0 < i \leq n$ et $n \in \mathbb{N}$. Dans ce cas, la persistence des liens d'accoitances va être maintenue que pendant un temps donné, à savoir : i. Quand l'environnement va détecter qu'un lien a été mémorisé à un instant t de la simulation et que maintenant la simulation se trouve à un instant $t + i$, l'environnement va détruire ce lien.

Ce paramètre pourra être, éventuellement, modifié par l'utilisateur tout au long de la simulation et on pourra ainsi moduler la persistence des arcs entre les différents nœuds du graphe. Et de fait, on pourra éventuellement utiliser cette persistence pour visualiser ou calculer le degré de communication qu'il peut exister entre les agents. Cette particularité peut être extrêment intéressante dans le cas où on veut justement pouvoir étudier ce genre de phénomène pour un agent ou si on veut étudier comment un agent va réagir si subitement, on lui efface tous les arc de communication qu'il peut posséder avec les autres agents. De plus, le fait de conserver les liens de communications qu'il existe entre les

agents nous amène à créer deux type d'actions de communication générique au système :

- Un message de type *envoi unidirectionnel*. Dans ce cas, l'environnement va envoyer le message juste à l'agent qui a été nommé par l'émetteur et seulement à celui-ci.
- Un message de type *envoi multidirectionnel*. Dans ce cas, l'environnement va non seulement envoyer le message à l'agent qui a été nommé par l'émetteur, mais aussi à tous ceux qui sont en communication avec l'émetteur. C'est-à-dire que pour un agent x qui émet un message, on va extraire un ensemble $\mathcal{E}$ qui va contenir tous les arcs de type (x,y) où $y \in \Gamma$ du graphe de communication et on aura donc $\mathcal{E} = \{(x,y_1),(x,y_2),\ldots,(x,y_{n-1}),(x,y_n)\}$ où $n \in \mathbb{N}$ représente le nombre d'agents qui sont en communication avec x. Puis on va envoyer le message à tous les agents $y_i \in \mathcal{E}$. Ce mécanisme va nous permettre de pouvoir prendre en compte le fait que le message destiné à un agent peut être aussi intéressant pour les autres avec lesquels ils communiquent. Or, cette particularité nous amène à modifier le transcodeur d'actions présenté dans la section 3.2.1 page 48 et ainsi que le gestionnaire de priorités car ceux-ci tels que nous les avons définis ne peuvent pas prendre en compte cette particularité. C'est pourquoi nous ajoutons dans la table du gestionnaire de priorités deux sous-types au type général *commande* : *commande unidirectionnelle* et *commande multidirectionnelle* et au transcodeur de commandes la capacité de comprendre ces messages en ne cherchant pas à les décomposer en actions de plus bas niveaux. De cette manière tout est en place pour pouvoir maintenant utiliser ces types de messages.

De manière opposée, nous sommes amenés à créer deux types de perceptions de communication génériques au système :

- Un message de type *perception unidirectionnelle*. Cette perception va être générée par l'environnement en réponse à un envoi unidirectionnel et va être déposée dans le capteur de l'instance environnementale. Ce type de message va permettre au système conatif de savoir qu'il est le seul à avoir reçu ce genre de perception. De la même manière que précédement, nous devons ajouter au trancodeur de perceptions la capacité de pouvoir interpréter ce genre de perception de telle sorte qu'il ne cherche pas à construire une perception de plus haut niveau avec celle-ci. Et enfin, nous devons ajouter dans la table du gestionnaire de priorités l'entrée correspondante à *perception unidirectionnelle* qui est un sous-type de *perception*.
- Un message de type *perception multidirectionnelle*. Cette perception va être générée par l'environnement en réponse à un envoi multidirectionnel et va être déposée dans le capteur de l'instance environnementale. Ce type de message va permettre au système conatif de savoir qu'il n'est pas le seul dans le système à avoir reçu ce genre de perception et donc de pouvoir appréhender le fait que d'autres agents sont en communication avec celui qui leur a envoyé le message. Ceci peut avoir une utilité dans la cas où le système cherche à déduire que des agents ont, en partie, les mêmes accoitances que lui. Comme pour le cas d'un envoi unidirectionnel, nous devons

ajouter au trancodeur de perceptions la capacité de pouvoir interpréter ce genre de perception de telle sorte qu'il ne cherche pas à construire une perception de plus haut niveau avec celle-ci. Et enfin, nous devons ajouter dans la table du gestionnaire de priorités l'entrée correspondante à *perception multidirectionnelle* qui est un sous-type de *perception*.

Mais, on peut aussi avoir besoin qu'un message soit destiné à tous les agents qui sont le long d'un chemin. C'est à dire, si un agent a envoi un message à un agent b qui possède lui-même un lien d'accointance avec un agent c (*e.g* (a,b) et $(b,c) \in \Gamma$), l'agent a peut souhaiter que ce message soit envoyé sur le chemin formé par $a \longrightarrow b \longrightarrow c$. Dans ce cas, l'agent a va générer un nouveau type de commande appelé *envoi de cheminement*. Lorsque l'environnement va recevoir ce type de commande, il va cherche dans Γ le chemin éventuel qu'il peut exister entre a et les autres agents et envoyer ce message à tous ces agents. De la même manière, l'environnement, quand il va envoyer ce type de message, va créer une nouvelle perception à déposer dans les capteurs des agent : une perception appelée *perception de cheminement*. Comme nous l'avons déjà fait fait remarquer ci-dessus, ces deux nouveaux types de messages nous amènent à modifier la table du gestionnaire de priorités afin qu'elle puisse appréhender ces nouveaux types de message. Finalement, nous modifions aussi les transcodeurs de perceptions et d'actions afin qu'ils n'essaient pas de transformer ces messages en actions ou perceptions de plus haut niveau.

De plus, lors du déroulement d'une simulation, celle-ci peut nécessiter de pouvoir dater chaque pas de la simulation. En d'autres terme, on veut pouvoir attacher une date réelle à chaque pas de simulation. C'est pourquoi l'environnement est muni d'un *minuteur* qui va cadencer et donc donner un rythme à la simulation. Grâce à ce dispositif, toutes les informations qui vont être générées, soit par l'environnement ou soit par le système conatif, vont correspondres à une date bien précise. On pourra donc faire évoluer une simulation tout au long de ce temps de simulation. De cette manière, il suffit de faire correpondre un temps t de simulation à un temps t' dans l'environnement (par exemple à un temps $t = 1$ sec correspond un temps $t' = 1$ jour dans la réalité du phénomène observé) simulé pour avoir des résultats de simulation qui expriment une réalité au niveau des phénomènes observés.

Concrètement, lors du déroulement de la simulation, à chaque pas de simulation t, l'environnement va réaliser les opération suivantes qui consistent en une boucle infinie :

1. Collecter tous les envois de messages (des actions) présents dans les effecteurs des instances environnementales des agents présents dans la simulation ;
2. Interpréter tous ces messages et selon qu'ils sont d'un type : envoi unidirectionnel, envoi multidirectionnel ou envoi de cheminement, générer l'ensemble $\mathcal{E}$ des agents candidats à une réception de message ;
3. Déposer dans tous les agents a_i de $\mathcal{E}$ les perceptions adéquates (perception unidirectionnelle, perception multidirectionnelle ou perception de cheminement) ;

4. Prévenir chaque instance environnementale de $\mathcal{E}$ qu'il y a eu une ou plusieurs perceptions qui ont été déposées dans leurs capteurs respectifs ;
5. Passer du temp de simulation t au temps de simulation $t + 1$;
6. Revenir à l'étape 1 page ci-contre.

De cette manière on est assuré que toutes les actions et toutes les perceptions relatives à un temps donné t de simulation vont être effectivement et exactement traitées au même moment par les agents et que ceci ne va pas se passer à un temps $t + 1$. De plus, chaque message contient une information supplémentaire qui peut être utile au système conatif pour prendre ses décisions car les messages expriment de par leurs types s'ls ont été envoyés directement à l'agent par un autre ou s'ls ont été envoyés de manière « détournée » par un autre agent.

La figure 3.14 illustre le modèle utilisé pour représenter des environnements à type de graphe.

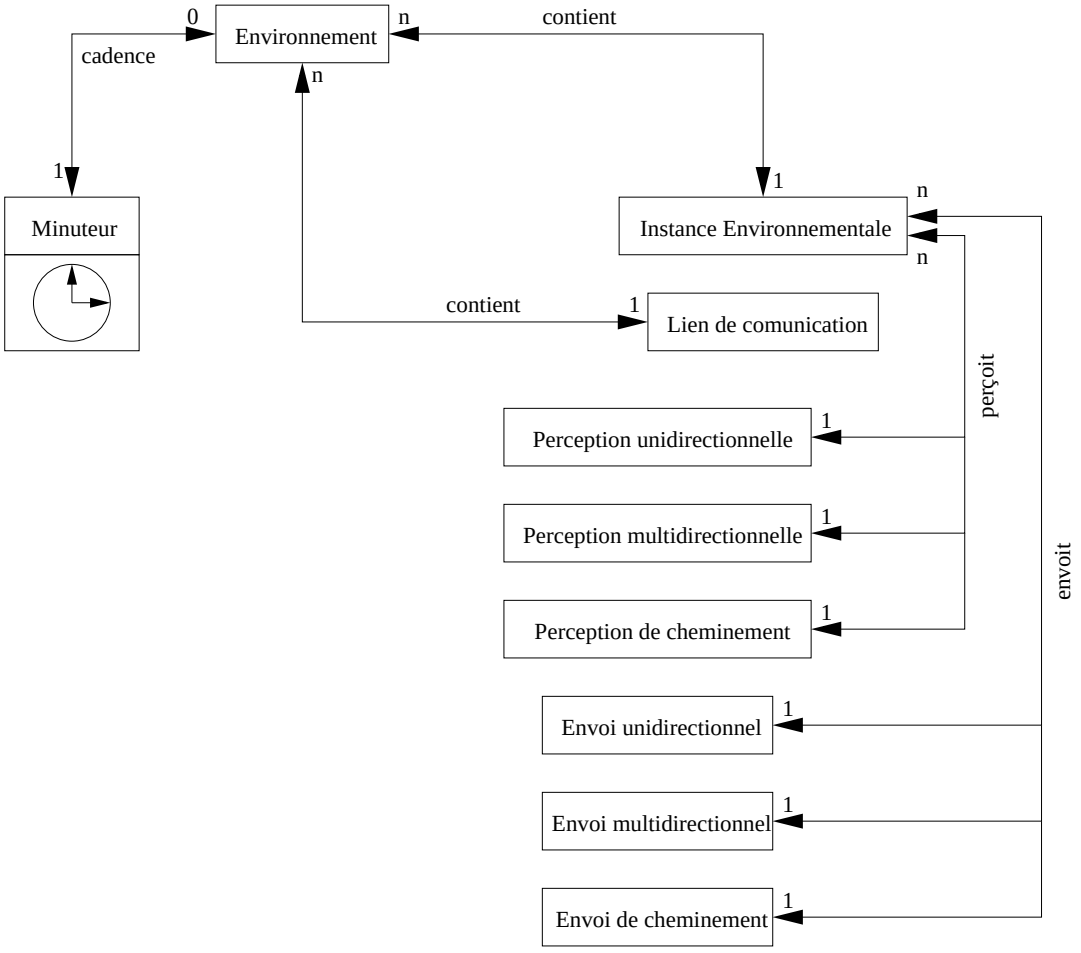


FIG. 3.14 – *Un modèle d'environnement à base de graphe*

Dans la prochaine section, nous allons présenter comment nous allons modéliser les environnements de type réels.

Les environnements de type réels

Ce type d'environnement, que ce soit en deux dimensions, trois dimensions, sous forme de grille ou bien encore d'automate cellulaire, sont largements utilisés dans les plateformes d'agents actuelles. Ils présentent, en effet, le grand avantage de pouvoir représenter le plus fidèlement possible les environnements que l'on souhaite modéliser et par là même pouvoir les prendre en compte pour la simulation. C'est pourquoi ils font l'objet d'un soin tout particulier lors de la phase de modélisation du système, car c'est l'environnement qui est le médium d'action et de perception entre les agents et il joue donc un rôle essentiel.

C'est pourquoi dans cette section nous allons nous attacher à fournir un modèle qui soit le plus générique possible et qui puisse tout de même prendre en compte toute la diversité des environnements de type réel que l'on peut rencontrer.

Tout d'abord, qu'est-ce que l'on appelle « *environnement* » ? Dans un premier temps, prenons la définition fournie par [Lar00] pour un environnement au sens général du terme :

> *... Ensemble des éléments constitutifs du paysage naturel ou du paysage artificiellement créé par l'homme. Par extension, cadre, contexte, circonstances de quelque chose ...*

Concernant les systèmes multi-agents, on peut reprendre la définition pour définir les environnement réels comme étant eux aussi les éléments constitutifs d'un paysage qui sera dans ce cas forcément artificiel ; ou bien encore dans le meilleur des cas, une représentation (une vue) d'un environnement physique réel. Dans ce cas, si on note S l'ensemble des éléments constituant un système multi-agents, et si on note A l'ensemble des agents de celui-ci, alors on peut dire que l'environnement E du système sera alors :

$$E = S - A$$

On peut donc dire que l'environnement, dans les systèmes multi-agents, est tout ce qui n'est pas un agent. Or dans notre modèle d'agent, celui-ci est défini comme la composition de deux entités : le système conatif et l'instance environnementale. Donc l'environnement va inclure les instances environnementales dans l'ensemble des objets qui le compose.

[Lar00] *Dictionnaire encyclopédique Hachette – Edition 2000.* – Hachette Livres, 2000.

Complexité de l'environnement

Du fait que l'environnement est par nature extrêmement hétérogène, les entités qui le compose sont donc elles aussi de types extrêment variés. Par conséquent, la modélisation et la prise en compte d'un environnement en entier devient très rapidement compliquée et très lourde en mémoire (pour l'ordinateur où se déroule la simulation). C'est pourquoi nous proposons dans cette section un mécanisme qui vise à réduire cette complexité.

Nous devons donc, dans un premier temps, définir une unité de mesure qui va nous permettre de comparer les différentes complexités de l'environnement. Pour cela, nous utiliserons le terme de « *porosité* » : c'est la mesure de la densité de cellules dans un paysage [FG86]. Comme référence, nous prenons un environnement E tel qu'il est défini par défaut et nous comptons le nombre de cellules dans celui-ci. La porosité P_E de E est donc $P_E = n \times m$ où n est le nombre de cellules qui composent la longueur de l'environnement et m est le nombre de cellules qui composent la hauteur de l'environnement.

Notre idée est d'agréger toutes les cellules proches du même type en une seule. C'est pourquoi, nous avons défini un « *critère d'agrégation* ». Au niveau le plus générique, on peut exhiber deux types de critères : un critère numérique et un critère symbolique. Pour notre simulation, nous avons choisi d'utiliser un critère numérique sur la couleur de la carte. En effet, une couleur de la carte satellite décrit la valeur précise d'une propriété (une température, une vorticité, ...). Nous utilisons donc cette couleur comme critère d'agrégation. L'opération d'agrégation se déroule donc de la manière suivante : quand des cellules voisines sont de même type (elles ont la même couleur) elles sont regroupées en une seule qui représente toutes les sous-cellules. Cette nouvelle entité, appelée *objet agrégé*, n'est pas obligatoirement du même type que les objets qui la compose car ces objets agrégés peuvent contenir plus d'informations que les sous-objets. Par exemple, si nous utilisons ce mécanisme pour réaliser l'agrégation d'un ensemble de briques, la nouvelle entité obtenue sera un mur, et celui-ci est complètement différent des sous-objets (forme, hauteur, largeur, ...).

Ce mécanisme réduit la porosité de l'environnement car quand un objet agrégé représente k ($k \in \mathbb{N}$ et $k \geq 0$) sous-objets de l'environnement (noté OA_k), alors :

$$P_E = (n \times m) - OA_k$$

Or nous pouvons avoir plusieurs objets agrégés dans l'environnment, donc au total, nous avons :

$$P_E = (n \times m) - \sum_{i=0}^{m} OA_{k_i}$$

où m est le nombre d'objets agrégés de l'environnement. Donc si nous notons $P_{E_{Agrégé}}$ la porosité d'un environnement agrégé, nous avons :

$$P_{E_{Agrégé}} \leq P_E$$

[FG86] R.T.T Forman et M. Godron. – *Landscape Ecology.* – John Wiley and Sons, 1986.

On peut noter que le pire des cas se produit lorsque tout l'environnement est totalement hétérogène, et donc il n'y a pas de possibilités d'agrégations et par conséquence, on a :

$$P_E = (n \times m) - OA_0 \text{ , or } OA_0 = 0$$

donc on obtient que $P_{E_{Agrégé}} = P_E$.

La Figure 3.15 illustre ce mécanisme.

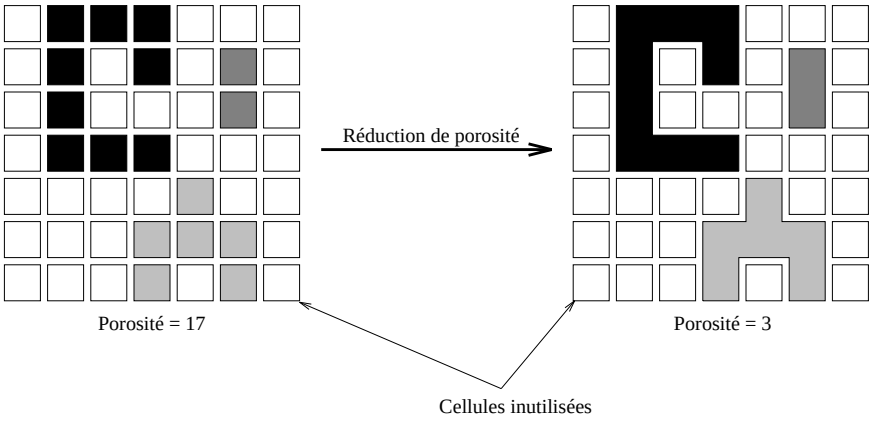


FIG. 3.15 – *Une réduction de porosité*

Dans notre simulateur, c'est l'environnement qui est chargé de réaliser les opérations d'agrégation. Donc, quand celui effectue à chaque pas de simulation les opérations présentées en 3.2.2 page 70, il doit en plus, effectuer les opérations d'agrégations.

De plus, ce mécanisme d'agrégation offre une autre possibilité intéressante pour l'utilisateur : il permet d'observer la simulation à des niveaux de visualisations différents. Il peut être, en effet, plus intéressant d'observer un phénomène à un niveau global plutôt qu'à un niveau très fin.

Quand l'environnement effectue une opération d'agrégation, l'opération sous-jacente réalisée consiste en un changement de repère dans l'environnement. Au niveau le plus bas (le niveau 1), l'environnement définit un repère Cartésien comme suit : $(O,\vec{i},\vec{j})$ – où O est l'origine du repère, $\vec{i}$ est le vecteur unité de l'axe des abscisses et $\vec{j}$ est le vecteur unité de l'axe des ordonnées – et après une opération d'agrégation qui agrège des objets d'un niveau n à un niveau $n+1$, le nouveau repère Cartésien est :

$$(O,(k * \vec{i}),(k' * \vec{j})) - k \in \mathbb{N} \text{ et } k' \in \mathbb{N} -$$

où k et k' sont définis par la taille du plus petit objet agrégé au niveau $n+1$. On peut réitérer cette opération jusqu'à ce qu'il n'y ait plus d'objets à agréger. De cette façon, on définit plusieurs niveaux d'observations qui correspondent en fait à la profondeur de l'environnement relativement aux objets agrégés.

Bien évidemment l'opération inverse est possible, et elle est appelée *opération de désagrégation*. Par exemple, dans le cas d'une opération de désagrégation d'un niveau n à

un niveau $n-1$, l'ancien repère Cartésien définit comme suit : $(O,\vec{\imath},\vec{\jmath})$, devient au niveau $n-1$:

$$(O,(-k*\vec{\imath}),(-k'*\vec{\jmath})) - k \in \mathbb{N} \text{ et } k' \in \mathbb{N} -$$

où k et k' sont définis par la taille de l'objet agrégé le plus grand au niveau $n-1$.

Chapitre 4

. . . À un modèle multi-environnemental

Sommaire

4.1 Introduction

Maintenant qu'il a été défini un modèle général pour représenter et gérer les interactions ainsi que les priorités entre les agents et l'environnement, nous allons, dans ce chapitre, proposer un modèle issu du précédent permettant de gérer des environnements multiples pour les agents.

Cette approche ne paraît pas forcément évidente au premier abord. Néanmoins, comme nous le décrit J. Ferber dans [Fer95], les analyses et développements de modèles multi-agents conduisent classiquement à deux types de systèmes multi-agents :

– Lorsque l'environnement, en tant que tel, est vide et que celui-ci contient des objets. Dans ce cas, on a un *un système multi-agents purement communiquant.*

[Fer95] Jacques Ferber. – *Les Systèmes Multi-Agents – Vers une intelligence collective.* – iia – InterEditions, 1995.

– Lorsque l'environnement est muni d'une métrique ainsi que d'un espace métrique on a *un système multi-agents situé.*

La problématique posée est la suivante : « comment faire pour gérer des agents qui appartiennent à la fois à un système multi-agents communiquant et à la fois à un système multi-agents situé ? » Il est tout à fait envisageable qu'on puisse se retrouver dans cette configuration. En effet, si on utilise un réseau d'accointances pour des agents qui ont aussi besoin de pouvoir capter des informations sur un environnement réel et de pouvoir aussi agir sur celui-ci, on va bien utiliser au moins deux environnements simultanément : le réseau d'accointances et l'environnement réel.

Pour ce faire, nous allons partir de notre modèle précédent et nous allons construire ce nouveau modèle multi-environnemental en plusieurs étapes :

– séparation des environnements ;
– accès aux données ;
– gestion du temps ;
– maintien de l'intégrité des données ;
– modifications du système conatif lié à l'apport des environnement multiples.

4.2 Séparation des environnements

Cette séparation va impliquer de nombreux changements au niveau du modèle présenté dans le chapitre précédent. Néanmoins, on peut isoler deux cas principaux dans cette séparation : le cas ou les données de l'environnement ne subissent pas intrinsèquement de modifications en fonction du temps et le cas ou les données de l'environnement évoluent en fonction du temps. C'est pourquoi nous allons maintenant présenter ces deux approches séparément.

4.2.1 Séparation des environnements sans prise en compte du temps

Tout d'abord, reprenons notre définition d'un agent définie dans le chapitre précédent (section 3.3 page 34). Un agent est défini comme présenté dans la figure 4.1 page suivante. Nous avons donc un système conatif et une instance dans l'environnement qui est chargé de l'action et de la perception. Le système conatif est chargé de la partie raisonnement et de l'autonomie, alors que l'instance dans l'environnement gère tout ce qui concerne l'environnement et lui seul. Entre ces deux entités, le lien bidirectionnel de dépendance

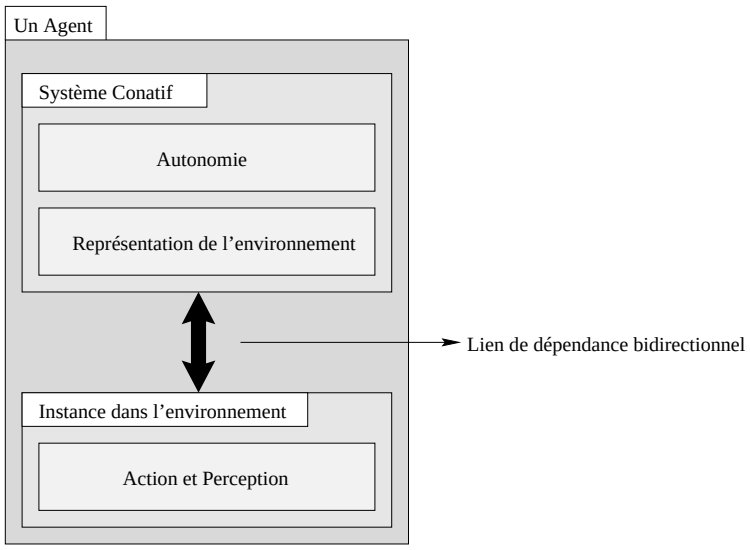


FIG. 4.1 – *Un agent avec ses deux parties et son lien de dépendance*

permet de faire transiter des informations entre le système conatif et l'instance dans l'environnement et inversement.

Or, nous voulons être capable de gérer plusieurs environnements (*i.e* au moins un environnement de type graphe d'accointances et un environnement réel). Il est donc nécessaire de pouvoir agir et capter des informations sur ces environnements. La seule méthode acceptable pour arriver à un tel résultat est de créer autant d'instances dans l'environnement que de types d'environnements que l'on doit gérer. C'est pourquoi ce modèle consiste dans un premier temps à créer pour chaque environnement de type t_i une instance dans l'environnement qui va correspondre à ce type t_i. Les principes d'actions et de perceptions sont conservés car tous les capteurs et les effecteurs qui sont contenus dans une instance dans l'environnement de type t_i sont effectivement capables d'agir sur l'environnement t_i. La figure 4.2 page suivante illustre cette première décomposition.

Ensuite, il faut être capable de pouvoir faire remonter les informations perçues par les instances dans l'environnement et, inversement, que le système conatif puisse envoyer des commandes aux instances dans l'environnement. Dans le premier sens, comme dans le deuxième sens, nous allons créer autant de liens de dépendances bidirectionnels que d'environnements. De la même manière que présenté dans le figure 4.2 page suivante, ces liens bidirectionnels vont posséder un type. Ce type sera le même que celui de l'environnement. C'est-à-dire qu'il pourra y avoir au moins un lien bidirectionnel pour un environnement réel et un lien bidirectionnel pour un environnement de type graphe.

Or, pour que le système conatif puisse gérer ces différents liens bidirectionnels, il faut que $\mathcal{RP}$ et par conséquent $\mathcal{RPO}$ soient capables d'accepter cette multitude de liens. C'est pourquoi $\mathcal{RP}$ sera équipé d'entrées directes sur les liens bidirectionnels. Quand $\mathcal{RP}$ va recevoir des informations, celles-ci seront automatiquement stockées dans ces entrées.

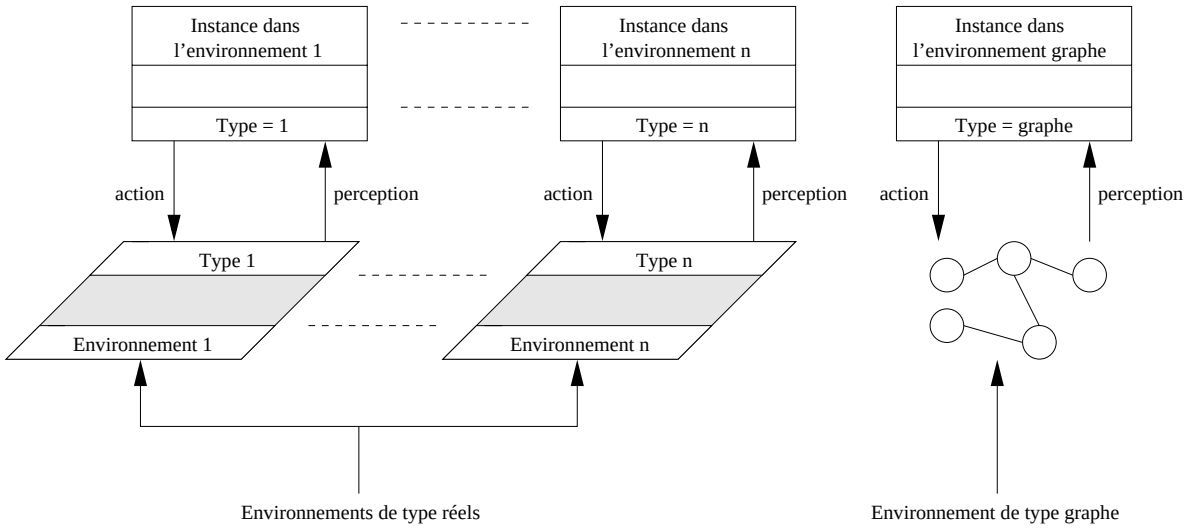


FIG. 4.2 – *Première étape de la décomposition : découpage des environnements*

Ces entrées, notées $\mathcal{ED}$ seront typées et leurs types seront les mêmes que celui des liens bidirectionnels (et par conséquent les mêmes que ceux des environnements). A chaque pas de simulation, avant que le gestionnaire de priorités ne fasse sont tri, $\mathcal{RP}$ va collecter toutes les informations contenues dans les différents $\mathcal{ED}$ pour créer un ensemble de perceptions. C'est cet dernier qui va maintenant être traité par le gestionnaire de priorités attaché au système conatif. On peut remarquer que, par définition de la décomposition en plusieurs environnements, les informations contenues dans les différents $\mathcal{ED}$ ne contiennent pas de doublons. En effet, les informations provenant d'un environnement donné étant fonctions du type de cet environnement, il ne peut pas y avoir deux fois la même information provenant d'environnements différents.

De la même manière, les commandes décidées par le système conatif de l'agent sont tout d'abord stockées dans $\mathcal{RC}$, puis dans $\mathcal{RCO}$ lorsque le gestionnaire de priorités a terminé sa tâche. De plus, il faut diriger les commandes sur les environnements appropriés, car si une commande qui doit être appliquée à un environnement donné se retrouve dans un autre environnement, celui-ci ne pourra pas l'exécuter. C'est pourquoi toutes les commandes sont étiquetées par le type t de l'environnement qui lui est associé. C'est pourquoi $\mathcal{RCO}$ contient, en fait, un ensemble de $\mathcal{RCO}$ dédiés appelés $\mathcal{RCO}_t$ où t est le type de l'environnement sur lequel une commande sera exécutée. De cette manière, lorsqu'une commande sera transmise au travers des liens bidirectionnels de dépendances, il est assuré que que cette commande possédera le bon type. Enfin, le gestionnaire de priorités aura la charge de faire (en plus de sa fonction initiale) ce travail de redirection des commandes dans les $\mathcal{RCO}_t$ appropriés.

Enfin, les transcodeurs (figures 3.8 page 50 et 3.9 page 52) de perceptions permettent de passer de perceptions de bas niveau à des perceptions de haut niveau et les transcodeurs d'actions permettent quant à eux de passer de commandes de haut niveau à des actions de bas niveau directement exécutables sur l'environnement eux aussi directement liés aux

types de l'environnement auxquels ils sont rattachés. En d'autres termes, des transcodeurs d'un type donné t ne peuvent agir sur des commandes ou des perceptions d'un autre type t'. On obtient alors une ensemble d'entités au niveau de l'environnement qui sont bien homogènes en terme de type, c'est à dire les données qui lui sont rattachées.

4.2.2 Séparation des environnements avec prise en compte du temps

Dans cette section, nous décrirons comment faire évoluer notre modèle pour qu'il prenne en compte le fait que tous les environnements peuvent évoluer et changer en fonction du temps. Ce cas de figure peut se rencontrer relativement fréquemment, notamment lorsque l'on dispose soit de données décrivant l'environnement à des instants t précis ou lorsque l'on sait décrire l'environnement en fonction d'équations qui sont elles-mêmes dépendantes du temps. C'est pourquoi il est fort important que le modèle proposé supporte ces cas de figures.

Dans le cas où ces environnements varient en fonction du temps tout au long de la simulation, il faut tout d'abord isoler deux cas distincts :

- Soit lors d'un pas de simulation, l'environnement ne change pas et dans ce cas, on se retrouve dans la situation décrite en 4.2.1 page 76. C'est à dire que l'on collecte juste les informations dans les capteurs appropriés et ensuite on effectue les commandes décidées par le le système conatif. Un fois ces opérations réalisées, on passe au pas de simulation suivant.
- Soit lors d'un pas de simulation, l'environnement doit changer les données qui le compose. Il va donc, dans un premier temps, mettre à jour ses données (soit en appliquant des équations, soit en effectuant un changement complet des données en allant les chercher dans une base de données ou dans une carte) et ensuite, il va effectuer la collecte des informations dans ce nouvel environnement pour les donner aux capteurs et, enfin, appliquer les commandes décidées par le système conatif sur ce nouvel environnement.

Donc, si on appelle t_c le pas de temps où s'effectuent des changements d'environnement, on aura donc pour un environnement donné ($e.g$ un environnement avec un type t), la totalité des environnements utilisés lors de la simulation pour un tel environnement t sera donc :

$$\text{environnement global} = \bigcup_{i=t_0}^{T} \text{environnement}_i$$

où $T = t_0 + (k \times t_c)$ et k est une constante ($k \in \mathbb{N}$). Et pour tout environnement$_i$ de

l'environnement global, nous avons :

- Un ensemble d'effecteurs $E_i = \{e_{1_i}, e_{2_i}, \ldots, e_{n-1_i}, e_{n_i}\}$ où n est le nombre d'effecteurs présents dans l'environnement$_i$;
- Un ensemble de capteurs $C_i = \{c_{1_i}, c_{2_i}, \ldots, c_{n-1_i}, c_{m_i}\}$ où m est le nombre de capteurs présents dans l'environnement$_i$.

Or, cette définition doit être élargie pour prendre en compte des environnements multiples. C'est-à-dire que pour un pas de simulation donné, nous aurons plusieurs environnements simultanément. Et ceux-ci, toujours à un pas de simulation donné, (éventuellement) fourniront autant d'informations. On peut noter que ce ne sont pas forcément tous les environnements qui retournent des informations systématiquement. En effet, il se peut très bien que sur l'ensemble des environnements utilisés, certains ne possèdent pas d'informations à ce pas de simulation. C'est pourquoi la définition précédente est élargie de la manière suivante.

Soit p le nombre d'environnement utilisés, l'environnement global utilisé pour la simulation sera donc :

$$\text{environnement global} = \bigcup_{j=1}^{p} \left(\bigcup_{i=t_0}^{T} \text{environnement}_{(i,j)} \right)$$

Dans ce dernier cas, l'environnement sera toujours muni d'un ensemble d'effecteurs et de capteurs. La seule différence majeure provient du fait que ces deux ensembles vont collecter (resp. appliquer) des données (resp. des commandes) sur des environnements qui ne seront pas les mêmes tout au long de la simulation.

On peut voir aisément que dans le cas d'un seul environnement, cette définition générale est toujours valable et il n'y a donc pas de limitation liée à cette généralisation. En effet, dans le cas où $p = 1$, on a :

$$\text{environnement global} = \bigcup_{j=1}^{p=1} \left(\bigcup_{i=t_0}^{T} \text{environnement}_{(i,j)} \right)$$

et on obtient donc :

$$\text{environnement global} = \bigcup_{i=t_0}^{T} \text{environnement}_{(i,1)}$$

De la même manière, dans le cas où le temps n'est pas pris en compte, on a :

$$\text{environnement global} = \bigcup_{j=1}^{p} \left(\bigcup_{i=t_0}^{t_0} \text{environnement}_{(i,j)} \right)$$

et on obtient donc :

$$\text{environnement global} = \bigcup_{j=1}^{p} \left(\text{environnement}_{(t_0,j)} \right)$$

Cette définition est donc suffisamment générale et permet de prendre en compte tous les cas de figures qui peuvent se présenter lors de la modélisation d'un système multi-agents.

Enfin, comme dans le cas où le temps n'est pas pris en compte, les transcodeurs de perceptions et d'actions sont toujours définis. C'est-à-dire qu'ils possèdent, eux aussi, le même type que l'environnement auquel ils sont attachés. Néanmoins, ils doivent pouvoir réagir aux variations temporelles éventuelles de leurs environnements respectifs. Ils devront être capables de s'adapter à de telles modifications. Le fait que les transcodeurs doivent éventuellement s'adapter à la modification de l'environnement relié est, en effet, très important. Par exemple, lorsque l'environnement est décrit par des images satellites, à chaque changement d'images, les données associées à la couleur des images, ne sont pas forcément les mêmes. De ce fait, les transcodeurs de perceptions et d'actions doivent être capables de s'adapter automatiquement à ces modifications.

4.3 Accès aux données

Les données, lors de simulations à base de systèmes multi-agents peuvent être de types complètement différents et hétérogènes. On pourra donc trouver des données de type graphe (section 3.2.2 page 65), des données provenant d'autres logiciels (section 2.6 page 23) ou tout simplement des données provenant d'un ensemble de données codées directement dans le simulateur. Bien entendu, on pourra, grâce au modèle présenté en section 4 page 75 utiliser une combinaison de ces types de données. Le propos ici, est d'isoler les mécanismes liés à ces accès aux données et de mettre en œuvre une architecture qui permette d'accéder facilement aux données nécessaires à la simulation multi-agents considérée.

Tout d'abord, on peut constater que chaque instance dans l'environnement possède un type qui lui est propre. De plus, de la même manière, l'environnement qui est relié à cette instance dans l'environnement possède le même type. Donc, chaque environnement est intrinsèquement rattaché à un type donné. Il est donc évident que l'accès aux données liées à cet environnement va se faire pour le type de données de cet environnement.

Pour ce faire, il faut donc équiper chaque environnement utilisé dans le modèle d'une interface appelée *interface d'accès aux données* qui va faire le lien entre les données qui vont être nécessaires pour les agents et les environnements. Néanmoins, cette interface devra prendre en le compte le fait que les données pourront soit :

être stockées : dans ce cas, l'interface d'accès aux données devra aller chercher automatiquement les données adéquates où elles sont stockées. Or, on peut recenser deux types principaux de stockage : les bases de données ou les fichiers. Donc l'interface d'accès aux données va posséder des primitives qui vont permettre d'utiliser ces formes de stockage.

calculées en temps réel à chaque pas de simulation : dans ce cas, l'interface d'accès aux données sera capable non pas, comme dans le premier cas d'accéder à une source contenant les informations, mais de calculer en permanence quelles sont les informations pertinentes pour l'environnement qui lui est rattaché. Dans le cas d'un environnement de type réseau d'accointances, l'interface d'accès aux données sera capable de fournir à l'environnement les données liées au graphe à chaque fois que ce sera nécessaire.

Dans cette approche, il faut aussi tenir compte de la phénoménologie (section 3.1.2 page 38) de l'environnement. En effet, toute ce qui peut survenir au sein de l'environnement et qui est lié à la phénoménologie est (ou peut être) soit stocké ou calculé. De la même manière que ci-dessus, l'environnement est relié à une interface qui fait le lien entre l'environnement est ces données liées à la phénoménologie. Cette interface appelée *interface de phénoménologie* va se comporter exactement de la même manière que l'interface d'accès aux données. En effet, dans le cas où la phénoménologie de l'environnement est décrite dans un (des) fichier(s) ou dans une (des) base(s) de données, il n'est pas nécessaire de calculer cette phénoménologie, il suffit juste d'aller chercher les données adéquates. Par contre, si la phénoménologie doit être calculée en temps réel à chaque pas de simulation (que l'on prenne en compte le temps ou non), les nouvelles valeurs qui découlent de cette phénoménologie devront être disponibles en permanence.

La figure 4.3 page suivante présente la deuxième étape de la construction du modèle multi-environnements.

4.4 Maintien de l'intégrité des données et gestion des conflits

La section précédente montre comment on peut construire un modèle à multiple environnements pour un simple agent. Mais cette architecture, telle qu'elle est définie ci-dessus, ne peut pas gérer l'intégrité des données sur l'ensemble des environnements et ainsi que les

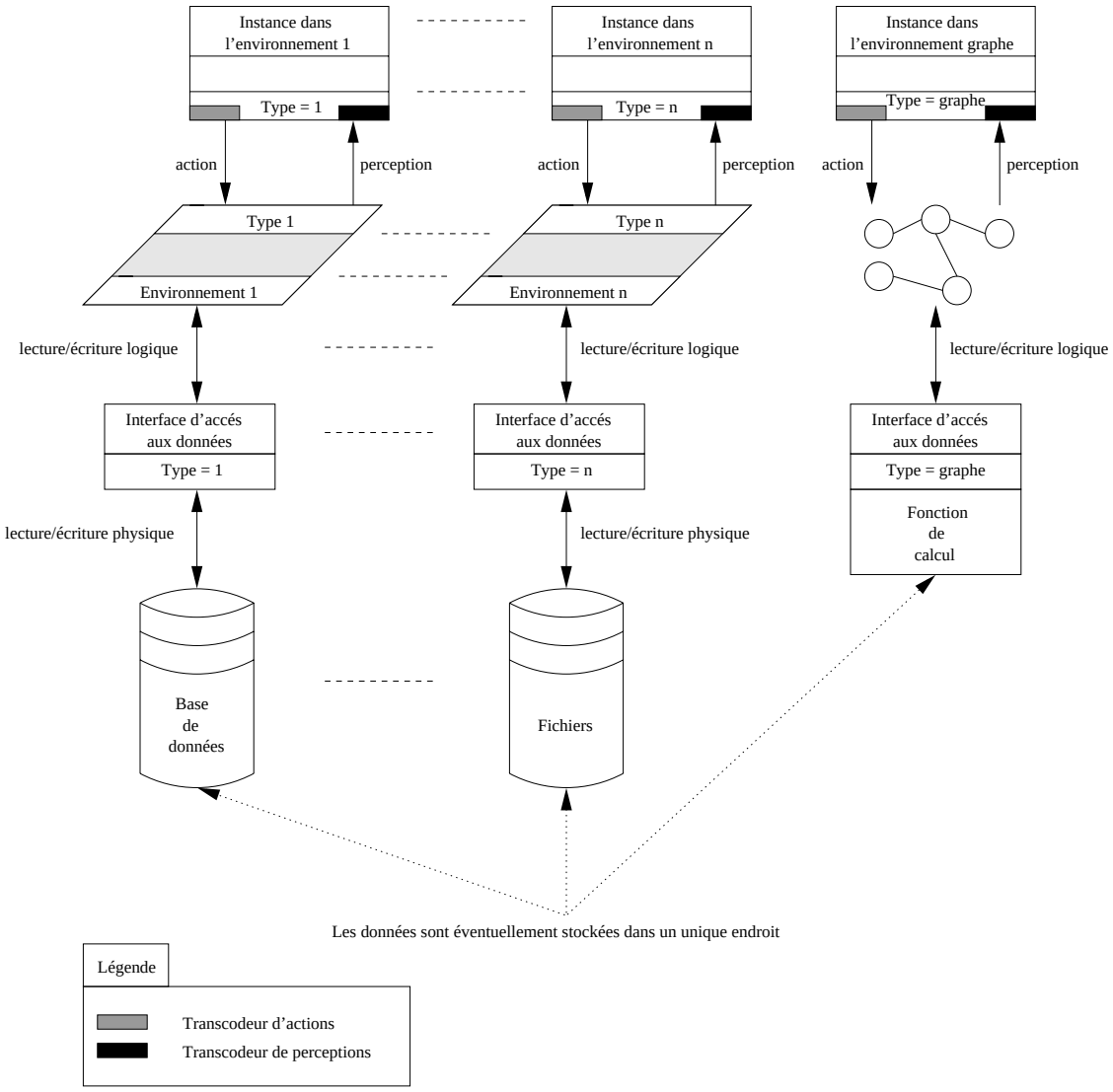


FIG. 4.3 – *Deuxième étape de la décomposition : accès aux données*

conflits qui pourraient apparaître. Par exemple, si le système conatif décide d'une action sur un environnement et que cette action implique des modifications non seulement sur cet environnement, mais aussi sur d'autres environnements, notre modèle va juste changer la valeur dans l'environnement concerné par l'effecteur et on obtiendra donc des environnements qui ne seront plus homogènes car ils auraient du être modifiés et ils ne l'ont pas été. Un autre exemple, plus concret, est lorsque le système conatif envoie un message contenant une action visant à effectuer une déplacement au sein de l'environnement. Comme cette action est typée en fonction de l'environnement où elle doit être exécutée, elle va donc juste être exécutée dans l'environnement concerné. Et dans ce cas, on peut obtenir après un nombre plus ou moins grand de pas de simulation que l'agent ne se situe pas au même endroit dans tous les environnements. C'est-à-dire que pour deux environnements

i et j, on n'aura pas forcément $position_i(agent) = position_j(agent)$[10]. Dans certains cas, cela est tout à fait normal car si on reprend la figure 4.3 page précédente, on peut voir que l'on a un environnement de type « graphe » et deux autres de type « environnement à deux dimensions ». Naturellement, il est normal que l'agent ne se situe pas à la même position dans le graphe que dans les images satellites, car la sémantique du mot position n'est évidemment pas la même pour un graphe que pour un environnement à deux dimensions. Si pour les deux environnements de type « environnement à deux dimensions », la position n'est pas la même, on pourra dire qu'il y a eu un problème dans la gestion de l'intégrité des données au sein des environnements.

Afin de pouvoir prévenir ces problèmes il faut rajouter dans le modèle une ou plusieurs entités qui auront la charge de gérer ce genre de conflits potentiels. C'est pourquoi, notre modèle a été étendu en lui ajoutant deux nouvelles entités :

- la première est appelée *instance dans l'environnement virtuelle* ;
- la seconde est appelée *environnement virtuel*.

L'instance dans l'environnement virtuelle

De la même manière que pour les instances environnementales, l'instance environnementale virtuelle est connectée au système conatif de l'agent et il contient un ensemble d'effecteurs et de capteurs. Cette instance est appelée « virtuelle » car elle n'est pas reliée à un environnement réel de la simulation, mais avec l'environnement virtuel qui est détaillé ci-dessous. L'instance environnementale virtuelle est reliée au système conatif de l'agent par un lien appelé *lien virtuel bidirectionnel de dépendance*. La fonction de ce lien est exactement le même que le lien bidirectionnel de dépendance qui relie le système conatif et les environnements, il ne doit son nom qu'au fait qu'il est relié à l'instance environnementale virtuelle et que les informations qui vont transiter sur ce lien ne concernent que l'instance environnementale virtuelle.

Concrètement, l'instance environnementale virtuelle contient les effecteurs qui auront une influence sur tous les environnements ou sur une partie des environnements (un effecteur de déplacement par exemple), les effecteurs qui sont présents dans toutes les autres instances environnementales des autres environnements. C'est à dire quand :

$$\bigcap_{i=1}^{n} \mathcal{EFF}_i \neq \emptyset, \forall i \in [1, \ldots n]$$

Et finalement les effecteurs qui sont nécessaires pour la simulation, mais qui ne sont pas rattachés à un environnement particulier.

10. $position_i$ signifie : "la position de l'agent dans l'environnement i"

De la même manière, l'instance environnementale virtuelle contient des capteurs qui sont présent dans toutes les instances environnementales. C'est à dire quand :

$$\bigcap_{i=1}^{n} \mathcal{CAP}_i \neq \emptyset, \forall i \in [1,\ldots,n]$$

Et les capteurs nécessaires à la simulation, mais qui ne sont pas rattachés à un environnement particulier.

De cette manière, on réduit considérablement le nombre d'effecteurs et de capteurs présent dans le modèle car on évite de dupliquer des effecteurs et des capteurs qui seraient présent dans tous (ou presque tous) les environnements. De ce fait le trafic sur les liens de dépendance bidirectionnels entre le système conatif et les instances dans l'environnement va être considérablement réduit. On peut isoler les messages en fonction de leur sens de transit :

Le sens système conatif $\longrightarrow$ instance dans l'environnement Dans ce sens, le système conatif va émettre des commandes qui doivent être exécutées sur l'environnement. Les commandes qui vont être spécifiques à un environnement donné vont transiter sur le lien bidirectionnel de dépendance adéquat. Par contre, les commandes qui vont être spécifiques à l'instance dans l'environnement virtuelle vont juste transiter sur le lien virtuel de dépendance bidirectionnel. Or, comme l'environnement virtuel contient des effecteurs qui se trouvaient auparavant dans tous (ou une partie) des environnements, le trafic lié à ces commandes va donc être considérablement réduit, car elles sont centralisées sur une seule entité.

Le sens instance dans l'environnement $\longrightarrow$ système conatif Dans ce sens, l'environnement va collecter des informations dans l'environnement et les envoyer au système conatif. Les informations qui vont être collectées sur des environnements autres que l'environnement virtuel vont donc transiter sur leurs liens de dépendance bidirectionnels respectifs. Par contre, les informations collectées sur l'environnement virtuel vont transiter sur le lien virtuel de dépendance virtuel. Or, comme l'environnement virtuel contient, entre autre, la réunion de tous (ou un partie) les capteurs qui étaient auparavant dans les autres environnements, le trafic lié à ces échanges d'informations se trouve considérablement réduit.

L'environnement virtuel

L'environnement virtuel sera relié avec l'instance environnementale virtuelle décrite ci-dessus. Par conséquent, tous les capteurs et les effecteurs de l'instance environnementale virtuelle vont être plongés dans cet environnement.

De plus, l'environnement virtuel est relié à tous les environnements « réels » de la simulation par le biais d'un lien appelé *lien d'intégrité*. Ce lien est un lien bidirectionnel et il va être utilisé pour :

- Maintenir l'intégrité des données sur l'ensemble des environnements. Par exemple, lorsqu'un effecteur de l'environnement virtuel exécute une action sur celui-ci, l'environnement virtuel va transmettre cette modification à tous les environnements grâce aux lien d'intégrité. Plus concrètement, imaginons que l'environnement reçoive une commande de changement de position. Or, lorsque l'environnement virtuel va avoir effectué ce changement de position, les instances dans l'environnement qui sont dans les autres environnements ne vont plus se trouver à la nouvelle position définie par l'environnement virtuel. Donc l'environnement virtuel va, tout d'abord, sélectionner les liens d'intégrités des environnements qui sont impliqués par ce changement de position et ensuite envoyer un message qui va prévenir les environnements que la position de l'agent à été modifiée.
- Informer les environnements lorsque qu'un environnement a effectué une opération qui puisse modifier des données dans les autres environnement. Par exemple, si un environnement effectue une opération qui entraîne des modifications dans les autre environnements, celui-ci va, tout d'abord envoyer un message à l'environnement virtuel. Lorsque ce dernier aura reçu ce message, il va sélectionner les environnements concernés par ce changement et transmettre le messages à ces environnements.
- Gérer le temps central et général du système (voir section 4.5 page 88)

Afin que l'environnement virtuel puisse mener à bien les tâches citées précédemment, il doit avoir une connaissance de ce que sont et ce que représentent les autres environnements. C'est pourquoi, lors de l'initialisation du système, chaque environnement présent dans le système va s'identifier auprès de l'environnement virtuel grâce à un message spécial qui va transiter le long des liens d'intégrité. Ce message sera exprimé sous la forme de balises « XML » [W3C] et sera de la forme :

```
<description>
   <type environnement>
   <actions>
      <action 1 --> consequence 1>
      <action 2 --> consequence 2>
      ...
      <action n --> consequence n>
   </actions>
   <sensibilite>
      <type 1 <--> action 1>
      <type 2 <--> action 2>
```

[W3C] The World Wide Web Consortium. – http://www.w3c.org/.

```
      ...
      <type n <--> action n>
   </sensibilite>
</description>
```

Lorsque l'environnement virtuel aura reçu l'ensemble de ces messages provenant des autres environnements, il va analyser toutes ces informations et déduire les règles qui lui permettent d'informer les environnements adéquats au bon moment.

Enfin, pour terminer avec les aspects liés à l'environnement virtuel, penchons nous maintenant sur les objets qui sont présents dans l'environnement. De la même manière que pour les environnements, chaque objet présent dans un environnement donné va posséder un type qui sera celui de l'environnement considéré. De cette manière, les objets vont donc se retrouver dans les environnements adéquats en fonction de leurs types. Ils seront donc sensibles à la phénoménologie liée à chaque environnement. Or, que faire des objets qui peuvent se trouver dans tous ou dans une partie des environnements ? Deux approches peuvent être envisagées :

- On laisse dans chaque environnement concerné les objets. De cette manière, on n'effectue aucun changement en ce qui concerne la gestion de ce objets. Ce faisant, on augmente considérablement le nombre d'entités dans le modèle. En effet, nous allons donc nous retrouver avec des objets en doublons au sein des différents environnements et par conséquence, les problèmes de mise à jour vont être importants. En effet, il faut pouvoir maintenir l'intégrité des objets dans tous ces environnements et s'ils sont disséminés dans un ensemble d'environnements, la mise à jour devient plus problématique à effectuer dans de bonnes conditions. C'est pourquoi cette approche ne nous a pas semblé judicieuse et nous ne l'avons pas utilisé.
- On décide de regrouper les objets de l'environnement qui se retrouvent dans tous (ou une partie) des environnements dans l'environnement virtuel. Ce faisant, il faut gérer les modifications qui peuvent intervenir au niveau de ces objets. Pour ce faire, nous allons utiliser les liens d'intégrités qui relient les environnements et l'environnement virtuel. Concrètement, lorsqu'un objet va être modifié au niveau d'un environnement, un message va être envoyé à l'environnement virtuel via le lien d'intégrité. Lorsque l'environnement virtuel reçoit ce message, il effectue la modification requise par l'environnement. C'est typiquement le cas lorsque la phénoménologie d'un environnement amène celui-ci à effectuer des modifications sur un objet. De manière inverse, si un effecteur de l'environnement virtuel effectue une modification sur un de ses objets, celui-ci va envoyer un message vers le ou les environnements qui sont concernés par cette modification via le lien d'intégrité. Cette approche est celle que nous avons choisi de développer dans notre modèle car elle est en adéquation avec l'ensemble du modèle que nous avons construit. De plus, bien qu'elle augmente l'échange de messages au niveau des liens d'intégrités, elle permet de simplifier le modèle et surtout, elle évite de dupliquer des objets dans un ensemble d'environnements.

Pour résumer, la figure 4.4 présente la troisième étape de la construction du modèle multi-environnements.

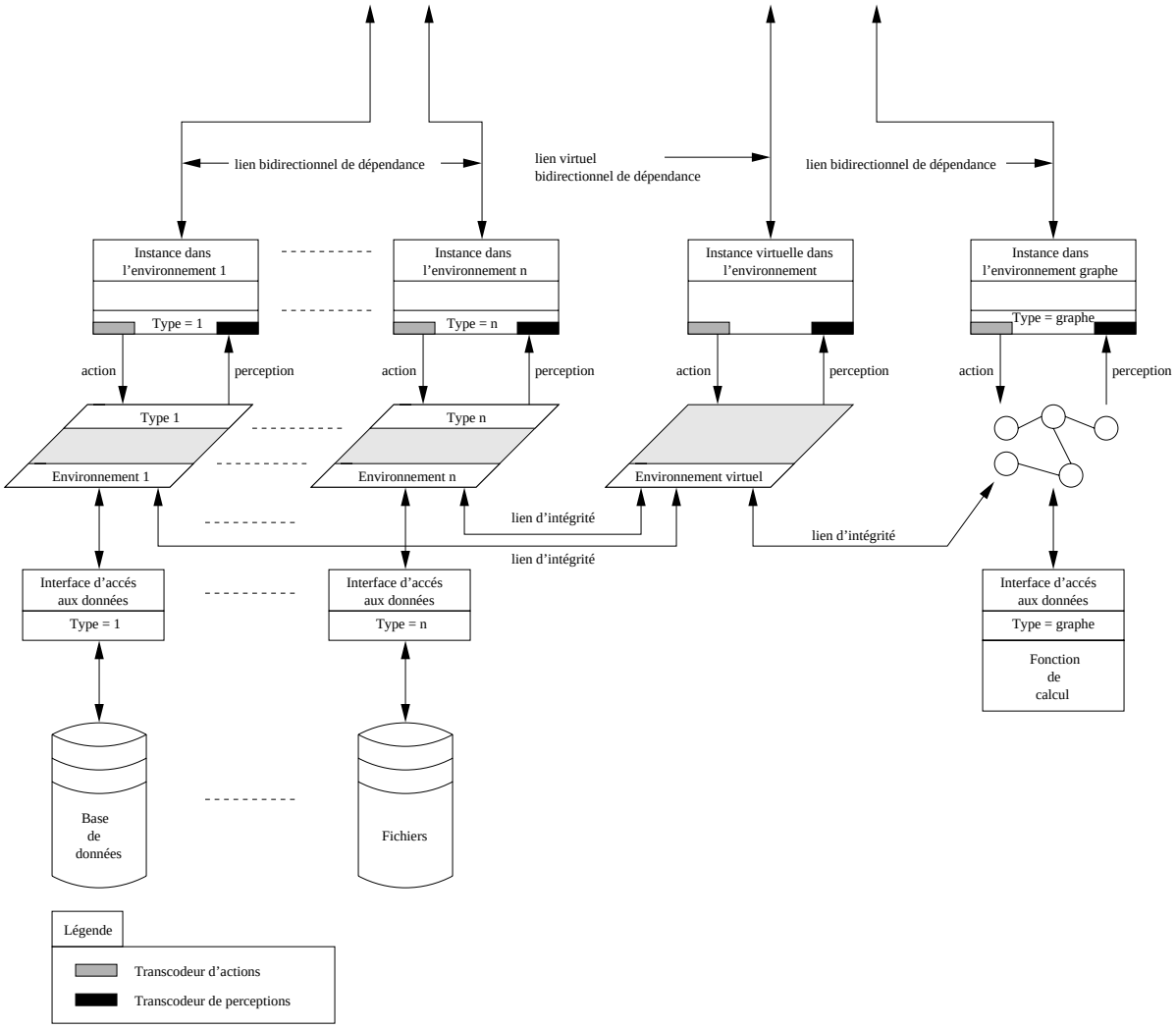


FIG. 4.4 – *Troisième étape de la décomposition : environnement virtuel et instance dans l'environnement virtuelle*

4.5 Gestion du temps

Dans la première approche mono-environnementale (section 3.2.2 page 65), le temps était géré classiquement par un minuteur au niveau de l'environnement. C'est-à-dire que c'etait ce minuteur qui envoyait des impulsions à l'environnement pour que celui-ci effectue les opérations suivantes :

– remplissage des capteurs des agents ;

- prise en compte des commandes décidées par les systèmes conatifs des agents ;
- modification éventuelle des valeurs contenues dans l'environnement à cause de la phénoménologie liée à celui-ci.

Or, suite aux modifications liées à l'apport du modèle multi-environnemental, nous sommes confrontés au problème suivant : *comment le temps peut il être géré dans cette multitude d'environnements ?*

Nous l'avons vu dans la section 2.7 page 27, dans les système multi-agents, on utilise communément : le temps centralisé et le temps dirigé par les évènements. Nous allons maintenant détailler pourquoi ces deux approches ne sont pas en adéquation avec notre approche multi-environnements.

Cas du temps centralisé

La première approche qui aurait pu être utilisée pour la plate-forme multi-environnements aurait été de considérer que l'on ne dote que l'environnement virtuel d'un minuteur. Dans ce cas, c'est lui qui se serait chargé d'envoyer à tous les environnements qui lui sont rattachés une impulsions qui préviendrait les environnements. Une fois que ceux-ci auraient reçu cette impulsion, ils effectueraient les opérations nécessaires.

Or, dans ce cas de figure, on se retrouve dans le cas de figure présenté par C. Le Page dans [PG97] : comment être sûr que l'on a bien trouvé la plus petite unité de temps qui puisse prendre en compte toutes les unités de temps les plus petites pour chaque autre environnement ? Pour répondre à cette question il faut effectuer une analyse fine du temps dans ces environnements et cette opération n'est pas forcément facile pour le concepteur du système multi-agents. De plus, cette manière de modéliser le temps dans une approche multi-environnementale n'est pas satisfaisante, car elle ne permet pas de prendre en compte le fait que le temps, pour chaque environnement, n'est pas forcément exprimé dans la même unité de temps et de plus, celui-ci n'a pas forcément la même signification d'un environnement à un autre. En effet, par exemple, si on prend le cas d'un environnement de type graphe et un autre environnement de type espace euclidien, le temps pourra ne pas avoir forcément la même signification si on se place dans le cas du graphe ou de l'espace euclidien. Donc pour toutes ces raisons, cette approche n'a pas été développée dans le cadre de notre plate-forme multi-environnements.

[PG97] Christophe Le Page et Vincent Ginot. – Vers un simulateur générique de peuplements piscicoles. *In : Proceedings Des 5èmes Journées Francophones d'Intelligence Artificielle et Systèmes Multi-Agents – JFIADSMA'97*, éd. par J. Quinqueton, M.C Thomas et B. Trousse. – Hermès, La Colle sur Loup, France, avril 1997.

Cas du temps dirigé par les évènements

La deuxième approche qui aurait pu être utilisée pour la plate-forme multi-environnements aurait été de considérer que tous les environnements utilisés dans le cadre de la simulation considérée s'enregistrent auprès de l'environnement virtuel afin que celui-ci garde dans une mémoire la liste des impulsions qu'il doit donner à tous les environnements.

Or, dans ce cas de figure, on suppose que l'environnement, entre deux périodes où il a déposé un évènement auprès de l'environnement virtuel, n'évolue pas. Cette supposition n'est pas valide dans notre cas. En effet, il est trop restrictif de supposer que l'environnement n'évolue pas. Une alternative aurait été de considérer que l'environnement virtuel doit à chaque fois qu'un environnement change déclencher un évènement pour cet environnement. Et ceci pour tous les environnements utilisés dans le cadre de la simulation considérée. Il apparaît évident que cette solution n'est pas acceptable non plus. En effet, le trafic généré par tous ces évènements serait tout à fait préjudiciable en terme de performance et de complexité pour l'ensemble du système. Donc pour toutes ces raisons, cette approche n'a pas été développée dans le cadre de notre plate-forme multi-environnements.

Approche proposée

L'option prise dans la plate-forme multi-environnements est en fait une forme hybride des deux approches citées précédemment.

Il suffit de constater que chaque environnement possède une autonomie qui lui est propre, c'est à dire que chaque environnement peut éventuellement évoluer à une vitesse qui lui est propre par rapport aux autres. C'est pourquoi chaque environnement est doté de son propre minuteur qui va envoyer des impulsions à l'environnement auquel il est relié. De cette manière, on pourra prendre en compte le fait que les environnements peuvent avoir une évolution qui leur est propre et autonome. Pour ce faire, le minuteur de chaque environnement possède une description de ce que représente le temps pour l'environnement considéré.

Or ce premier ajout dans notre modèle implique un problème de fond relatif à l'autonomie de chaque environnement existant dans la simulation considérée : *comment faire en sorte que l'on ne perde pas la synchronisation éventuelle entre les environnements ?* En effet, comme les environnements évoluent (et par là même le temps) de manière autonome, on peut se trouver confronté au fait que les environnements ne soient plus synchrones. C'est à dire qu'à un instant t, des environnements peuvent se retrouver à des instants $t' \neq t$. Dans une telle situation, les informations qui seront reçues par le système conatif de l'agent vont donc être incohérentes et fausses. De ce fait, le système conatif, lors de son processus de raisonnement, sera amené à prendre des décisions qui ne seront pas en accord avec le temps réel et la réalité à cet instant dans l'environnement.

Pour pallier à ce problème, chaque environnement s'enregistrera (et donc utilisera une approche de type « temps dirigé par les évènements ») auprès de l'environnement virtuel grâce au lien d'intégrité qui les relie. Ce faisant il va définir des *points de synchronisation* qui vont être communs à tous les environnements. Ces points de synchronisation vont donc être des points de passage obligés que tous les environnements devront respecter. L'environnement pourra donc avoir une évolution autonome, mais celle-ci sera contrainte par ces points de synchronisation. Par exemple, si dans un environnement donné, le temps s'écoule très vite et qu'il a terminé avant le point de synchronisation suivant, cet environnement va se mettre en sommeil et attendre de recevoir une information provenant de l'environnement virtuel lui permettant de redémarrer son minuteur. Ces points de synchronisation permettront donc de garder une certaine stabilité et synchronisation entre les environnements. Néanmoins, le choix de ce point de synchronisation, qui est crucial pour le modèle, est à effectuer le plus judicieusement possible sous peine de retomber dans les problèmes décrits ci-dessus.

Concrètement, lors de la phase d'initialisation du modèle, chaque environnement va donc aller déclarer à l'environnement virtuel quel est son intervalle de temps entre lequel il va se modifier (que ce soit à cause de la phénoménologie ou à cause des données le décrivant) et se mettre en attente d'une réponse de l'environnement virtuel. Une fois que tous les environnements utilisés dans la simulation se sont déclarés auprès de l'environnement virtuel, celui-ci va calculer la meilleure solution en fonction de paramètres qui auront été définis par le concepteur. Lorsque cette phase de détermination de la meilleure solution est terminée, l'environnement virtuel enverra cette solution aux environnements toujours grâce aux liens d'intégrité. Une fois cette phase d'initialisation terminée, la simulation peut commencer.

4.6 Modifications dans le système conatif de l'agent

Les changements liés à l'ajout de l'environnement virtuel, l'instance dans l'environnement virtuelle et les lien virtuel bidirectionnel de dépendance entraîne implicitement des modifications au niveau de l'agent et de son système conatif.

Tout d'abord, reprenons le modèle du système conatif défini dans la figure 3.7 page 48. Celui-ci est composé deux entité principales : l'entité d'autonomie et la représentation de l'environnement que se construit le système conatif au fur et à mesure de son évolution. A ces deux entités principales, viennent se greffer le gestionnaire de priorités et le registre de perceptions retardées ($\mathcal{RPR}$), le registre de perception ($\mathcal{RP}$), le registre de perceptions ordonnées ($\mathcal{RPO}$), le registre de perceptions urgentes ($\mathcal{RPU}$), le registre de commandes ($\mathcal{RC}$), le registre de commandes ordonnées ($\mathcal{RCO}$) et le registre de commandes urgentes ($\mathcal{RCU}$). Dans les deux cas, les perceptions et les commandes (au travers de l'ordonnanceur de commandes) transitent le long du lien bidirectionnel de dépendance pour aller vers l'instance dans l'environnement.

Pour les perceptions et les actions

Comme il a été montré précédemment, il y a maintenant autant autant de liens de dépendance bidirectionnels que d'environnements. Les perceptions et les actions vont donc être découpées en autant de liens. Le système conatif doit donc être modifié afin de pouvoir prendre en compte cet ensemble de perceptions et d'actions. C'est pourquoi le système conatif va maintenant contenir un registre pour chaque type d'environnement auquel il va être rattaché. Concrètement, si le système conatif est représenté dans n environnements, $\mathcal{RP}$ va maintenant être défini de la manière suivante :

$$\mathcal{RP} = \mathcal{RP}_{type_1} + \mathcal{RP}_{type_2} + \cdots + \mathcal{RP}_{type_{n-1}} + \mathcal{RP}_{type_n}$$

Par conséquence, les registres qui découlent de $\mathcal{RP}$ vont être eux aussi définis de la même manière. Nous allons donc avoir :

$$\mathcal{RPO} = \mathcal{RPO}_{type_1} + \mathcal{RPO}_{type_2} + \cdots + \mathcal{RPO}_{type_{n-1}} + \mathcal{RPO}_{type_n}$$

et

$$\mathcal{RPU} = \mathcal{RPU}_{type_1} + \mathcal{RPU}_{type_2} + \cdots + \mathcal{RPU}_{type_{n-1}} + \mathcal{RPU}_{type_n}$$

Par analogie, $\mathcal{RC}$ va maintenant être défini de la manière suivante :

$$\mathcal{RC} = \mathcal{RC}_{type_1} + \mathcal{RC}_{type_2} + \cdots + \mathcal{RC}_{type_{n-1}} + \mathcal{RC}_{type_n}$$

ainsi que $\mathcal{RCO}$:

$$\mathcal{RCO} = \mathcal{RCO}_{type_1} + \mathcal{RCO}_{type_2} + \cdots + \mathcal{RCO}_{type_{n-1}} + \mathcal{RCO}_{type_n}$$

et $\mathcal{RCU}$:

$$\mathcal{RCU} = \mathcal{RCU}_{type_1} + \mathcal{RCU}_{type_2} + \cdots + \mathcal{RCU}_{type_{n-1}} + \mathcal{RCU}_{type_n}$$

De cette manière, à chaque pas de simulation, le système conatif va pouvoir avoir à sa disposition l'ensemble des informations qui auront été perçues dans les environnements auxquels ses instances dans l'environnement sont reliées. Ces définitions des différents registres est rendue possible par le fait que le médium qui permet aux informations qui vont transiter entre les instances dans l'environnement et le système conatif est le lien bidirectionnel de dépendance et qu'il y autant de liens de dépendance bidirectionnels que de types d'environnement. De ce fait, la connexion entre les différents $\mathcal{RP}_i$ et $\mathcal{RC}_i$ ($i \in \{1,\ldots,n\}$ pour n représentant le nombre des environnements) et les liens de dépendance bidirectionnels se fait naturellement en fonction du type des informations.

De plus, le gestionnaire de priorités va lui aussi posséder autant de liens vers $\mathcal{RP}$ que de types présents dans les liens bidirectionnels de dépendance. De cet fait, le gestionnaire de priorités va maintenant posséder un gestionnaire de priorités propre par type. A chaque

pas de simulation, les gestionnaires de priorités $\mathcal{GP}_i$ ($i \in \{1, \ldots, n\}$) vont donc, en accord avec les tendances propres du système conatif :

- Réordonner, trier et séparer les perceptions contenues dans $\mathcal{RP}$. Chaque perception d'un type précis va donc se retrouver, éventuellement, dans une nouvelle position en fonction des décisions du gestionnaire de priorités. Les perceptions à caractères non urgents vont se retrouver dans $\mathcal{RPO}_t$ (où t est le type) et les perceptions qui ont été déclarées comme urgentes vont donc se retrouver dans $\mathcal{RPU}_t$ (où t est le type).
- Réordonner, trier et séparer les actions prises par le système conatif. Chaque action d'un type précis va donc se retrouver, éventuellement, dans une nouvelle position en fonction des décisions du gestionnaire de priorités. Les actions à caractères non urgents vont se retrouver dans $\mathcal{RCO}_t$ (où t est le type) et les actions qui ont été déclarées comme urgentes vont donc se retrouver dans $\mathcal{RCU}_t$ (où t est le type).

Les figures 4.5 et 4.6 page suivante illustrent les modifications apportées au niveau du système conatif pour les perceptions et les actions liées à l'approche multi-environnements.

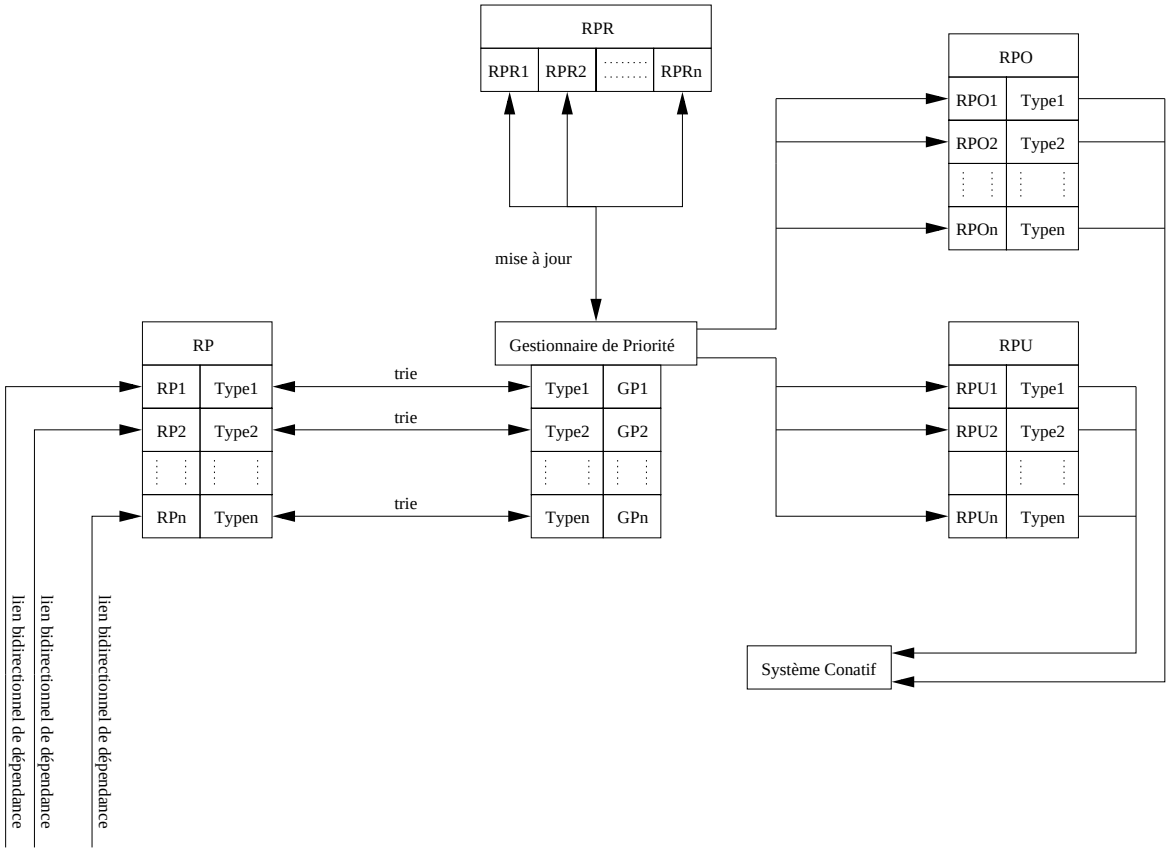


FIG. 4.5 – *Avant-dernière étape de la décomposition : modification du système conatif du côté des perceptions...*

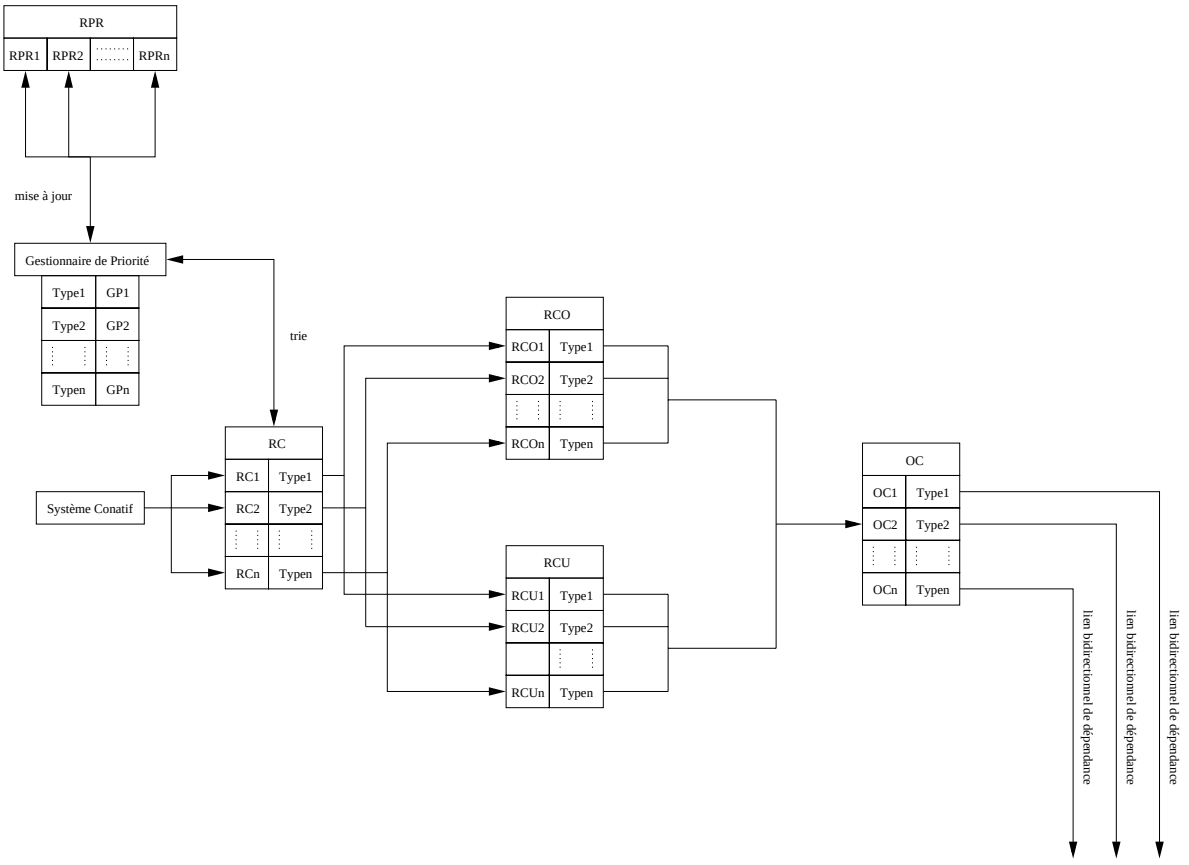


FIG. 4.6 – *...Et modification du système conatif du côté des actions*

Pour la représentation de l'environnement

Le représentation de l'environnement est l'autre composante principale et primordiale (voir section 3.2.1 page 46). Cette représentation est construite par l'entité d'autonomie au fur et à mesure de la simulation. Elle représente *la manière* dont l'agent se représente l'environnement. On se rend bien compte que cette représentation est intimement liée aux perceptions qui lui sont parvenues. De plus, ces perceptions à un temps t pouvant être le résultat d'actions déclenchées au niveau de l'environnement à un temps $t' < t$, l'entité d'autonomie peut tout à fait effectuer des déductions quant à la portée sur l'environnement d'une action donnée.

Or, comme nous l'avons montré précédemment, notre modèle utilise maintenant une multitude d'environnements dans lesquels on peut avoir un certain nombre de capteurs et d'effecteurs. Afin de pouvoir prendre en compte ces spécificités, deux solutions peuvent être envisagées :

Une représentation globale des environnements Cette approche revient à utiliser

le fait que l'environnement global est défini comme suit :

$$\text{environnement global} = \bigcup_{j=1}^{p} \left(\bigcup_{i=t_0}^{T} \text{environnement}_{(i,j)} \right)$$

Grâce à cette définition, il nous sera possible de créer au sein d'un programme informatique ce qu'est l'union temporelle et l'union des environnements typés. De ce fait, au fur et à mesure de la simulation, le système conatif mettra à jour les données grâce à ces unions. De cette manière, un environnement global pourra être géré est construit en fonction des perceptions et des actions décidées par l'agent. Or, si *a priori* il semble facile de travailler sur l'union temporelle (le temps progresse tout au long de la simulation de manière plus ou moins uniforme et on peut aisément définir une unité de temps commune ou un intervalle de temps commun), il apparaît rapidement qu'il peut être extrêmement difficile de travailler sur l'union des différents environnements. La principale raison vient de la diversité éventuelle des types que les environnements représentent. En effet, les types qui sont représentés par les environnements peuvent être de type totalement incompatible. Par exemple, dans le cas où on utilise un environnement de type « espace euclidien » et un autre environnement de type « graphe », il n'y a pas de méthodes qui puissent permettre de représenter l'union de ces deux types d'environnements. Comment représenter les arcs (représentant la liaison sociale de l'agent dans une société d'agents) d'un environnement de type « graphe » avec des valeurs liées à une positions dans un environnement de type « espace euclidien » par exemple ? Dans certain cas, il va pouvoir être possible de réaliser cette opération mais pas dans d'autres cas.
C'est pourquoi, cette représentation, bien qu'intéressante et prometteuse (notamment en terme d'occupation d'espace mémoire lors de simulations lourdes) n'a pas été retenue. Le principal inconvénient venant du fait que l'on ne peut pas gérer dans tous les cas la diversité des types des environnements et travailler sur les unions des ces environnements.

Une représentation décomposée des environnements Cette approche consiste, de la même manière que pour les liens bidirectionnels de dépendance, à décomposer la représentation de l'environnement construite au fur et à mesure de la simulation en autant de représentations de l'environnement que de types présents dans les environnements utilisés dans la simulation. De ce fait, à chaque type présent au niveau du lien bidirectionnel de dépendance, il va être créé une représentation de l'environnement. Il y aura donc une égalité totale entre les environnements utilisés et les représentations de l'environnement. Lorsque l'agent reçoit une perception correspondant à un type t donné, l'entité d'autonomie va automatique mettre à jour la représentation de l'environnement du type t correspondant.
Néanmoins, cette approche peut se heurter au problème énoncé en section 4.4 page 82. En effet, lorsque l'entité d'autonomie va avoir construit les représentations des environnements, il se peut que l'environnement virtuel effectue des modifications via les liens d'intégrité reliant l'environnement virtuel et les autres environnements. Dans ce cas, les représentations des environnements ne vont plus être en adéquation avec la réalité des environnements. Bien que les représentations des environnements

ne soient que « la vision » que se fait un agent de l'environnement, celui-ci doit pouvoir quand même être tenu au courant des modifications effectuées par l'environnement virtuel, afin de corriger ces modifications. Afin de contourner ce problème, à chaque fois que l'environnement virtuel va effectuer une modification via les liens d'intégrités, l'environnement (ou les environnements) concerné(s) par cette (ces) modification(s) va (vont) envoyer un message spécial au système conatif via le lien bidirectionnel de dépendance. Ce message aura comme principales caractéristiques :

- Il sera considéré comme urgent. De ce fait, ce message va passer avant tous les autres messages. Cela va permettre au système conatif de mettre à jour ses représentations de l'environnement avant que d'autres perceptions ne lui parviennent. Dans ce cas, lorsque les autres perceptions vont lui parvenir, il aura mis à jour sa représentation et lorsqu'il va entrer dans son processus de décision, il ne va pas s'appuyer sur des données erronées.
- Il va contenir les informations liées au changement concerné. Afin de pouvoir mettre à jour sa représentation de l'environnement, l'entité d'autonomie va se servir des informations contenues dans ce message. Or, l'entité d'autonomie ne construit que sa propre représentation de l'environnement, c'est pourquoi ces informations vont être soumises à une éventuelle altération liées au système conatif.

Cette approche est celle qui a été retenue dans notre modèle. Elle présente le principal avantage d'être extrêmement souple et de pouvoir s'adapter facilement à différents types d'environnements, à la différence de la première approche. Par contre, il est indéniable qu'elle alourdit considérablement le système car il y a autant de représentations des environnements que d'environnements utilisés dans le système multi-agents et elle augmente fortement le trafic entre les environnements, les instances dans l'environnement, les liens bidirectionnels de dépendance et le système conatif. Néanmoins c'est la seule approche qui fournisse la souplesse nécessaire à notre modèle multi-environnements.

La figure 4.7 page ci-contre illustre la dernière étape de la décomposition.

4.7 Conclusion

Dans ce chapitre, nous avons montré comment, à partir du modèle mono-environnemental, on peut construire un modèle d'agent permettant à celui-ci de prendre en compte des environnements multiples de types différents. Cette approche s'appuie sur la décomposition des environnements selon leurs types intrinsèques. Au delà de cette séparation des environnements, le modèle d'agent a été enrichi afin de pouvoir prendre en compte ces aspects. Enfin, nous avons présenté des mécanismes qui permettent de maintenir l'intégrité des données dans les environnements.

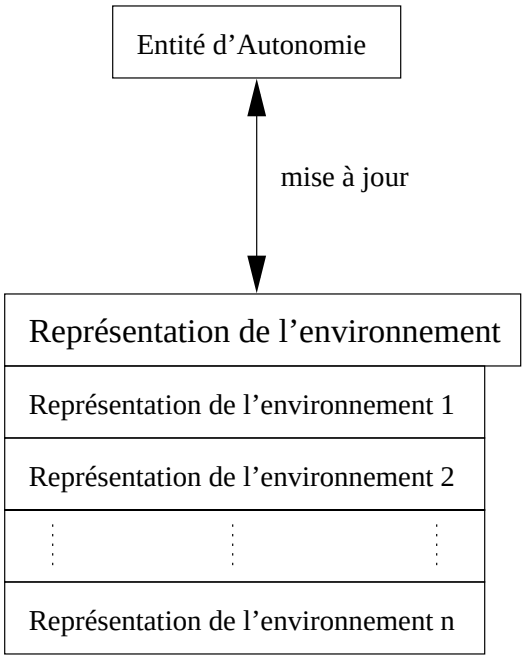


FIG. 4.7 – *Dernière étape de la décomposition : modification du système conatif*

Au delà des aspects purement environnementaux, cette approche permet, en outre, de simplifier le travail du concepteur. En effet, elle permet à celui-ci de se focaliser purement et simplement sur les aspects typologiques des environnements. Cette décomposition permet d'isoler de manière claire les mécanismes mis en jeu pour un type d'environnement donné. De plus, en séparant tout ce qui concerne un type d'environnement dans des entités séparées, on peut identifier plus facilement les interconnexions qui peuvent exister entre les environnements de différents types. Par exemple, lorsqu'un agent modifie une valeur dans un environnement donné et que cette modification entraîne, par un effet de cascade, d'autres modifications dans d'autres environnement, on peut aisément imaginer que la modification initiale a une conséquence sur les autres environnements. Et par là même, on isole quel environnement et quelle action a une influence sur les autres. De cette manière, on peut donc modéliser beaucoup plus facilement l'influence d'un environnement sur un autre.

Néanmoins, malgré les apports indéniables de cette approche, celle-ci souffre d'un défaut qui est intimement lié à sa définition : le complexité du modèle se trouve considérablement augmentée. Nous avons argumenté dans ce chapitre sur les choix qui ont été faits en terme de modélisation. Bien souvent les choix les plus simples ne permettaient pas de prendre en compte tous les cas de figure qui pouvaient se présenter. C'est pourquoi, l'augmentation de la complexité a, malheureusement bien souvent, été le seul moyen de pouvoir gérer cet apport des environnements multiples dans le modèle d'agent.

Nous avons donc justifié dans ce chapitre du passage d'un modèle mono-environnemental à un modèle multi-environnemental. Nous allons voir dans le chapitre suivant

comment ce modèle à base d'environnements multiples a été implémenté pour arriver aux aspects applicatifs.

Chapitre 5

La plate-forme multi-environnements

Sommaire

5.1 Introduction

Le formalisme que nous allons employer pour décrire notre plate-forme multi-environnements est celui du paradigme objet [RBEL91]. Ce paradigme s'appuie sur la description de prototypes appelés *classes*. Ces classes sont le modèle d'entités dans un monde réel ou artificiel. La structure fondamentale d'un système orienté objet est l'*objet* qui est une instance d'une classe. Cette structure est la combinaison de deux type précis : *statique* et *dynamique*. Les propriétés statiques décrivent l'état de l'objet et sont appelées les *variables d'instances*. Les propriétés dynamiques décrivent les comportements de l'objet et sont appelées les *méthodes*. Donc, les objets qui sont des instances d'une même classe vont avoir des valeurs pour les états qui leurs sont propres, mais par contre, leurs comportements

[RBEL91] J. Rumbaugh, M. Blaha, F. Eddy et W. Lorensen. – *Object-Oriented Modeling and Design*. – London, England, Prentice All International, 1991.

vont être les mêmes entre eux. La partie dynamique des objets s'appuie sur l'envoi et la réception de messages. En effet, pour demander à un objet d'exécuter une action, il suffit de lui envoyer un message qui contient l'information associée. De ce fait, les objets sont capables de communiquer entre eux en utilisant de mécanisme d'envoi et de réception de message.

A la lecture de cette définition pour un objet, on peut voir comment les agents peuvent être très aisément modélisés en utilisant ce paradigme. En effet, un agent va posséder un ensemble de variables qui vont décrire ses propres ressources et les méthodes vont donc pouvoir décrire, éventuellement, les tendances qui le décrivent et les messages qu'il va être capable de recevoir afin de permettre l'échange d'informations entre les agents. Néanmoins, il convient d'ajouter au paradigme objet les capacités d'autonomie et prise de décision. Moyennant cet ajout, on peut donc très facilement modéliser des agents à l'aide des objets.

Dans ce chapitre, nous allons tout d'abord décrire le modèle objet de notre plate-forme multi-environnements. Ensuite, nous allons décrire la dynamique liée à ce système et enfin nous allons présenter brièvement l'implémentation de la plate-forme mettant en œuvre les environnements multiples en JAVA [NP97, Fla97].

5.2 Le modèle objet

Comme nous l'avons présenté en section 4 page 75, notre plate-forme est décomposée en deux composantes principales : l'agent et l'environnement. Nous allons donc présenter le modèle objet de ces deux composantes.

5.2.1 Pour l'agent

L'agent est composé principalement des attributs suivants :

- L'attribut `id` qui va être l'identifiant unique attribué à un agent dans la société d'agents. Ceci afin de pouvoir retrouver facilement un agent donné.
- L'attribut `ConativeSystem` qui représente le système conatif de l'agent qui va avoir la responsabilité décisionnelle.
- L'attribut `BDLinks` qui représente l'ensemble des liens bidirectionnels de dépendance qui arrivent sur l'agent. Cet attribut est en fait une table de hachage dont la clé de

[NP97] Patrick NIEMEYER et Joshua PECK. – *Exploring Java (2^{nd} Edition).* – O'Reilly, 1997, *The Java Series.*

[Fla97] David FLANAGAN. – *Java in a nutshell (version 1.1).* – O'Reilly, 1997, 2nd édition.

hachage est le type relatif au lien.
- L'attribut `RP` qui représente $\mathcal{RP}$. Cet attribut est en fait une table de hachage dont la clé de hachage est le type relatif au lien auquel il est relié.
- L'attribut `ORP` qui représente $\mathcal{RPO}$. Cet attribut est en fait une table de hachage dont la clé de hachage est le type relatif au $\mathcal{RP}$ auquel il est relié.
- L'attribut `URP` qui représente $\mathcal{RPU}$. Cet attribut est en fait une table de hachage dont la clé de hachage est le type relatif au $\mathcal{RP}$ auquel il est relié.
- L'attribut `PriorityManager`. C'est le gestionnaire de priorité de l'agent. Cet attribut est en fait une table de hachage dont la clé de hachage est le type relatif au $\mathcal{RP}$ auquel il est relié.
- L'attribut `RC` qui représente $\mathcal{RC}$. Cet attribut est en fait une table de hachage dont la clé de hachage est le type relatif au lien auquel il est relié.
- L'attribut `ORC` qui représente $\mathcal{RCO}$. Cet attribut est en fait une table de hachage dont la clé de hachage est le type relatif au $\mathcal{RC}$ auquel il est relié.
- L'attribut `URC` qui représente $\mathcal{RCU}$. Cet attribut est en fait une table de hachage dont la clé de hachage est le type relatif au $\mathcal{RC}$ auquel il est relié.
- L'attribut `RPR` qui va contenir les perceptions retardées par le gestionnaire de priorités. Cet attribut est en fait une table de hachage dont la clé de hachage est le type relatif au gestionnaire de priorités auquel il est relié.
- L'attribut `Society` qui représente le lien que possède l'agent vers la société d'agent à laquelle il appartient.

Du point de vue de la dynamique liée à l'objet, l'agent contient les méthodes suivantes (outre les méthodes classiques d'accès) :

- La méthode `tickFirstStep` qui va permettre à l'agent de s'initialiser. Cette méthode va donc être appelée une seule fois en début de simulation. Concrètement, la méthode `tickFirstStep` appelle la méthode `firstStep` contenue dans la classe `ConativeSystem`. Au niveau de la plate-forme générique, cette dernière méthode est abstraite car on ne peut pas définir quelles seront les opérations initiales à effectuer dans le cadre de l'application concernée.
- La méthode `tick` qui va permettre à l'agent de passer à un nouveau pas de temps. Cette méthode va successivement exécuter les opérations suivantes :

  1. Activer le gestionnaire de priorités pour mettre à jour les perceptions qui ont été reçues de l'environnement par le biais des différents capteurs.
  2. Activer le processus de raisonnement du système conatif en fonction des dernières perceptions reçues.
  3. Collecter les commandes qui ont été décidées par le système conatif.
  4. Activer le gestionnaire de priorités pour mettre à jour les commandes à envoyer.
  5. Envoyer les commandes ordonnées le long des liens bidirectionnels de dépendance.

- La méthode `receives` qui va être activée par les liens bidirectionnels de dépendance lorsque ceux-ci ont des informations à stocker au niveau de l'agent.
- La méthode `sends` qui va être activée par lé méthode `tick` pour envoyer les commandes le long des liens bidirectionnels de dépendance.

5.2.2 Le système conatif

L'autre classe prépondérante dans le cadre de la modélisation de l'agent est le système conatif. En effet, c'est dans celle-ci que tout le processus de prise de décision de l'agent. Comme présenté dans la section 5.2.1 page 100, cette classe se nomme `ConativeSystem`. Cette classe est composée des attributs suivants :

- L'attribut `Agent` qui permet de faire lien entre le système conatif et l'agent auquel il est rattaché. De cette manière, on peut accéder aux attributs de la classe agent par le biais de des méthodes d'accès.
- L'attribut `InformationMemories` qui va permettre à l'agent de stocker les perceptions qu'il a reçu tout au long de sa simulation. La taille de stockage sera plus ou moins grande en fonction des desiderata du concepteur du système. En effet, si l'agent n'a pas besoin de posséder une grande « mémoire » (*e.g* l'agent est plus réactif), la taille de cet attribut sera moindre ; par contre, si l'agent a besoin dans son processus de raisonnement d'un grand nombre d'informations (*e.g* l'agent est plus cognitif), la taille de la mémoire sera plus grande. Cet attribut est une table de hachage dont la clé est le type de la perception stockée.
- L'attribut `CommandsMemories`. De la même manière que pour l'attribut `InformationMemories`, sa taille est plus ou moins grande (et pas forcément la même). Cet attribut est une table de hachage dont la clé est le type de la perception stockée.

Du point de vue de la dynamique liée à l'objet, le système conatif d'un agent contient les méthodes suivantes :

- La méthode `process` qui est en fait le processus de raisonnement du système conatif. Selon la manière dont on va programmer cette méthode, on va pouvoir faire varier l'agent de purement réactif à totalement cognitif. A ce niveau d'abstraction, le comportement de cette méthode ne peut être défini car il est dépendant de l'application considérée. C'est pourquoi au niveau de la plate-forme générique, cette méthode est définie comme abstraite.
- La méthode `tickFirstStep`. Comme nous l'avons déjà présenté, cette méthode permet de réaliser les diverses initialisations au début de la simulation. A ce niveau d'abstraction, les opérations d'initialisations liées à cette méthode ne peuvent être définis car ils sont dépendants de l'application considérée. C'est pourquoi au niveau de la plate-forme générique, cette méthode est définie comme abstraite.

– La méthode `receives`. Cette méthode va prendre les perceptions dans les registres $\mathcal{RCO}$ et $\mathcal{RCU}$. Elle permet donc au système conatif de pouvoir accéder aux perceptions effectuées par les capteurs au niveau de l'environnement.
– La méthode `sends`. Cette méthode va envoyer toutes les commandes qui ont été prises lors du processus de prise de décision (*e.g*lors de l'exécution de la méthode `process`) à l'agent qui va ensuite les traiter et les laisser au soin du gestionnaire de priorités.

Enfin, afin de résumer notre modèle d'agent, la figure 5.1 est une représentation simplifiée de celui-ci.

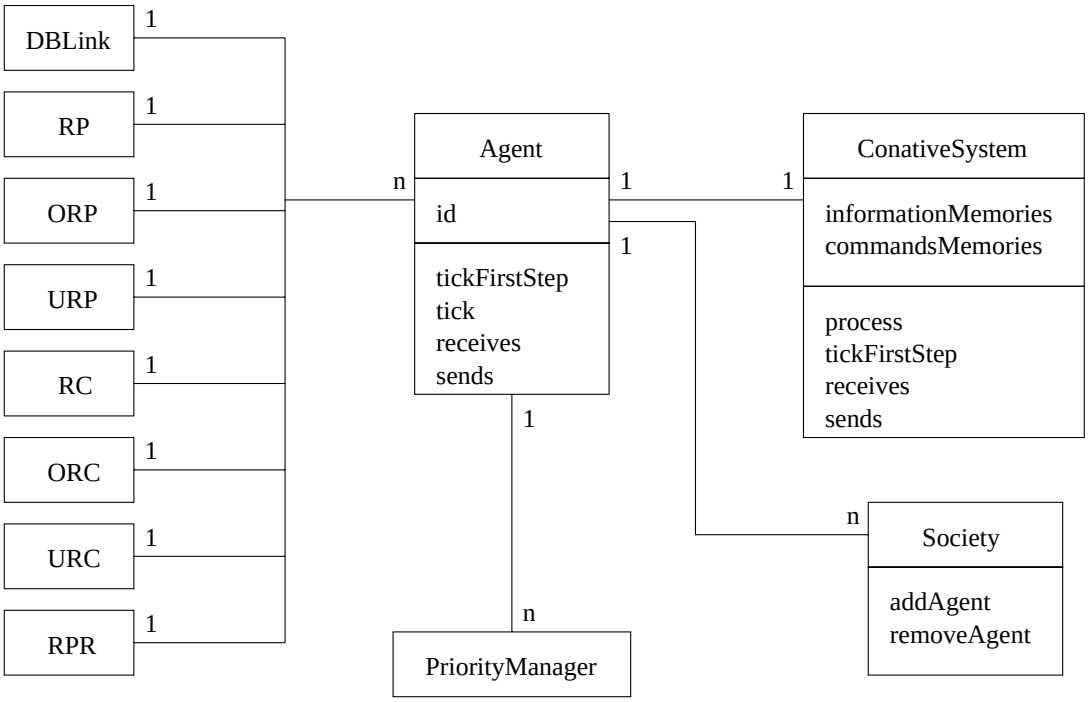


FIG. 5.1 – *Une représentation du modèle d'agent*

5.2.3 Pour les classes liées à l'environnement

Le modèle objet de l'environnement est articulé autour de cinq classes principales : l'environnement virtuel, l'environnement, le minuteur et l'accès aux données.

L'environnement virtuel

La classe liée à l'environnement virtuel s'appelle `VirtualEnvironment`. C'est la racine d'héritage pour tous les autres environnements. L'environnement virtuel ne possède pas

de type car il est unique. Elle est constituées des attributs suivants :

- L'attribut `Timer`. C'est grâce à cet attribut, qui est une association vers une autre classe, que l'environnement virtuel va pouvoir gérer le temps.
- L'attribut `IntegrityLinks`. Cet attribut regroupe l'ensemble des liens d'intégrités qui relient les environnements et l'environnement virtuel. De la même manière que pour les liens bidirectionnels de dépendance, cet attribut est une table de hachage dont la clé de hachage est le type du lien d'intégrité.
- L'attribut `Effectors`. Cet attribut regroupe les effecteurs qui sont applicables sur l'environnement virtuel. Comme ci-dessus cet attribut est une table de hachage dont la clé de hachage est le type de l'effecteur.
- L'attribut `Captors`. Cet attribut regroupe les capteurs qui sont applicables sur l'environnement virtuel. Comme ci-dessus cet attribut est une table de hachage dont la clé de hachage est le type de capteur.
- L'attribut `Commands`. Cet attribut regroupe les commandes qui doivent être exécutée à un instant t donné sur l'environnement virtuel. Cet exécution est de la responsabilité de l'environnement virtuel, c'est celui-ci qui décidé si oui ou on une commande donnée peut être appliquée sur l'environnement virtuel par le biais des effecteurs.

Du point de vue de la dynamique liée à l'objet, l'environnement virtuel contient les méthodes suivantes :

- La méthode `update`. Cette méthode est exécutée à chaque pas de temps. C'est elle qui va réaliser séquentiellement les opérations suivantes :

  1. Exécution des commandes qui sont stockées dans l'attribut `Commands`.
  2. Mise à jour des environnements grâce aux liens d'intégrités.
  3. Récupérations des valeurs souhaités par le système conatif grâce aux capteurs

- La méthode `tick` qui est appelée par le séquenceur à chaque pas de simulation.
- La méthode `tickFirstStep` qui est appelée lors de l'initialisation du système. Son rôle est exactement le même que la méthode `tickFirstStep` de l'agent.
- La méthode `tickChangeMap`. Cette méthode est appelée par la méthode `tick` lorsque l'environnement virtuel a besoin d'être mis à jour. Il se peut que l'on change de pas de simulation sans que l'environnement virtuel change de pas de simulation sans avoir à être change et dans ce cas, la méthode `tick` n'exécute pas cette méthode ; par contre, dans le cas où l'environnement virtuel va être modifié, cette méthode va être automatiquement appelée.
- La méthode `changeMap`. Cette méthode va être appelée par la méthode ci-dessus lorsque l'environnement virtuel va être modifié. Cette méthode va être complètement dépendante du type d'environnement virtuel qui est considéré. En effet, les opérations à effectuer ne seront pas les mêmes si l'environnement virtuel est de type

espace à deux dimensions ou de type graphe. C'est pourquoi à ce niveau d'abstraction, cette méthode est déclarée comme abstraite car on ne peut pas définir les opérations nécessaires au changement d'environnement virtuel.

L'environnement

L'environnement est une sous classe de l'environnement virtuel. Il va donc posséder par défaut toutes les caractéristiques de l'environnement virtuel. De plus, il possède un attribut `type`. Cet attribut décrit le type de l'environnement. Celui-ci doit donc correspondre exactement avec les types des liens bidirectionnels et des instances dans l'environnement.

Au niveau des méthodes, cette classe ne possède pas méthode particulière outre les méthodes d'accès et de modifications liées au type de l'environnement.

Le gestionnaire de temps

La classe liée au gestionnaire du temps s'appelle `Timer`. Comme il a été présenté ci-dessus, l'environnement virtuel et les environnements possèdent un gestionnaire de temps. De cette manière, nous pouvons prendre en compte les spécificités de notre modèle vis à vis du temps. Cette classe possède cinq attributs :

- L'attribut `VirtualEnvironment`. Cet attribut permet de faire le lien entre l'environnement et les gestionnaires de temps.
- L'attribut `gap`. Cet attribut représente l'intervalle de temps entre deux pas de simulation. Par exemple si le `gap` représente des heures et que le pas de simulation est un jour, on va attendre 24 pas de temps pour changer de pas de simulation. Cet attribut va nous permettre de gérer des pas de temps plus ou moins fin selon les environnements dans lequel on se situe.
- L'attribut `current`. Cet attribut représente à quel pas de simulation le système se situe à un instant donné.
- L'attribut `start`. Cet attribut représente le temps auquel commence la simulation.
- L'attribut `end`. Cet attribut représente le temps de la fin de la simulation.

Du point de vue de la dynamique liée à l'objet, le gestionnaire de temps contient les méthodes suivantes :

- La méthode `tickFirstStep`. Le comportement de cette méthode est identique aux autres méthodes `tickFirstStep`. Elle va donc permettre d'initialiser correctement le gestionnaire de temps et de pouvoir lancer la simulation. A la différence des autres

classes qui possèdent cette méthode, celle-ci est déclarée comme abstraite, car à ce niveau d'abstraction, on ne peut pas définir quel type de temps va être utilisé lors de la simulation.

- La méthode `tick`. Cette méthode indique à l'environnement que l'on vient de passer à un nouveau pas de simulation. A la différence des autres classes qui possèdent cette méthode, celle-ci est déclarée comme abstraite, car à ce niveau d'abstraction, on ne peut pas définir quel type de temps va être utilisé lors de la simulation.
- La méthode `changeDate`. Cette méthode calcule la nouvelle date de la simulation. En effet, comme nous utilisons des environnements qui peuvent avoir un temps différent, il faut pouvoir effectuer des conversions entre ces temps. Cette méthode va donc permettre de réaliser cette tâche. Cette méthode est déclarée comme abstraite, car à ce niveau d'abstraction, on ne peut pas définir quel type de temps va être utilisé lors de la simulation.
- La méthode `setDate`. Analogue à la méthode ci-dessus, elle permet de modifier l'attribut `current` afin de mettre à jour le pas de temps de la simulation. Cette méthode est déclarée comme abstraite, car à ce niveau d'abstraction, on ne peut pas définir quel type de temps va être utilisé lors de la simulation.
- La méthode `getDate`. Analogue à la méthode ci-dessus, elle permet de récupérer la valeur de l'attribut `current` afin de connaître le pas de temps de la simulation. Cette méthode est déclarée comme abstraite, car à ce niveau d'abstraction, on ne peut pas définir quel type de temps va être utilisé lors de la simulation.

L'accès aux données

La partie d'accès aux données comprend deux classes : la classe `DatabaseConnectivity` et la classe `ImagedDataBaseConnectivity`. Nous allons maintenant détailler ces deux classes.

La connexion à une base de données Cette classe permet à un environnement d'aller chercher des informations dans une base de données. Cette classe appelée `DatabaseConnectivity` contient sept attributs :

- L'attribut `VirtualEnvironnement`. Cet attribut permet de faire le lien entre l'environnement virtuel (ou les environnements) et cette classe d'accès aux données.
- L'attribut `TimeZone`. Cet attribut permet de définir dans quel zone GMT on se situe. Ceci est lié aux spécificités de notre modèle relationnel lié à la base MySQL.
- L'attribut `HostName`. Cet attribut permet de définir sur quelle machine se trouve la base de données MySQL.
- L'attribut `HostDomain`. Cet attribut permet de définir sur quel nom de domaine se trouve la base de données MySQL.

- L'attribut `Base`. Cet attribut spécifie le nom de la base que l'on va utiliser dans MySQL.
- L'attribut `User`. Cet attribut spécifie le nom de l'utilisateur qui se connecte à la base `Base` dans MySQL.
- L'attribut `Passwd`. Cet attribut spécifie le mot de passe de l'utilisateur `User` qui se connecte à la base `Base` dans MySQL.

Du point de vue de la dynamique liée à l'objet, la classe `DatabaseConnectivity` ne contient que des méthodes d'accès à ces différents attributs.

La connexion à une base de données d'images Cette classe permet à l'environnement de se connecter à la base de données d'images satellites utilisées dans MUFFINS (voir section 6.3.4 page 133). Elle est une sous classe de la classe `DatabaseConnectivity` et elle hérite donc automatiquement des attributs et méthode cette dernière. Elle est composée de six attributs :

- L'attribut `DataBase`. Cet attribut spécifie l'objet base de donnée qui nous permet de réaliser les requêtes pour extraire des informations de la base de données.
- L'attribut `Type`. Cet attribut spécifie le type sur lequel on va réaliser des requêtes. Ce type doit donc être exactement le me que celui de l'environnement auquel cette classe est reliée.
- L'attribut `ImageTable`. Cet attribut spécifie la table dans laquelle se situent les images. Cet attribut est intimement lié avec le type.
- L'attribut `CurrentDate`. Cet attribut représente le temps (dans le format de la base de données) courant dans lequel se trouve la simulation.
- L'attribut `Images`. Cet attribut représente l'ensemble des images qui sont disponibles pour une date donnée. Dans la plupart des cas, il n'y a qu'une seule image pour une date donnée, mais dans certains (plus rares), il est possible d'avoir plusieurs images.
- L'attribut `Image`. Cet attribut représente l'image courante sur laquelle on va effectuer des requêtes afin de pouvoir récupérer des informations.

Du point de vue de la dynamique liée à l'objet, la classe `ImagedDataBaseConnectivity` contient les méthodes suivantes :

- La méthode `getValueAt`. Cette méthode permet d'aller chercher une valeur dans une image précise. A ce niveau d'abstraction, on ne peut définir quels sont les opérations à réaliser pour obtenir cette valeur, c'est pourquoi elle est définie comme une méthode abstraite.
- La méthode `changeDate`. Cette méthode est invoquée par le gestionnaire de temps via l'environnement. Lorsque la simulation change de pas de temps, il faut changer les informations de cette classe afin quelle se situe bien sur la bonne image dans la base de données. C'est le rôle de cette méthode.

– La méthode `constructConnectionString`. Cette méthode permet de construire la chaîne de caractère qui permet de se connecter à la base de données MySQL. En outre, elle est capable de résoudre les adresses IP sous la forme `nom.de.domaine.pays`.

Enfin, afin de résumer notre modèle d'agent, la figure 5.2 est une représentation simplifiée de celui-ci.

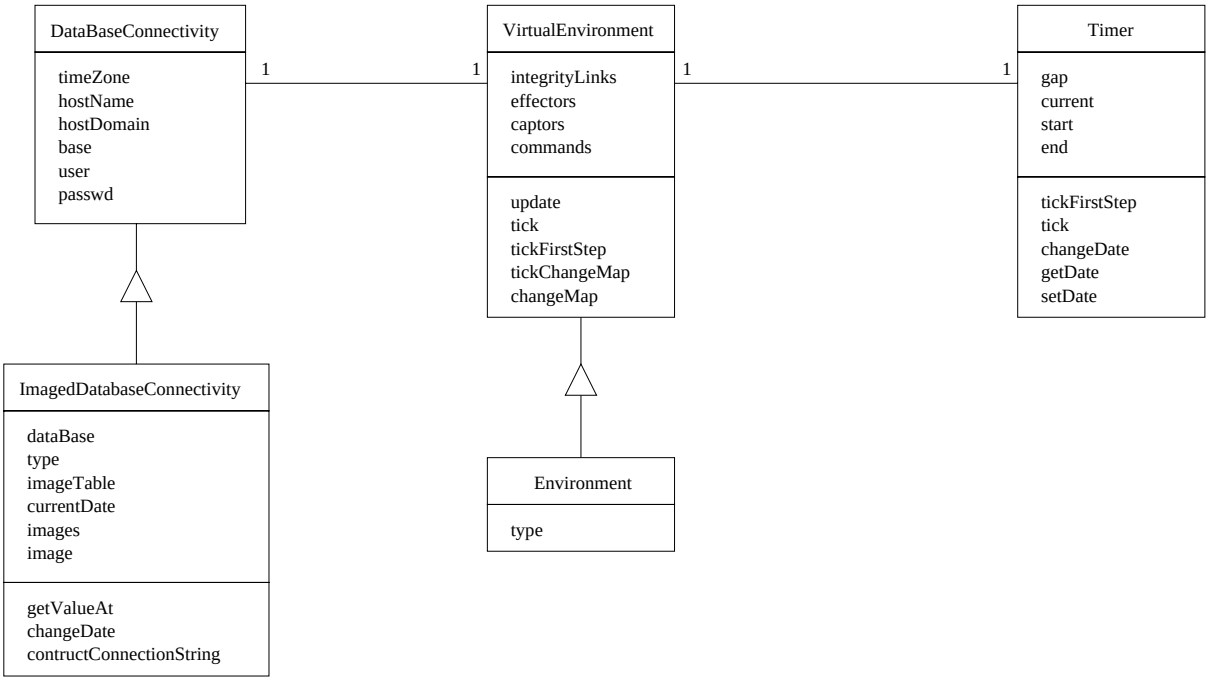


FIG. 5.2 – *Une représentation du modèle d'environnement*

5.3 La dynamique du modèle

Dans cette section nous allons nous attacher à montrer comment se comporte notre modèle multi-environnements. Pour ce faire, nous allons étudier plus précisément les dynamiques des deux parties fondamentales de notre système : l'agent et les environnements.

5.3.1 La dynamique de l'agent

La dynamique liée à l'agent est en fait assez simple, elle peut être présentée comme suit :

1. L'agent est ajouté dans la société d'agents.

2. Lors de son initialisation, l'agent crée le système conatif associé et celui-ci pointe automatiquement sur le comportement par défaut qu'on lui aura passé en paramètre. Ce comportement pourra être éventuellement modifié lors de l'exécution de la simulation.
3. L'agent crée un lien virtuel de dépendance bidirectionnel et crée l'instance physique virtuelle. Ensuite celui-ci connecte ces deux entités. Enfin, il crée les capteurs et les effecteurs associés à l'instance physique virtuelle.
4. L'agent crée autant de liens bidirectionnels de dépendance que nécessaire. Le nombre de liens et leurs types est passé en paramètre lors de la création de l'agent. De même que précédemment, l'agent crée dans la foulée les instances physiques qui sont rattachées à lien donné et les connecte aux liens bidirectionnels de dépendance. Enfin, il crée les capteurs et les effecteurs associés à l'instance physique.
5. L'agent exécute la méthode `tickFirstStep` qui lui permet d'initialiser les attributs nécessaires à l'application (*e.g* aller chercher des valeurs dans une base de données, dans un fichier, ...).
6. Une fois ces étapes d'initialisation effectuées, l'agent rentre dans une boucle qui ne prendra fin qu'à la fin de la simulation. Dans cette boucle, il effectue les opérations suivantes :

   - Il récupère (s'il y en a) les perceptions dans les registres de perception.
   - Il les transmet au système conatif qui va les stocker afin de les prendre en compte.
   - Il récupère les commandes qui ont été prises par le système conatif.
   - Il dépose ces commandes dans les liens bidirectionnels de dépendance.
   - Il met à jour (éventuellement) les priorités dans les tables de priorités du gestionnaire de priorité.
   - Enfin, il met à jour ses représentations des environnements.

Enfin, on peut noter que les perceptions contenant en leur sein le pas de temps de la simulation, l'agent est automatiquement tenu au courant de l'évolution du temps lors de la simulation.

5.3.2 La dynamique des environnements

La séquence d'initialisation des environnements se fait en deux étapes :

- Tout d'abord, les environnements réels sont crées. Cette création est réalisée en fonction d'une liste de paramètres qui est passée au processus d'initialisation. Les capteurs et les effecteurs crées par l'agent sont rattachés aux environnements en fonction de leurs types. Enfin, les gestionnaires de temps sont initialisés en fonction de leurs propres types.

– Enfin, l'environnement virtuel est crée. Les capteurs et les effecteurs liés à l'instance physique virtuelle sont reliés à l'environnement virtuel. Les liens d'intégrités sont crées et ils relient les environnements réels à l'environnement virtuel. Enfin, le gestionnaire de temps est initialisé.

Lors de l'exécution de la simulation, les environnements et l'environnement virtuel réalisent, d'une manière générale, les mêmes opérations :

– Collecte des informations adéquates afin d'alimenter les capteurs de l'environnement.
– Exécution des commandes se trouvant dans les effecteurs.
– Application de la phénoménologie sur l'environnement.

Néanmoins, on peut isoler des opérations spécifiques pour les environnements et pour l'environnement virtuel :

Pour les environnements Comme nous l'avons présentés en 4.4 page 82, les environnements sont capables de détecter lorsqu'une commande peut avoir un effet sur un autre environnement. Dans ce cas, les environnements vont envoyer l'information correspondante à l'environnement virtuel. De la même manière, l'environnement peut recevoir une notification de mise à jour provenant de l'environnement virtuel. Dans ce cas, l'environnement va prendre en compte cette notification et exécuter la modification correspondante.

Pour l'environnement virtuel L'environnement virtuel est capable

– de prendre en compte des notifications de mise à jour provenant des environnements. Dans ce cas, l'environnement virtuel va prendre en compte ce message et le transmettre aux environnements concernés par cette modification.
– de s'apercevoir qu'une modification qui a été effectuée et qui doit informer les environnements concernés. Dans ce cas, il va envoyer les informations adéquates à tous les environnements concernés.

5.4 Son implémentation

L'implémentation de ce modèle à été réalisée en Java [Fla97, Jav]. Elle est articulée autour de deux packages distincts : le package `multienv.mas.agent` et le package `multienv.mas.environment`. Ces packages contiennent toutes les classes présentées dans la section 5.2 page 100.

[Jav] The Source for Java^{TM} Technology. – http://www.javasoft.com/.

Néanmoins, l'implémentation dans le langage Java nous a conduit à utiliser ses spécificités dans les classes suivantes :

- La classe `multienv.mas.agent.Agent` implémente l'interface `java.lang.Runnable`. De cette manière, l'agent va donc se comporter comme un processus léger autonome.
- De la même manière, la classe `multienv.mas.environment.VirtuelEnvironment` implémente elle aussi l'interface `java.lang.Runnable` et donc l'environnement virtuel va se comporter comme un processus léger. Enfin, comme la classe `multienv.mas.environment.Environment` hérite de la classe `multienv.mas.environment.VirtualEnvironment`, l'environnement va lui aussi se comporter comme un processus léger.

Enfin, toutes les classes développées dans ces deux packages implémentent l'interface `java.io.Serializable`. De cette manière, on peut effectuer des sauvegardes des objets instanciés dans un seul fichier. Cette fonctionnalité nous permet, entre autre, de pouvoir sauvegarder des simulations et de pouvoir les relancer ultérieurement. Afin de pouvoir réaliser ses propres sauvegardes, les classes surchargent les méthodes `vriteObject(java.io.ObjectOutputStream out)` et `readObject(java.io.ObjectInputStream in)`

5.5 Conclusion

Dans cette section, nous avons présenté comment notre modèle multi-environnemental a été mis en œuvre grâce au paradigme objet. Ce modèle objet a été implémenté dans le langage JAVA. Cette plate-forme est un micro noyau générique qui permet de construire facilement des applications spécifiques. A ce jour, ce micro noyau comporte environ 5000 lignes de code. Enfin, la plate-forme multi-environnements est fournie avec un ensemble de primitives graphiques qui permettent de visualiser l'évolution d'une simulation.

Nous avons choisi volontairement de développer un micro noyau simple qui fournisse les primitives minimales nécessaires. Ce choix s'explique par le fait que le modèle multi-environnemental est déjà lui-même très complexe et nous ne voulions pas rajouter un niveau supplémentaire de complexité dans notre micro noyau. Ce choix se fait, bien évidement au détriment de la facilité de construction des applications. Néanmoins, nous pensons que les services fournis par ce micro noyau permettent de construire rapidement des applications simples.

Chapitre 6

Un exemple : le déplacement des espadons

Sommaire

6.1 Avant-propos : la pêche palangrière pélagique ciblant l'espadon

6.1.1 Les espèces pélagiques

La pêche pélagique s'effectue dans l'ensemble (surface et profondeur) des eaux marines non directement liées au fond ; elle s'exerce sur des poissons bons nageurs, se déplaçant habituellement en bancs et capables d'effectuer des migrations de grande amplitude. Elle

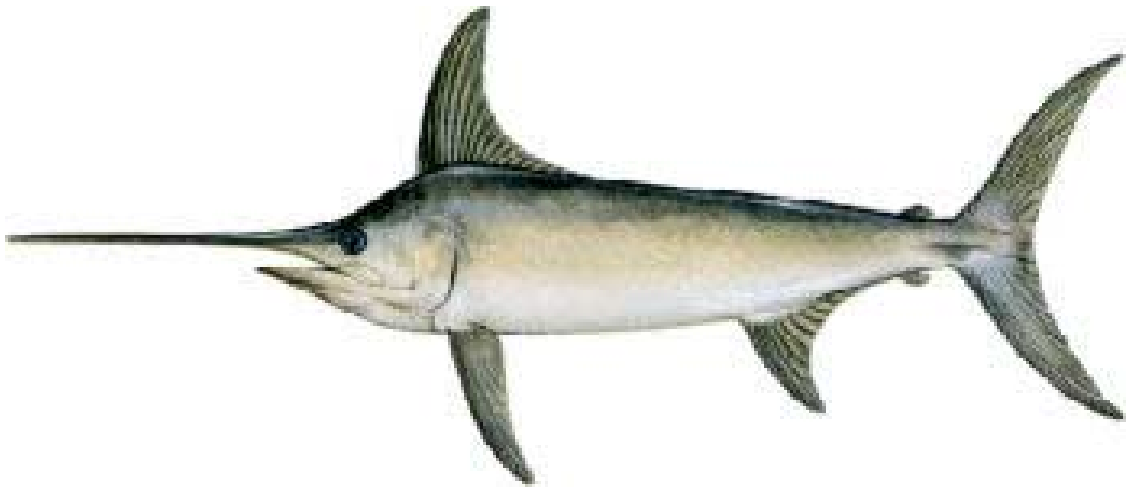


FIG. 6.1 – *Xiphias Gladius*

s'oppose à la pêche démersale, qui concerne surtout les espèces vivant sur le fond ou à proximité immédiate de celui-ci (domaine benthique) et, de ce fait, à nage plutôt lente et à déplacements relativement limités.

Par rapport au relief sous-marin, on distingue le secteur néritique, du littoral à la bordure du plateau continental, et le secteur océanique, qui regroupe les eaux du large, situées au-dessus du talus continental et au-delà du talus continental. Selon la profondeur, le domaine pélagique est réparti en épipélagique ($0-100$ m), mésopélagique ($100-500$ m), infrapélagique ($500-1000$ m), bathypélagique et abyssopélagique (plus de 1000 m). Jusqu'à présent, la pêche n'intéresse pratiquement que la zone épipélagique, les autres zones ne faisant l'objet que d'une exploitation limitée ou ponctuelle.

La zone épipélagique est celle qui est la plus influencée par les mouvements de la mer (vagues, marées, courants de marée et courants généraux) ainsi que par les changements climatiques. Si l'on considère les mouvements verticaux de la mer, les zones de remontée d'eaux de profondeur, plus froides mais aussi plus riches en sels nutritifs, constituent des secteurs très favorables au développement des espèces pélagiques. Par ailleurs, les zones de contraste, horizontalement et verticalement, coïncident en général avec des modifications de la répartition ou du comportement des animaux marins.

Les espèces pélagiques sont essentiellement constituées par des poissons, des mollusques céphalopodes, des crustacés nageurs et des mammifères. Dans les poissons, on distingue habituellement les petits pélagiques, représentés principalement dans le secteur néritique et qui regroupent les poissons des familles des harengs, sardines, anchois, maquereaux, chinchards, capelans, poutassous, etc. Les grands pélagiques sont essentiellement des thons, des espadons et des requins, fréquentant normalement le secteur océanique mais pouvant également se trouver dans le secteur néritique. Pour les mollusques céphalopodes, il s'agit avant tout des encornets ou calmars, bons nageurs et vivant en concentrations importantes entre deux eaux. Les crustacés sont représentés par une espèce de crabe nageur

(le blue crab des Américains) et surtout par des sortes de petites crevettes planctoniques, formant souvent des bancs très denses et étendus, comme le « krill » que l'on trouve principalement dans les régions antarctiques.

L'exploitation des espèces pélagiques se fait à bord de bateaux de tailles très variées, de la pirogue au grand navire de pêche océanique. On ne considérera ici que les méthodes de capture pratiquées par les navires de pêche semi industrielle, unités de moyen ou fort tonnage dotées d'une autonomie suffisante pour opérer au large et parfois à de grandes distances de leur port d'attache.

6.1.2 La pêcherie réunionnaise

Depuis 1991, la pêche palangrière de surface ciblant l'espadon (*Xiphias gladius*) est pratiquée par des petites unités de la pêche côtière (à partir de 12 m) jusqu'à celles de grande pêche de plus de 25 mètres. Ces bateaux de tailles différentes exploitent des zones de pêche à des distances et pendant des marées de durées variables, selon leur capacité de stockage en cales frigorifiées ou réfrigérées et leur rayon d'action de pêche.

Ils utilisent tous la même technique : la palangre semi-automatique dérivante de surface, composée d'une ligne mère en monofilament (diamètre : $2-3$ mm) déroulée sur 30 à 50 miles de long (environ 60 à 100 km) à partir d'une bobine de stockage fixée sur le bateau. Sur cette ligne sont gréés des avançons munis d'hameçons appâtés à l'encornet, et des bouées qui assurent la flottabilité de l'engin qui dérive pendant la nuit au gré des courants. L'espèce ciblée est l'espadon ; les captures accessoires sont constituées de thonidés, dorades, requins et poissons porte-épée. Selon la stratégie de vente des armements et/ou de la durée des marées, le poisson est conservé à bord étêté-vidé en frais ou en longes (filets) congelées. L'ensemble est destiné à l'exportation vers l'Europe (principalement). La figure 6.2 page suivante illustre comment est mise en œuvre une pêcherie palangrière.

6.1.3 La capturabilité du poisson

Présents dans tous les océans, de l'équateur jusqu'à des latitudes parfois très élevées ($50° - 60°$), les espadons sont toutefois préférentiellement pêchés dans certaines régions plus restreintes de l'océan. Les principales pêcheries d'espadons se concentrent en effet à la frontière des grands courants (Gulf Stream sur la côte est-américaine, Kuroshio au Japon) ou en bordure de structures bathymétriques tels que des hauts-fonds ou des îles océaniques (Hawaii, Est-Australie). Les pêcheries de plein océan n'échappent pas à cette particularité : les pêcheurs ciblent les espadons en fonction de la configuration de l'environnement et des indices qu'ils perçoivent, considérant que le poisson n'est pas accessible uniformément partout.

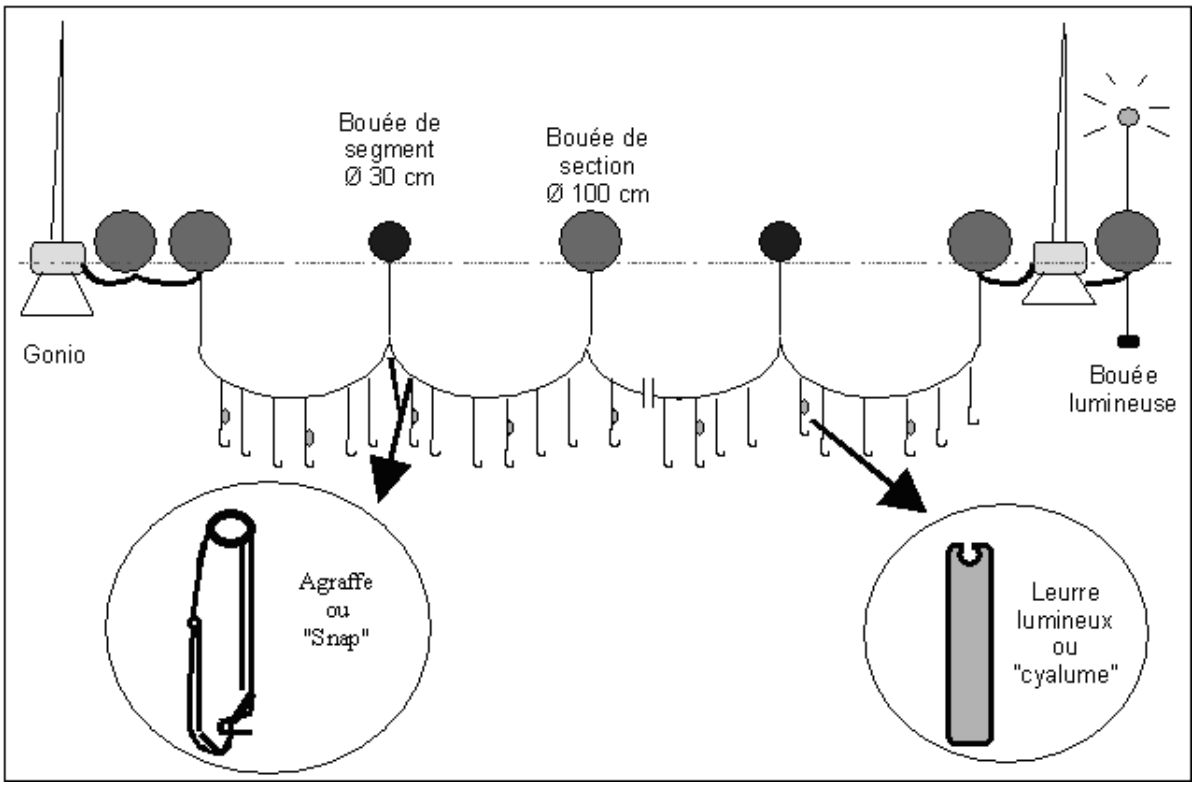


FIG. 6.2 – *Schéma d'une pêcherie palangrière*

C'est ce que les chercheurs halieutes appellent la capturabilité du poisson, qui dépend de son comportement dans l'océan, notamment vis-à-vis des paramètres physico-chimiques et biologiques de l'écosystème, mais aussi du comportement du pêcheur. Cette capturabilité, paramètre clé de la définition de bons indices d'abondance de la ressource, objet de la recherche halieutique, est généralement abordée par des outils statistiques complexes mais peu robustes, où les influences de l'environnement et des dynamiques spatio-temporelles ressource/pêcheurs sont difficilement prises en compte.

6.1.4 Télédétection et pêche océanique

Le guidage des flottilles recherchant les espèces pélagiques de surface comme le thon peut se faire par voie aérienne (avion opérant à partir de terre ou hélicoptère porté par le navire). Outre cette recherche par vision directe des bancs, on a parfois recours à la télédétection par voie satellitaire qui fournit des indications sur les conditions hydrologiques dans la zone de pêche, notamment la température des eaux superficielles. Ces données permettent de localiser les indices de présence potentielle de poissons et constituent un élément complémentaire de la prise de décision du pêcheur.

6.1.5 Importance économique et perspectives

Les statistiques annuelles de la Food and Agriculture Organization of the United Nations (1986) [Pel01] montrent que, sur les 80 millions de tonnes de ressources vivantes marines capturées dans le monde, environ 47,5 millions (soit 55,7 %) sont constituées par des espèces pélagiques. Parmi celles-ci, le groupe des harengs, sardines et anchois est le plus représenté, avec une prise totale de 23,9 millions de tonnes, soit près de 30 % de la production globale mondiale. Viennent ensuite, par ordre de tonnages décroissant : les chinchards et balaous, les maquereaux, les thons, bonites et espadons et les encornets. Bien qu'il soit difficile d'évaluer la proportion de cette production attribuable aux flottilles de pêche industrielle, on admet qu'elle en représente la plus grande partie, les captures de la pêche artisanale étant plus dispersées géographiquement et en moindre quantité. A l'heure actuelle, la Réunion débarque environ 2 000 tonnes de pélagiques par an.

Du point de vue de la valeur commerciale, il existe une grande disparité selon les espèces. Les petits pélagiques ont en général une faible valeur, leur utilisation étant essentiellement la transformation industrielle, en particulier dans les pêcheries minotières. Dans les grands pélagiques, on trouve par contre des espèces - thons, espadons - dont la chair est bien appréciée pour la consommation humaine et qui ont, de ce fait, une valeur plus élevée sur le marché mondial.

Compte tenu de l'augmentation constante de la puissance de pêche des flottilles et de la diminution de l'abondance estimée des ressources en espèces pélagiques, l'aménagement des pêcheries apparaît primordial. Selon les commissions scientifiques internationales chargées d'évaluer le niveau d'exploitation de ces ressources, la plupart des espèces pélagiques sont déjà surexploitées. Seules quelques rares espèces (thon blanc) sont encore considérées comme sous-exploitées. La production ne paraît donc pas devoir augmenter considérablement dans les prochaines années. En conséquence, des dispositions réglementaires sont prises chaque année, soit pour fixer les quantités maximales allouées pour les captures (quotas) dans les différents pays, soit pour restreindre l'effort de pêche. Mais la recherche d'une exploitation rationnelle des stocks à l'échelon mondial doit aussi prendre en considération l'évolution des flottes de pêche, en particulier la situation nouvelle créée par le développement des pêcheries de proximité, exploitées directement par des flottilles nationales armées dans les pays en voie de développement.

En complément de mesures d'aménagement, une diversification des captures permettrait un rééquilibrage de l'effort de pêche vers des espèces encore modérément exploitées. Il convient aussi de rechercher dès maintenant la mise au point de nouvelles techniques de capture pour les espèces peu ou pas utilisées, en particulier celles de la zone mésopélagique – comme les encornets océaniques dont on soupçonne des concentrations saisonnières abondantes dans certaines régions du globe.

[Pel01] *Données encyclopédiques.* – Hachette Multimédia / Hachette Livre, 2001.

6.2 Présentation

L'étude du comportement des grands organismes pélagiques dans leur environnement marin (hauturier et tropical) présente des difficultés du fait de l'immensité de l'écosystème considéré et des aspects extrêmement dynamiques de cet environnement. A l'heure actuelle, il n'existe pas de logiciel qui permettent de simuler le déplacement des grand organismes pélagiques à l'échelle d'un environnement. Il y a deux raisons principales à cela :

- Les scientifiques spécialisés dans l'étude des pélagiques n'ont pas la connaissance totale de leurs processus de raisonnement, ainsi que les paramètres essentiels qui interviennent dans leurs prises de décisions.
- L'environnement (*e.g.* l'océan Indien) est intrinsèquement dynamique et par conséquent il n'est pas facile de le décrire et de simuler cette dynamicité.

Du fait de sa situation géographique particulière (une île dans l'Océan Indien autour de laquelle une grande variété de pélagiques vivent), l'Ile de la Réunion ainsi que l'océan qui l'entoure, offre de nombreuses facilités pour étudier ces pélagiques. De plus, la stabilité de l'écosystème des ressources halieutiques est très fragile du fait de l'intensification depuis quelques années de la pêche professionnelle. C'est pourquoi, conjointement avec l'IFREMER[11] et l'IRD[12], nous avons commencé une nouvelle étude qui consiste à construire un système multi-agents qui vise à reproduire le comportement et le déplacement des pélagiques. Plus précisément, nous axons notre travail sur une espèce de pélagique bien précise : l'espadon (*Xiphias gladius*). Enfin, l'IRD a mis à notre disposition des données sous forme d'images satellites. Ces images sont prises à intervalles réguliers par des satellites tels que NOAA[13] ou ERS1-2[14].

Les figures 6.3 page suivante et 6.4 page 120 sont des exemples de ces images satellites.

Ces images décrivent une partie de l'océan Indien. La couleur (ici des niveaux de gris) des ces images peut avoir des significations différentes selon le satellite qui l'a pris. Dans le cas présenté ici, les couleurs vont représenter les paramètres suivant :

La température de surface de l'océan Cette caractéristique nous est fournie par les satellites NOAA. La température est obtenue par le capteur AVHRR[15] de ces satellites. Cette caractéristique est plus connue sous le nom de SST (Sea Surface Temperature)

11. l'Institut Français de Recherche pour l'Exploitation de la Mer
12. l'Institut de Recherche pour le Développement (anciennement appelé ORSTOM)
13. Satellite américain dont l'exploitation est assurée par la National Oceanic and Atmospheric Administration
14. Satellite européen dont l'exploitation est assurée par l'Agence Européenne pour l'Espace
15. Acronyme de Advanced Very High Resolution Radiometer

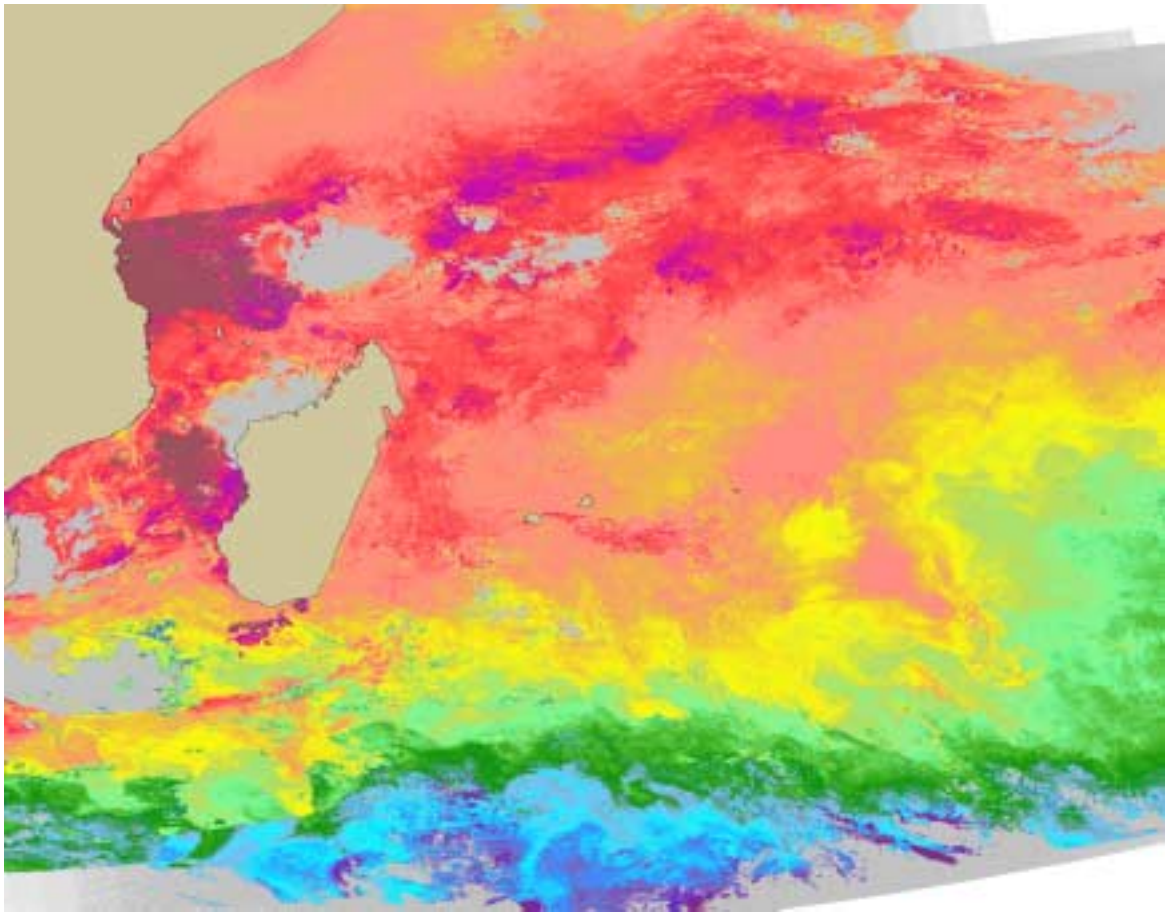


FIG. 6.3 – *Une image satellite représentant la température de surface*

La vorticité de l'océan C'est la valeur de la composante verticale du flux circulaire résultant de la friction du vent sur la surface. La valeur de la vorticité à un point de coordonnées x et y est calculée de la manière suivante :

$$vorticité = \frac{1}{\rho f}\left(\frac{\partial \tau_y}{\partial x} - \frac{\partial \tau_x}{\partial y}\right)$$

où τ est la tension de surface, ρ est la densité de l'océan et f le coefficient de Coriolis (ce coefficient est proportionnel à $\sin x$ où x est la latitude).

La bathymétrie de l'océan C'est la représentation de la topologie océanique (hauts fonds, canyon, crêtes sous-marine, ...).

La teneur en chlorophylle-a Exprimée en $mg \times m^{-2}$, c'est l'indice de la présence de phytoplancton (production primaire, 1^{er} niveau trophique de l'écosystème pélagique.

L'anomalie de hauteur d'eau . Cette caractéristique est exprimée en cm et elle indique les anomalies de la hauteur de l'eau par rapport au niveau moyen des oceans. Une anomalie positive indique qu'il y a une plongée d'eau en surface et donc une convergence. Une anomalie négative indique qu'il y a une remontée d'eau en surface

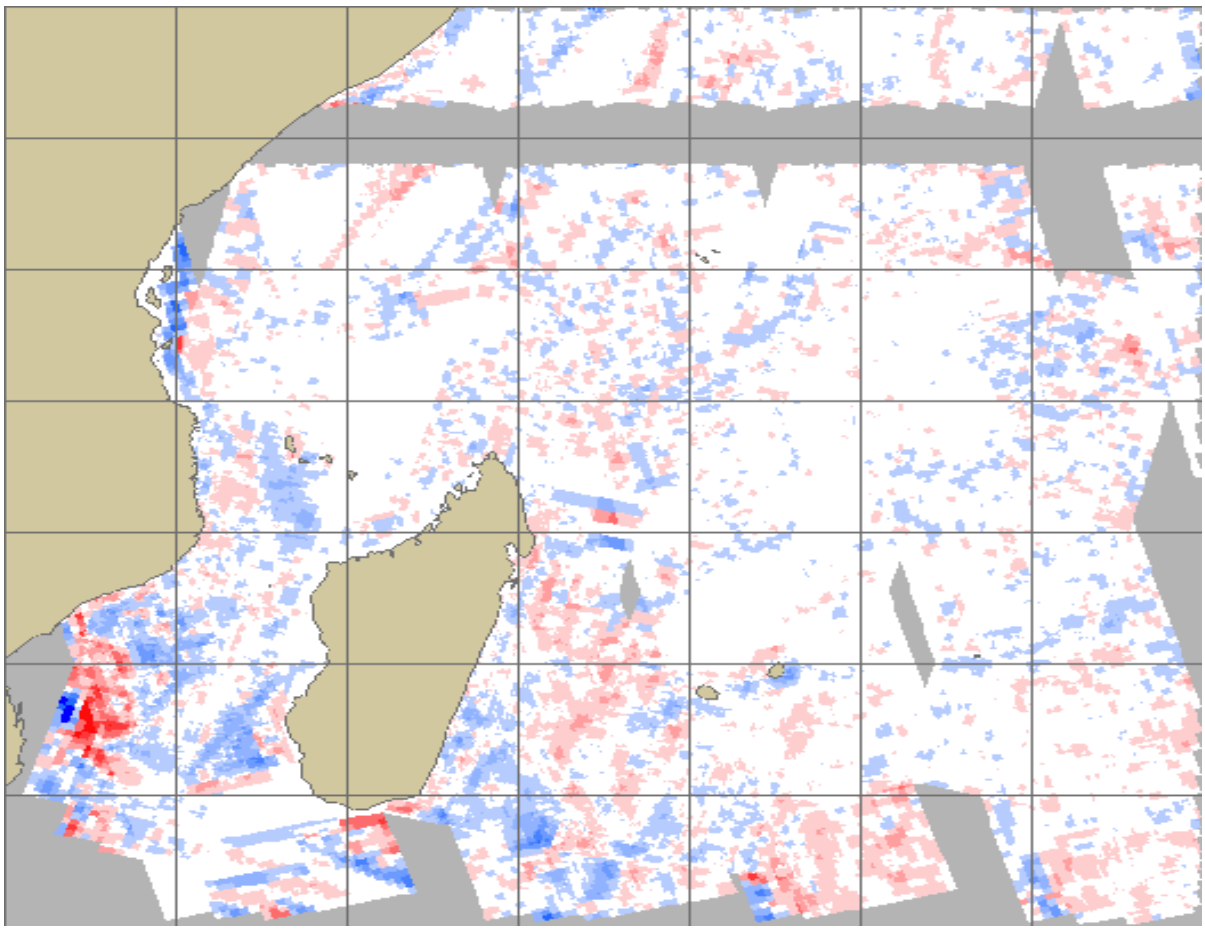


FIG. 6.4 – *Une image satellite représentant la vorticité*

et donc une divergence. Cette caractéristique est plus connue sous le nom de SLA (Sea Level Anomaly).

Toutes ces images satellites couvrent une surface allant de 5°N, 35°E à 30°S, 70°E. Elles sont collectées à intervalles réguliers (mais pas tous identiques) par l'IRD et stockées dans des bases de données spécifiques. De cette manière, on peut obtenir une description de l'état de l'océan à un instant t donné en fonction de certains paramètres bien précis. Le but de cette plate-forme va donc être de simuler les déplacements des bancs d'espadons au sein de ces paramètres océaniques.

Bien évidemment, le but de ce travail n'est pas de permettre aux pêcheurs professionnels d'augmenter leurs volumes de pêche, mais de pouvoir arriver à une meilleure gestion des ressources renouvelables de l'Ocean Indien.

6.3 Implémentation : la plate-forme MUFFINS

6.3.1 Objectifs

A l'heure actuelle, il y a un grand nombre de données qui sont stockées dans des bases de données dédiées. La plupart du temps, ces données contiennent des informations environnementales et par conséquence elles peuvent prendre un caractère très intéressant pour les personnes qui veulent construire des simulations à base de données environnementales. C'est la cas, en particulier des systèmes multi-agents. Mais, malheureusement, la plupart du temps, les plates-formes multi-agents et ces bases de données ne marchent pas sur le même système d'exploitation informatique ou elles ne sont pas écrites dans le même langage ou bien encore, elles ne sont pas physiquement située au même endroit. Jusqu'à présent, il n'y avait que deux solution pour faire coopérer ces deux mondes :

1. On devait demander aux personnes qui possédaient les bases de données de fournir ces données. De cette manière, on pouvait réaliser des simulations localement avec ces informations. Mais, malheureusement, cette solution, bien que valide, n'est pas facile à mettre en œuvre car les informations contenues dans les bases de données sont bien souvent sous le coup de copyright ou de droit de propriété et les organismes qui les possèdent ne veulent pas toujours fournir physiquement leurs informations. De ce fait, il devient irréalisable d'utiliser ces données pour nos simulations.

2. Il est possible de construire son propre système multi-agents et de le donner aux personnes qui possèdent les informations environnementales dans le but qu'ils réalisent eux-mêmes les simulations. Mais la encore, cette solution ne parait pas satisfaisante. En effet, pour les mêmes raisons que précédemment, les personnes ayant développé le simulateur multi-agents peuvent ne pas vouloir diffuser leur plate-forme. D'autre part, les personnes possédant les informations environnementales ne possèdent pas éventuellement les compétences adéquates pour pouvoir utiliser un simulateur qui, dans bien des cas, n'est encore qu'une version bêta.

Le développement massif d'Internet depuis les dernières années va venir pallier cette lacune. Les capacités et les techniques réseau vont, en effet, permettre de pouvoir mettre en œuvre la seule solution qui puisse satisfaire toutes les différentes parties de ce problème, à savoir laisser le simulateur multi-agents chez ses développeurs et laisser les informations environnementales chez ceux qui les possèdent. Entre ces deux mondes, il faudra donc utiliser le réseau pour faire communiquer ces parties. La figure 6.5 page suivante illustre l'architecture qui va être utilisée dans notre simulateur.

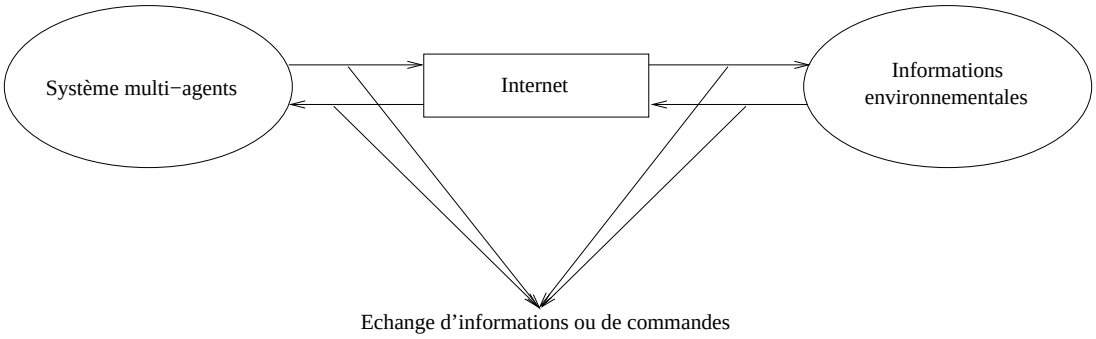


FIG. 6.5 – *L'architecture qui va être utilisée*

6.3.2 Fonctionnement général et dynamique

Pour ce faire, il faut mettre en œuvre un processus qui permette au système multi-agents d'envoyer des requêtes aux données environnementales et vice-versa, qui permette aux données environnementales de répondre à ces requêtes. Or, du côté des données environnementales, il n'existe aucun logiciel qui puisse décoder, interpréter, et exécuter les commandes envoyées par le système multi-agents. C'est pourquoi il faut fournir aux personnes possédant les données environnementales un petit serveur qui puisse faire l'interface entre le système multi-agents et les données. Ce serveur servira de point d'entrée sur le réseau. De cette manière on va donc pouvoir faire communiquer ces deux parties. Ce serveur sera, bien évidement, différent selon les données qu'il aura à traiter. Il est, en effet impossible de pouvoir mettre en œuvre un serveur suffisamment générique qui puisse traiter des données binaires, des données sous forme de matrice ou des données sous forme d'objets par exemple. Le fonctionnement du serveur sera le suivant : une fois lancé par l'utilisateur, il va vérifier tout d'abord réaliser une communication avec le système multi-agents afin de :

1. vérifier que le simulateur multi-agents associé est effectivement prêt a envoyer des requêtes ;
2. de s'authentifier auprès du simulateur – ceci afin d'être sûr qu'il n'y aura d'éventuels problèmes de sécurité – ;

ensuite, reste en veille sur le réseau afin de pouvoir écouter toutes les commandes provenant du système multi-agents ; lorsque la simulation est terminée, le système multi-agents va envoyer un message spécial qui va donc être perçu par le serveur et celui-ci va terminer toutes ses tâches en cours et il va ensuite automatiquement cesser sa veille sur le réseau et s'arrêter. Lors du déroulement d'une simulation, il a fallu mettre en place un modèle de scénario afin que les deux parties puissent communiquer en harmonie. La figure 6.6 page ci-contre présente un scénario de communication entre les deux parties.

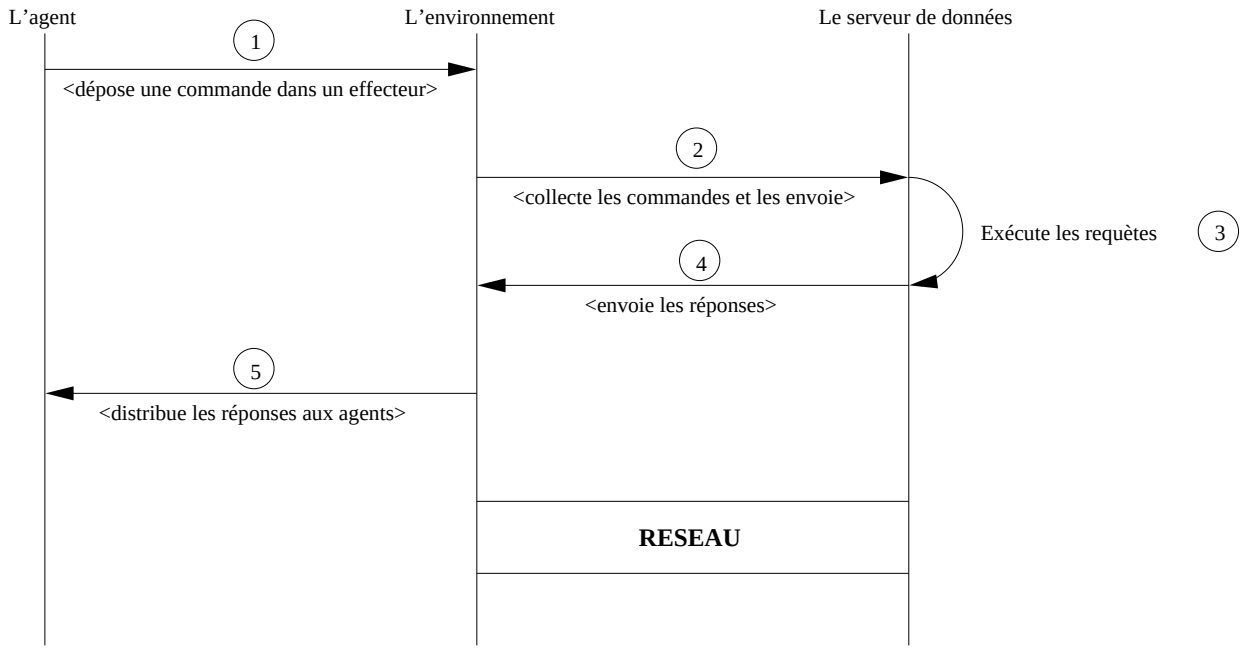


FIG. 6.6 – *Un scénario de communication*

Voyons maintenant étape par étape le déroulement d'un scénario :

1. Lorsqu'un agent désire prendre connaissance d'une information relative à l'état de l'environnement à une position donnée à un instant t, celui-ci va déposer la commande relative à son désir. Mais ce n'est pas le seul genre de commande qu'un agent peut déposer même si c'est apparemment la plus courante. Une autre forme de commande, qui va elle aussi être assez courante, concerne l'action sur l'environnement. En effet, lorsque qu'un agent désire agir sur l'environnement il va aussi déposer des commandes pour l'environnement.
2. L'environnement collecte toutes les commandes qui ont été déposées par l'ensemble des agents compris dans la simulation. Ces commandes vont ensuite être interprétée par l'environnement qui va les mettre toutes ensemble afin de ne réaliser qu'une seule requête sur le serveur et non pas autant de requêtes que de commandes. De plus, comme c'est l'environnement qui gère le temps, c'est lui qui va gérer la synchronisation entre le système multi-agents et le serveur de données environnementales. En effet, comme il a été présenté ci-dessus, toutes les commandes des agents sont relatives à un instant t donné et il est donc primordial que si le simulateur se trouve à un instant t, le serveur soit lui aussi au même instant t afin de pouvoir travailler sur les informations adéquates. C'est d'autant plus important que dans le cadre de MUFFINS, les cartes satellites étant prises à des instants donnés, elle changent donc en fonction du temps. Lorsque l'image satellite change (pas forcément à chaque pas de simulation, cela va dépendre de la manière dont a été conçu le simulateur multi-agents), sa description (*e.g* le contenu de la carte au format RGB) va être

automatiquement préparée pour un envoie. Donc l'environnement va automatiquement ajouter à chaque pas de temps messages spéciaux qui vont contenir l'heure courante de la simulation. Une fois l'ensemble du message préparé, l'environnement va émettre celui-ci sur le réseau à destination du serveur de données environnementales.

3. Lorsque le serveur de données environnementales reçoit un message du système multi-agents, la première tâche dont il va s'acquitter va être de séparer le message en autant de commandes émises par les agents du système. Une fois cette tâche réalisée, il va commencer par lire le message spécial concernant le temps. Si son propre temps diffère de celui qui lui a été envoyé par le système multi-agents, il va émettre pour celui-ci un message spécial lui demandant de confirmer le pas de temps dans lequel il se trouve. Dans le cas d'une réponse positive de la part du système multi-agents, il va actualiser son horloge interne. Dans le cas d'une réponse négative du système multi-agents, il va continuer en ignorant la partie temporelle contenu dans le message reçu. Une fois la partie temps traitée, le serveur de données environnementales va ensuite traiter les commandes concernant les agents. Comme il a été présenté précédemment, les commandes sont ordonnées par priorité et donc le serveur va tout naturellement exécuter les unes à la suite des autres en fonction de cette priorité. Dans le cas d'une consultation, le serveur va interroger la carte adéquate et récupérer la valeur voulue. De la même manière, dans le cas d'une action, le serveur va aller directement modifier les valeurs dans les cartes.

4. Tous les résultats des requêtes sont mises bout à bout afin de constituer une seule et unique trame. Néanmoins, ces résultats ne sont pas rangés n'importe comment. En effet, lorsque les commandes ont été reçues par le serveur, celles-ci étaient ordonnées selon des priorités fixées par chaque entité d'autonomie de l'agent, et ensuite ordonnées par l'environnement lorsque celui-ci a récolté toutes ces commandes ; et donc on peut se douter que si une commande prenait un caractère urgent pour un agent, la réponse associée à celle-ci va prendre elle aussi un caractère urgent pour l'agent. Donc les réponses aux différentes requêtes vont être ordonnées en fonction des priorités liées aux commandes. De plus, dans le cas d'un changement de carte (comme présenté ci-dessus), la description de la carte va être ajoutée en entête de cette trame. Une fois cette trame préparée, il va envoyer celle-ci au travers du réseau à destination du système multi-agents avec lequel ce serveur de données environnementales communique.

5. Lorsque l'environnement reçoit les résultats des requêtes qu'il a envoyé provenant du serveur de données environnementales, la première tâche qu'il va réaliser va consister à séparer la trame de données en autant d'informations que d'agents à informer. De la même manière que lorsque l'environnement envoie des requêtes au serveur de données, ces résultats de requêtes sont ordonnées par leurs priorités respectives. Donc, l'environnement va commencer par mettre dans les capteurs des agents des informations les plus prioritaires et ensuite les moins prioritaires, et ainsi de suite. Néanmoins, il faut noter qu'avant toutes choses, l'environnement va commencer par appliquer les lois physiques de l'environnement et ainsi que les phénomènes locaux.

Cette description du déroulement du scénario n'est pas complètement exhaustive, mais néanmoins, comme décrit dans la section 3 page 31, toutes les opérations sont effectuées. Que ce soit, par exemple, les opérations de mise à jour des priorités ou de réordonnancement des messages en fonction de leurs priorités.

6.3.3 Pour le modèle mono-environnenmental

Implémentation en UDP

Le protocole UDP[16] est un protocole orienté sans connexion qui tourne au-dessus des réseaux IP. UDP/IP ne fourni pratiquement aucun services de recouvrement d'erreurs, par contre, il permet d'envoyer et de recevoir directement des datagrammes au-dessus d'un réseau IP. La première utilisation du protocole UDP était d'envoyer des messages de type « broadcast » sur un réseau. Par contre, à l'heure actuelle, UDP est utilisé quand l'utilisation de TCP[17] serait trop complexe, trop lente ou tout simplement inadaptée.

Dans MUFFINS, quand un objet dépose une commande dans ses effecteurs, celles-ci sont décrites par des objets (au sens objet du terme). Malheureusement, comme le protocole UDP n'accepte pas de traiter un autre format qu'un tableau de bytes, ces commandes doivent pouvoir être décrites dans de simples chaînes de caractères. C'est pourquoi tous les objets de type `commande` contiennent tous une méthode qui permet de décrire celles-ci dans une chaîne de caractère. Ensuite, grâce aux fonctions prédéfinies de Java, cette chaîne de caractères va être transformées en un tableau de bytes. Cette opération de conversion est effectuée par l'environnement lorsqu'il a collecté toutes les commandes des agents. Une fois cette opération de transformation effectuée, l'environnement va construire un seul et même tableau de bytes contenant tous les tableaux de bytes des commandes. Une fois ce tableau construit, il va de la même manière ajouter au début du tableau un ensemble de bytes contenant une description de pas de simulation dans lequel se trouve dans lequel se trouve le simulateur. Une fois toutes ces opérations de codage effectuées, l'environnement va envoyer ce tableau de bytes au serveur de données environnementales au travers d'une connexion UDP. Quand le serveur reçoit cet ensemble de bytes, il doit pouvoir être capable de les décoder et de les interpréter. C'est pourquoi le serveur contient un interpréteur de commandes. Cet interpréteur fonctionne à base de règles. A chaque fois qu'il lit une chaîne de bytes, il les transforme en chaîne de caractères. Grâce aux règles définies dans l'interpréteur, il va pouvoir les reconnaître et lui associer une méthode. Bien entendu, ces règles sont entièrement dépendantes de la simulation considérée et elle doivent être en accord avec les informations envoyées par l'environnement. Ces chaînes de caractères

16. Acronyme de User Datagram Protocol

17. Acronyme de Transfer Control Protocol : un autre protocole au-dessus d'IP qui est utilisé lorsque les programmeurs veulent être sûr qu'un paquet envoyé au travers du réseau soit bien arrivé. Malheureusement, du fait de ce contrôle d'erreur, TCP est moins performant qu'UDP. Néanmoins, TCP est à l'heure actuelle très largement utilisé par les programmeurs.

sont décrites dans un format « à la lisp ». Dans MUFFINS, la requête `(swordfish 1 (temp? (12, 24)))` correspond à la question : pour le swordfish numéro 1, quelle est la température de l'environnement à la position (12,24) ? Les requêtes de type « consultation environnementale » auront la forme suivante :

```
(type_de_l'agent id (type_de_la_requête? (position en x,y)))
```

Les requêtes de type « réponse de consultation environnementale » seront, de manière analogue, de la forme suivante :

```
(type_de_l'agent id (type_de_la_réponse# (position en x,y)))
```

De la même manière, toutes les primitives présentées en 3 page 31 sont définies ici dans une syntaxe bien particulière ce qui permet à l'interpréteur de pouvoir les identifier facilement.

Une fois que le serveur de données environnementales, via l'interpréteur de requêtes, il les exécute sur les données environnementales contenues dans la base de données attachée au serveur. Afin de faciliter les traitements au niveau de l'agent, il a été choisi que pour toute commande émise par l'agent au niveau de ses effecteurs, il va forcément correspondre une réponse émise par le serveur. En effet, comme toutes les commandes émises par l'agent sont ordonnées selon une priorité définie par le gestionnaire de priorité rattaché au système conatif, il aurait été plus difficile de pour l'agent de reconnaître à quelle commande correspondait telle réponse. Avec le choix effectué ici, l'agent saura que pour toute commande c_i avec une priorité p_i, il y aura une réponse r_i avec la même priorité p_i. C'est pourquoi, lorsque le serveur de données environnementales aura exécuté toutes les commandes sur les cartes de données, il va reconstruire une nouvelle chaîne de caractères qui va contenir les réponses associées aux commandes (par exemple, dans le cas d'une commande de modification de l'environnement, on pourrai s'attendre à ce qu'il n'y ai pas de réponse. Mais comme le choix qui a été fait nous impose de fournir une valeur de retour, il a été décidé que dans ce cas, la chaîne associée contiendra juste, outre les éléments fondamentaux tels que « type de l'agent », « id », ..., la valeur de retour qui sera mise à `true` si la modification a été effectuée avec succès et la valeur de retour qui sera mise à `false` dans le cas contraire. De cette manière, l'agent pourra être informé que sa demande de modification n'a pas été effectuée et ainsi, le système conatif pourra gérer cet état de fait).

Comme toutes les commandes possèdent des réponses associées, le serveur de données environnementales va donc construire, de la même manière que pour les commandes, une chaîne de caractères associée à chaque réponse. Puis, il va ensuite construire le tableau de bytes associé à chaque commande et ensuite, il va construire le tableau de byte global. Une fois cette opération effectuée, le serveur va émettre à l'intention de l'environnement du système multi-agents une trame spéciale qui va le prévenir qu'une réponse va arriver. Tout de suite après, il va envoyer à l'environnement toute la trame contenant toutes les réponses associées aux commandes.

Quand l'environnement reçoit la trame spéciale en provenance du serveur de données environnementales, il se met en écoute sur un port spécial qui va lui permettre de lire le flot d'informations (qui peut être volumineux). Puis l'environnement va décomposer la trame contenant les réponses en autant de réponses dédiées pour un seul et unique agent. Enfin, il va déposer ces réponses dans chaque effecteur associé à celle-ci. De cette manière, les agents vont maintenant pouvoir prendre en compte les résultats des commandes qu'ils ont effectué sur l'environnement.

Comme il a été présenté dans la section 6.3.1 page 121, l'environnement contient un minuteur qui va cadencer le rythme de la simulation. Dans MUFFINS, ce minuteur est un « thread » [NP97, Fla97]. Quand le simulateur est démarré, celui-ci rentre dans une boucle infinie, et à chaque pas de simulation, il appelle deux méthodes spéciales : `newHour` et `newDay` dans l'environnement. Lors d'un changement d'heure, il n'est pas nécessaire que le serveur de données environnementales soit tenu au courant. C'est pourquoi, lorsque la méthode `newHour` est invoquée, toutes les modifications se passent au niveau de l'environnement et des agents. Par contre, lorsque la méthode `newDay` est invoquée, le serveur de données environnementales va être prévenu afin qu'il puisse savoir sur quelle carte l'environnement se situe. De cette manière, on est sûr que le serveur sera toujours en accord avec le simulateur multi-agents. La figure 6.7 page suivante est un diagramme utilisant la représentation OMT [RBEL91] du serveur de données environnementales en UDP.

Implémentation avec RMI

La technologie RMI[18] permet à un objet, qui s'exécute sur une machine, d'invoquer une méthode sur un objet Java qui se situe sur une autre machine distante. La figure 6.8 page 129 illustre ce mécanisme.

Le « stub » a trois principales fonctions :

- Il présente la même interface distante que l'objet auquel il est rattaché ;
- Il travaille avec la machine virtuelle Java et le système RMI sur la machine distante pour « sérialiser » tous les arguments d'un appel à une méthode distante et pour envoyer ces informations au serveur ;
- Il reçoit tous les résultats provenant d'un appel à une méthode distante et les renvoie à l'objet auquel il est rattaché.

18. acronyme de Remote Method Invokation

[NP97] Patrick NIEMEYER et Joshua PECK. – *Exploring Java (2nd Edition).* – O'Reilly, 1997, *The Java Series.*

[Fla97] David FLANAGAN. – *Java in a nutshell (version 1.1).* – O'Reilly, 1997, 2nd édition.

[RBEL91] J. RUMBAUGH, M. BLAHA, F. EDDY et W. LORENSEN. – *Object-Oriented Modeling and Design.* – London, England, Prentice All International, 1991.

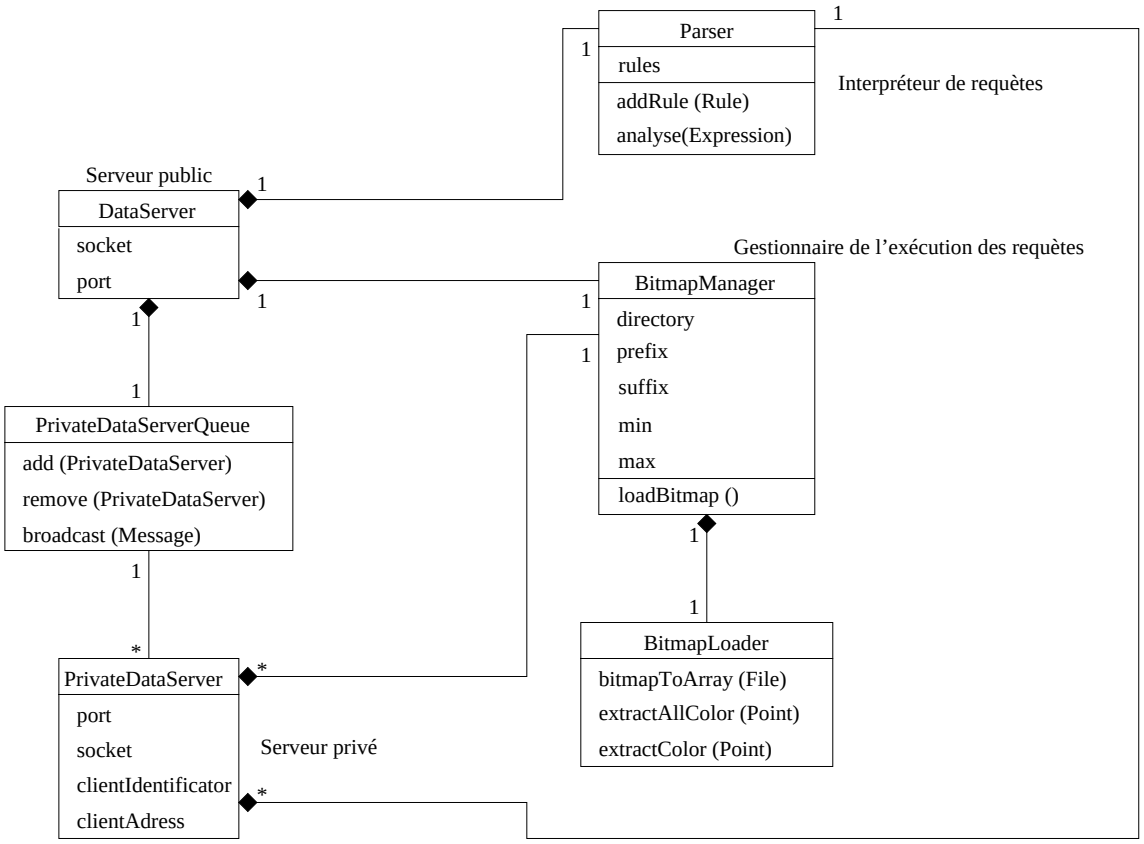


FIG. 6.7 – *La représentation objet du modèle* UDP

De la même manière, le « skeleton » a trois principales fonctions :

- Il reçoit les appels à une méthode distante, ainsi que les arguments qui lui sont associés ;
- Il travaille avec la machine virtuelle Java et le système RMI pour « désérialiser » chaque argument d'un appel à une méthode distante et ensuite, il invoque la méthode appropriée sur l'objet situé au niveau du serveur ;
- Il reçoit toutes les valeurs de retour d'un appel à une méthode distante et il les envoie à la machine distante.

Une description complète de la technologie RMI et de son fonctionnement pourra être trouvée dans [O'N98].

Dans MUFFINS, quand on utilise la technologie RMI, le premier travail consiste à définir quelle méthode sera publique et par conséquent, les méthodes qui seront capables d'être invoquées à distance. En accord avec le manuel RMI [O'N98], ces méthodes doivent

[O'N98] Joseph O'NEIL. – *Java Beans Programming from the ground up.* – Osborne – McGraw-Hill, 1998.

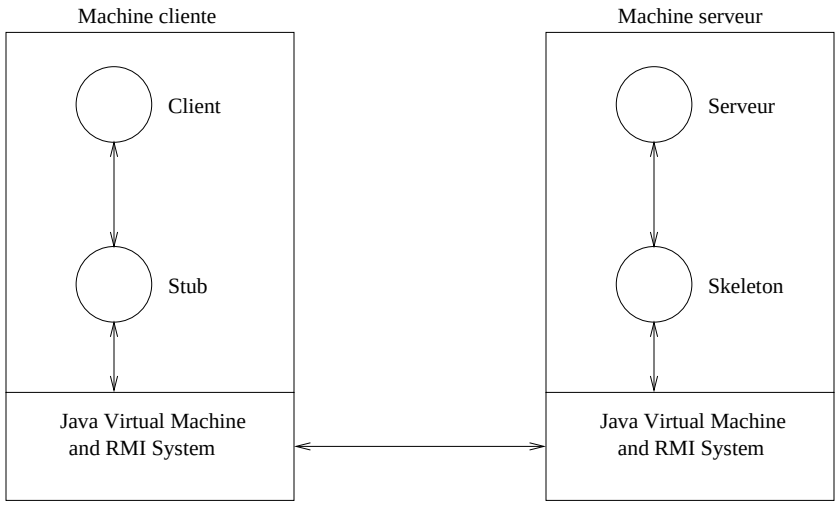


FIG. 6.8 – *Fonctionnement de* RMI

être déclarées dans une interface qui hérite de l'interface prédéfinie `java.rmi.remote`. Dans cette partie générique de la plate-forme, il a été crée une interface spéciale dédiée à cette fonctionnalité. Cette interface est appelée `RMIData`. Par exemple, dans MUFFINS, le serveur RMI implémente une interface appelée `IndianOceanData`. Tout naturellement, cette interface hérite de l'interface `RMIData` et elle définie toutes les méthodes qui pourront être appelées à distance. A savoir :

- La méthode `getTemp(Point)` qui va retourner la valeur de la température à un point donné ;
- La méthode `setTemp(Point, Temp)` qui va modifier la valeur de la température à un point donné ;
- La méthode `getVorticity(Point)` qui va retourner la valeur de la vorticité à un point donné ;
- La méthode `setVorticity(Point, Vorticity)` qui va modifier la valeur de la vorticité à un point donné ;
- La méthode `getBathymetrie(Point)` qui va retourner la valeur de la bathymétrie à un point donné ;
- La méthode `setBathymetrie(Point, Bathymetrie)` qui va modifier la valeur de la bathymétrie à un point donné.
- La méthode `newDay` qui va indiquer au serveur qu'il doit changer l'ensemble des cartes satellites qu'il possède. Et aussi incrémenter son temps local afin que l'environnement (et le système multi-agents) et le serveur de données environnementales soient synchrones.

Quand le serveur a été compilé en utilisant la commande `rmic` de Java, deux classes spéciales ont été automatiquement créées : le « skeleton » et le « stub ». Par conséquent, l'environnement est alors capable d'invoquer des méthodes à distance. En fonction du

scénario décrit en 6.3.2 page 122, quand un agent (*e.g* un espadon) va déposer une commande dans ses effecteurs, l'environnement va les collecter et ensuite, il va les associer avec les méthodes adéquates. Et finalement, il va invoquer ces méthodes à distance sur le serveur de données environnementales. Quand le serveur exécute les méthodes qui lui sont demandées par l'environnement, il va envoyer les valeurs de retour (s'il y en a) à l'environnement par le biais du réseau et toujours en utilisant les fonctionnalités de RMI. Le figure 6.9 est un diagramme OMT du modèle de serveur de données environnementales en RMI.

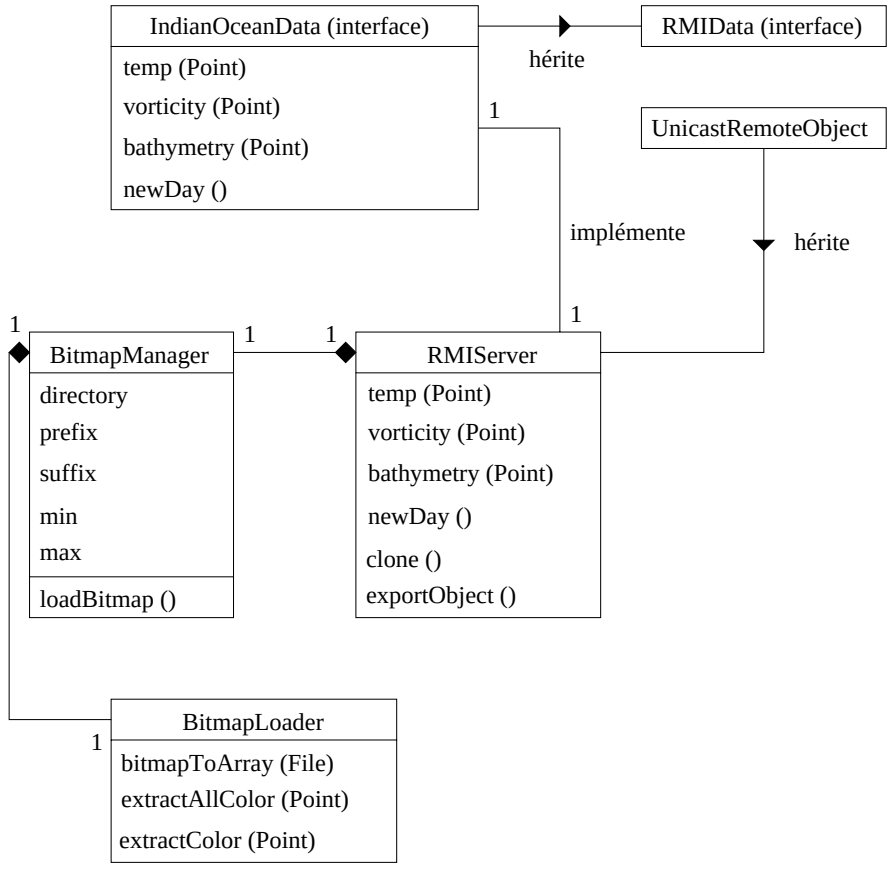


FIG. 6.9 – *Diagramme* OMT *du serveur* RMI

Résultats et performances

Ces deux technologies – RMI et UDP – sont très différentes les unes des autres. En effet, d'un côté, UDP est un protocole bas niveau et privilégie les performances plutôt que la sécurité. De plus, l'utilisation d'UDP n'est pas très facile car elle requiert l'implantation d'un protocole spécifique. Et par conséquent, un interpréteur pour ce protocole est nécessaire. Par contre, de l'autre côté, RMI fournit un service de haut niveau, ainsi qu'un grand nombre de facilités (et notamment pendant la phase de conception et de programmation). En effet, il est plus facile de faire communiquer des entités distribuées sur le

réseau par le biais d'appels à des méthodes qui masquent justement cette partie réseau, plutôt que de gérer un protocole de communication plus ou moins compliqué, mais qui dans tous les cas va s'avérer plus difficile à gérer. Malheureusement ces avantages doivent avoir forcément un coût...Et ce coût c'est la vitesse. C'est pourquoi il a été décidé de mesurer comment ces deux approches se comportement lorsque le nombre de clients (des agents) augmente. Ce critère est le plus important (on pourrait mesurer un certain nombre d'autres critères. Par exemple, le nombre d'informations perdues sur le réseau, le temps mis par le serveur pour exécuter toutes les commandes, le temps mis par l'environnement pour collecter toutes les informations...) car dans le cadre d'une simulation multi-agents, on peut avoir très facilement un très grand nombre d'agents et il convient de pouvoir étudier le comportement de ces deux architectures quand le nombre d'agents augmente.

Ces mesures ont été réalisées sur deux stations Sun UltraSparc 5 sous Solaris 2.6 avec 64 Mo de RAM et reliées entre elles par un réseau local ethernet à 10 mb/s. Le scénario de cette analyse est le suivant : n (de 1 à 100) agent(s) sont crées. il y a donc automatiquement un système conatif et une représentation environnementale qui sont crées par agent. Ensuite, chaque agent dépose deux fois une commande dans ses effecteurs (5 secondes entre chaque requêtes). Ces requêtes concernent la récupération de la température ou de la vorticité ou de la bathymétrie en un point donné (le choix entre ces trois types de requêtes se faisant aléatoirement). Une fois la valeur récupérée dans ses capteurs, l'agent quitte la simulation. La table 6.1 représente les résultats obtenus lors de ces simulations avec différents agents.

Nombre d'agents	Serveur UDP	Serveur RMI
1	5.6 sec	7.3 sec
3	6.4 sec	8.6 sec
10	8.6 sec	14.3 sec
30	13.2 sec	27.0 sec
50	15.1 sec	time out
100	Nombre maximal de socket ouvertes	

TAB. 6.1 – *Performances du protocole* UDP *et de la technologie* RMI

On peut observer que l'utilisation d'UDP est plus performante et plus rapide que la technologie RMI. De plus, on peut s'apercevoir qu'on peut utiliser plus d'agents avec UDP qu'avec RMI (la limite pour RMI est de 30 agents alors que la limite avec UDP est de 50 agents). Mais comme il n'a pas été pris en compte les erreurs entre les deux machines dues au réseau, on ne peut pas savoir si UDP est plus sûr ou moins sûr que RMI. De plus, l'approche UDP ne peut être utilisée que dans le cas d'un réseau local car la majorité des firewalls mis en place par les administrateurs systèmes pour sécuriser leur réseau n'acceptent pas les paquets UDP. Alors qu'avec la technologie RMI (grâce à la `rmi registry` et à l'opération de `lookup`), les firewalls n'effacent pas les paquets utilisés pour invoquer des méthodes distantes.

Maintenant que la communication entre ces deux univers que sont les systèmes multi-

agents et les bases de données environnementales est mise en place, il est possible de s'attacher à simuler les déplacements des espadons au sein de l'environnement dans le système multi-agents. Il n'est pas facile, à ce jour, d'essayer de reproduire ces déplacements. La seule constatation fiable qu'il a été faite in situ lors de différentes campagnes d'observations des pélagiques exhibe le fait que ces pélagiques suivent, d'une manière générale, les gradients de température et de vorticité. Même si cette constatation n'est pas encore absolument sûre, elle se rapproche assez de ce que l'on peut observer. C'est pourquoi, à chaque pas de simulation, l'agent va calculer une nouvelle trajectoire de la manière suivante :

- On va calculer le gradient maximum entre la cellule c_e où se trouve l'espadon et les 8 cellules (notée c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8) qui entourent c_e. Et ce pour la température, la vorticité et la batymétrie.
- On aura le maximum de la température (noté Max_t) qui va être calculé de la manière suivante :

$$Max_t = Max(c_e - c_1, c_e - c_2, c_e - c_3, c_e - c_4, c_e - c_5, c_e - c_6, c_e - c_7, c_e - c_8)$$

- On aura le maximum de la vorticité (noté Max_v) qui va être calculé de la manière suivante :

$$Max_v = Max(c_e - c_1, c_e - c_2, c_e - c_3, c_e - c_4, c_e - c_5, c_e - c_6, c_e - c_7, c_e - c_8)$$

- On aura le maximum de la batymétrie (noté Max_b) qui va être calculé de la manière suivante :

$$Max_b = Max(c_e - c_1, c_e - c_2, c_e - c_3, c_e - c_4, c_e - c_5, c_e - c_6, c_e - c_7, c_e - c_8)$$

- Le gradient maximum (noté MAX) total sera calculé de la manière suivante :

$$MAX = Max(Max_t, Max_v, Max_b)$$

- Et finalement, le processus de prise de décision concernant la nouvelle cellule où va aller l'espadon va être le suivant :

  si $MAX = 0$ **alors**
  nouvelle cellule = $Random(c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8)$
  sinon
  nouvelle cellule = cellule qui contient le MAX

Finalement, la figure 6.10 page suivante illustre le déplacement d'agents espadons au sein de l'océan décrit par des images satellites dans le simulateur MUFFINS.

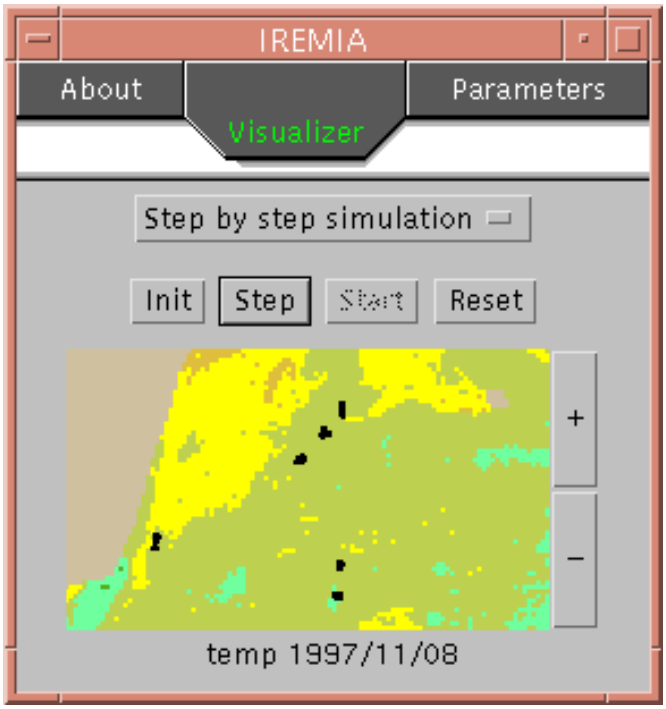


FIG. 6.10 – *Une capture d'écran de* MUFFINS

6.3.4 Pour le modèle multi-environnemental

Dans cette partie, nous allons présenter comment, à partir de la plate-forme présente en section 5 page 99. Tout d'abord, nous allons présenter comment les classes nécessaires à la simulation ont été définies. Ensuite, nous allons présenter le comportement principal utiliser lors de la simulation. Après cette phase, nous allons présenter comment l'accés aux données a été réalisé en utilisant le toolkit de M. Desruisseaux [DSD00, Des] que nous avons porté sous MYSQL [MyS]. Enfin, nous présentons le simulateur, ainsi que les resultats obtenus lors de diverses simulations.

Définition des classes

Les classes utilisés dans MUFFINS sont directement dérivées du noyau générique de la plate-forme multi-environnements. Elles sont au nombre de 16. Nous allons donc mainte-

[DSD00] Martin DESRUISSEAUX, Robert SIRON et Hervé DEMARCQ. – *Regrouper les efforts de création d'outils géographiques Java : Proposition d'une avenue.* – Rapport de recherche, Observatoire du Saint Laurent – Canada, 2000.

[Des] Martin DESRUISSEAUX. – Seagis Home Page. – http://seagis.sourceforge.net/.

[MyS] MySql Home Page. – http://www.mysql.com/.

nant détailler ces classes et quelles sont leurs rôles au sein de MUFFINS.

- La classe `FisherySociety`. Sous-classe de la classe `Society` du noyau générique, elle représente la société des agents. Elle regroupe donc l'ensemble des agents présents dans le système.
- La classe `Swordfish`. Sous-classe de la classe `Agent` du noyau générique, elle représente l'agent espadon utilisé pour les simulations.
- La classe `SwoConativeSystem`. Sous-classe de la classe `ConativeSystem` du noyau générique, elle représente la partie conative de l'agent. C'est cette classe qui définie donc les comportements des agents espadons de la simulation. En outre, c'est le point de passage de toutes les perceptions et de toutes les actions.
- La classe `SwoVBDLink`. Sous-classe de la classe `VirtualBDLink` du noyau générique, elle représente le lien qui existe entre l'agent espadon et l'instance virtuelle dans l'environnement. C'est le long de ce lien que vont transiter les commandes qui doivent être envoyées vers l'instance virtuelle dans l'environnement et les perceptions qui ont été réalisées par l'instance virtuelle dans l'environnement.
- La classe `SwoVPhysicalInstance`. Sous-classe de la classe `VirtualPhysicalInstance` du noyau générique, elle représente l'instance virtuelle dans l'environnement de l'agent. Cette classe va donc être reliée à l'agent par le biais de la classe `SwoVBDLink`.
- La classe `TypedBDLink`. Sous classe de la classe `BDLink` du noyau générique, elle représente le lien qu'il existe entre l'agent espadon et ses instances dans l'environnement. Cette classe est typée, c'est à dire qu'elle va dépendre du type de l'environnement auquel elle est reliée.
- La classe `TypedPhysicalInstance`. Sous-classe de la classe `PhysicalInstance` du noyau générique, elle représente l'instance dans l'environnement. De la même manière que pour la classe précédente, elle est typée et son type est le même que celui de l'environnement.
- La classe `SeasEnvironnement`. Sous-classe abstraite de la classe `Environnement` du noyau générique. Cette classe permet de se connecter à la base de données environnementale et permet aussi de se connecter à la base de données qui va contenir les résultats des simulations.
- La classe `SeasIntegrityLink`. Sous-classe de la classe `IntegrityLink` du noyau générique, elle représente le lien qui relie les environnements avec l'environnement virtuel.
- La classe `TypedSeasEnvironment`. Sous-classe de la classe `SeasEnvironment` de MUFFINS, elle représente un environnement réel dans lequel l'agent espadon va être plongé. Cet environnement est typé et c'est ce type qui va être répercuté à toutes les autres classes qui sont en relation avec cet environnement.
- La classe `VirtualSeasEnvironment`. Sous-classe abstraite de la classe `VirtualEnvironnement` du noyau générique.
- La classe `VirtualOcean`. Sous-classe de la classe `VirtualSeasEnvironment` de MUFFINS, elle représente l'environnement virtuel utilisé dans MUFFINS

- La classe `VirtualOceanTimer`. Sous classe de la classe `Timer` du noyau générique. Elle va représenter le gestionnaire du temps pour l'environnement virtuel.
- La classe `TypedSeasTimer`. De la même manière que pour la classe précédente, cette classe est une sous-classe de la classe `Timer` du noyau générique. Elle va représenter le gestionnaire du temps pour l'environnement typé auquel elle est rattachée.
- La classe `ResultConnectivity`. Sous-classe de la classe `DataBaseConnectivity` du noyau générique, elle permet de se connecter sur une base de données dédiée à la gestion des résultats de simulation. De plus, elle définie toutes les primitives permettant d'effectuer les opérations de sauvegarde des résultats de simulations.
- La classe `SeasConnectivity`. Sous-classe de la classe `ImagedDataBaseConnectivity` du noyau générique, elle permet de se connecter sur une base de données dédiée à la gestion des images satellites. De plus, elle contient toutes les requêtes nécessaires pour récupérer les images satellites d'une date donnée. Enfin, elle contient aussi les requêtes permettant de récupérer des valeurs des paramètres environnementaux contenus dans les images satellites.

De plus, on peut noter que, en plus des classes citées précédement, un autre ensemble de classe est utilisé, ce sont les classes qui définissent les messages échangés entre les diverse entités de notre modèle. A savoir les classes : `MotionInformation` qui représente une commande et `TypedInformation` qui représente une information perçue dans un environnement d'un type donnée.

Enfin, à cette liste, il faut ajouter les effecteurs et les capteurs utilisés dans la simulation. Nous avons donc défini les classes : `MotionEffector` qui va permettre à l'agent de se déplacer au sein de son environnement et `TypedCaptor` qui va permettre d'aller cherché une valeur environnementale à une position donnée et pour un type donné.

Comportements utilisés

Une grand nombre de travaux ont déjà été effectué sur le comportement des espèces pélagiques. Citons par exemple : les travaux de L. Dagorn [Dag94], les travaux de D.L. De Angelis [AY84] et ceux de A. Huth et C. Wissel [HW94]. Pour notre part, nous nous sommes concentrés sur des comportements les plus classiques qui sont décrits dans la littérature ci-dessus. Nous avons donc utilisé principalement trois types de comportement :

[Dag94] Laurent DAGORN. – *Le comportement des thons tropicaux modélisé selon les principes de la vie artificielle.* – Rennes, Thèse de Doctorat, ENSAR, 1994.

[AY84] D.L. De ANGELIS et G.T. YEH. – *An Introduction to Modeling Migratory Behavior of Fishes*, chap. " ", pp. 445–469. – Plenum Press, New-York and London, 1984, *Mechanisms of Migration in Fishes.*

[HW94] A. HUTH et C. WISSEL. – The simulation of fish schools in comparison with experimental data. *Ecological Modelling*, vol. 75/76, 1994, pp. 135–145.

la marche aléatoire, la maximisation du gradient et la symétrie bilatérale.

Marche aléatoire Comme son nom l'indique, cette méthode consiste à faire évoluer de manière totalement aléatoire l'espadon dans l'océan Indien. Ceci implique que sa direction de déplacement et ainsi que sa vitesse de déplacement ne seront pas forcément les mêmes à chaque pas de simulation. Cette méthode, bien que produisant parfois des résultats surprenant de réalisme, n'est pas à proprement parler réaliste, mais elle permet de réaliser rapidement des simulations lors de la mise en œuvre du simulateur.

Maximisation du gradient Cette méthode consiste à récupérer des valeurs dans l'environnement autour d'une position donnée. Lorsque toutes ces valeurs sont récupérées, on calcule chaque différence entre la valeur à la position actuelle et la valeur pour les positions autour de la position initiale. Afin de prendre le meilleur gradient, il suffit de prendre la différence la plus forte des valeurs calculées. Une fois cette étape réalisée, il suffit de prendre comme direction de déplacement l'endroit où se situe le gradient le plus fort. Cette approche dépend essentiellement de quelle distance on prend autour de la position actuelle de l'espadon. En effet, si on prend une distance très courte, il y a peu de chances que l'on trouve un gradient car la probabilité que l'espadon se trouve dans une zone homogène est assez forte. Par contre, si on prend un rayon plus grand, on va trouver plus facilement des gradients, mais dans ce cas le coût en terme de temps de calcul est beaucoup plus grand. Néanmoins, si on prend comme rayon de calcul le déplacement maximum que peut effectuer un espadon sur un pas de temps, on peut conserver des temps de calcul corrects tout en se rapprochant d'une certaine réalité.

Symétrie bilatérale Un espadon a une très forte probabilité de ne pas changer de direction entre deux pas de temps de 1 heure : on peut considérer que 80% des déplacements se font dans une direction allant de -45° à +45° par rapport à la direction précédente (tout droit, 365 pas de temps sur 457), 15% entre 35° et 45° ou +45° et +135° (virages, 81 pas de temps sur 457), et seulement 5% pour le reste (demi-tour, 11 pas de temps sur 457). Ainsi, la règle de décision codée pour le comportement « symétrie bilatérale » (bilat) tient compte de ces observations réelles : un espadon n'a pas la même probabilité de déplacement dans toutes les directions. Cet effet a été codé comme suit : à chaque pas de temps, l'agent espadon sélectionne l'ensemble des pixels dans un rayon correspondant à son rayon de perception (paramètre « profondeur », d'ailleurs mal trouvé puisqu'il y a risque de confusion avec la vraie profondeur atteinte par le poisson, qui sera vraisemblablement intégrée dans MUFINS un jour ou l'autre). Parmi ces pixels, il a plus de chances de sélectionner un pixel « devant » lui que « derrière ». Pour ce faire, on tire un nombre aléatoire (entre 0 et 1) pour chaque pixel sélectionné : s'il est « devant » l'agent et que le nombre aléatoire est inférieur ou égal à 0.8 (80% de chances), il est conservé dans la liste des pixels potentiels ; de même, si le pixels est « sur le côté » et que le nombre aléatoire est inférieur ou égal à 0.15 (15% de chances), il est conservé. Au final, le pixel définitif est choisi aléatoirement au sein des pixels conservés.

Cet effet de symétrie bilatérale permet de faire évoluer les agents sur de plus grandes

distances que sans cet effet (où les agents ont tendance à effectuer des « allers-retours » entre deux même pixels).

Les vitesses de déplacements moyennes se situent autour de deux modes : 1 km/h et 3 km/h, qui correspondent plus ou moins respectivement aux zones côtières avec activité de chasse sur le fond (sur un banc de poisson de fonds assez statique) et aux zones du large, avec une plus grande mobilité. La vitesse maximale observée ici est de près de 9 km/h, ce qui est une pointe rarement atteinte (exceptionnelle). La vitesse maximale observée est d'environ 6 km/h (soit 144 km/jour). Cette vitesse maximale peut être considérée comme atteignable dans des conditions de courants favorables, une vitesse de 3,5 km/h (84 km/jour) semble plus cohérente avec la physiologie de l'animal et ses performances de nage « pure ». C'est cette distance de 84 km qui a été choisie comme la distance maximale journalière parcourue par un espadon dans le modèle. Ainsi, 84 km correspond à 45 pixels sur une carte de SST, 11 pixels sur une carte de vorticité, 3 pixels sur une carte de SLA et 9 pixels sur une carte de Chl-a. La « profondeur » choisie (rayon d'exploration à chaque pas de temps, « step » dans le modèle) a déterminé le nombre de pas de temps par jour : pour une profondeur de 15 pixels sur une carte de SST, 3 pas de temps journaliers sont nécessaires pour atteindre 45 pixels au maximum ; pour une profondeur de 5 pixels, 9 pas de temps sont nécessaires.

Le lecteur désirant approfondir ces notions pourra se rapporter aux travaux de Carey et Robison [CR81], Carey [Car89] et Benhamou [BB89].

Accés aux données environnementales

Les données qui nous sont fournis par l'IRD sont sous la forme d'images PNG ou sous la forme de données ASCII. De plus, elles ne décrivent par l'environnement avec la même résolution et, bien entendu, elle ne représentent pas les même données. Pouvoir accéder de manière simple à ces données était une véritable gageure pour notre travail. Or, M. Desruisseaux de la Maison de la Télédétection de Montpellier a développé un outil logiciel qui permet de réaliser ce travail [DSD00, Des]. Avec son accord, nous avons utilisé ce logiciel, tout en modifiant la base de données afin qu'elle soit plus conforme à nos besoins. La base de données a été développée sous MySQL [MyS] car ce SGBD présente l'avantage d'être un produit libre et donc celà n'imposait pas l'achat d'une licence. Enfin, la caractéristique principale de MySQL est sa rapidité et c'est justement ce dont nous

[CR81] F.G. CAREY et B.H. ROBISON. – Daily patterns in the activities of swordfish xiphias gladius observed by acoustic telemetry. *Fishery Bulletin*, vol. 79, n° 2, 1981, pp. 277–292.

[Car89] F.G. CAREY. – Further acoustic telemetry observations of swordfish. *In : Proceedings of the 2nd International Billfish Symposium, Planning the Future of Billfishes, Research and Management in the 90s and Beyond,*, éd. par Stroud R.H. pp. 103–122. – National Coalition for Marine Conservation, Inc., Marine Recreational Fisheries Savannah, Georgia, USA, Kalia-Kona, Hawaii, USA, 1989.

[BB89] S. BENHAMOU et P. BOVET. – How animals use their environment: a new look at kinesis. *Animal Behaviour*, vol. 38, 1989, pp. 375–383.

avions besoin.

Afin de pouvoir organiser les données et y accéder facilement, M. Desruisseaux à mis en place le schéma relationnel présenté dans la figure 6.11.

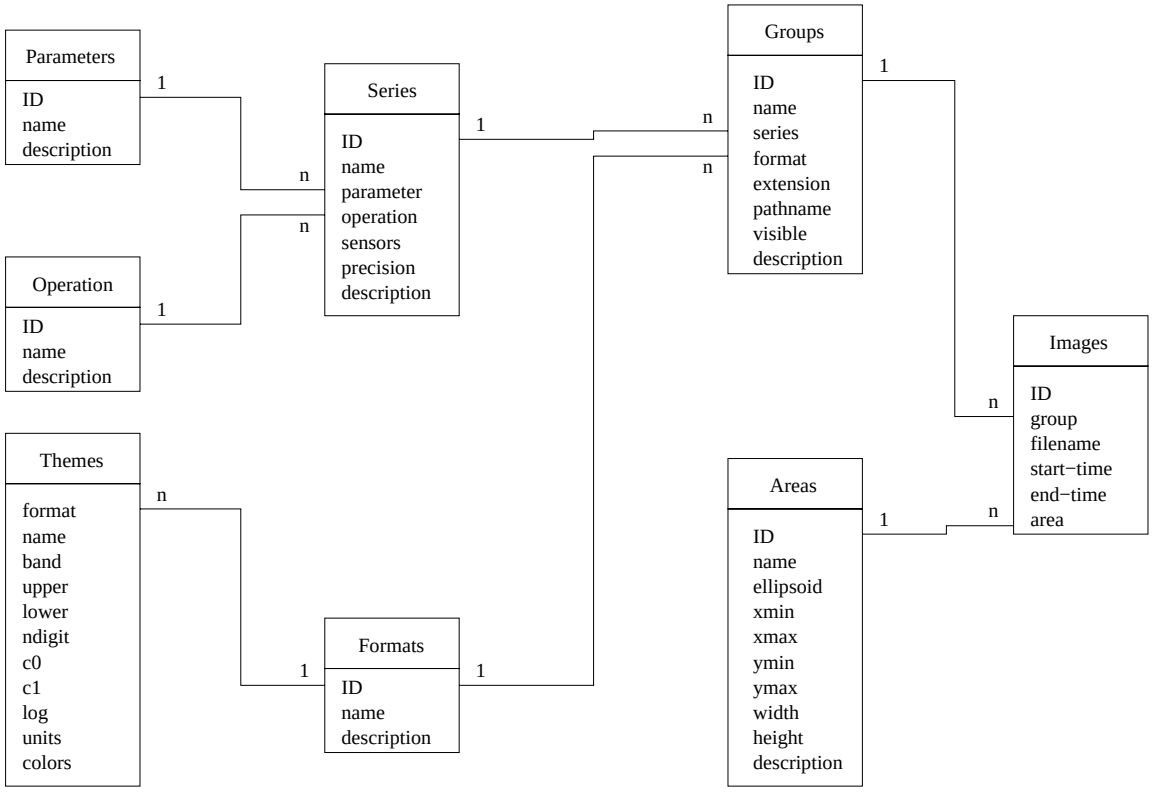


FIG. 6.11 – *Le modèle relationnel lié à l'accés aux images satellites*

La requête exécutée pour retrouver une valeur dans une image de la base est la suivante :

```
SELECT Groups.series, Groups.pathname, Images.filename, Groups.extension,
Images.starttime, Images.endtime, Areas.xmin, Areas.xmax, Areas.ymin,
Areas.ymax, Areas.width, Areas.height, Groups.format
FROM (Images INNER JOIN Areas ON Images.area \= Areas.ID)
INNER JOIN Groups ON Images.grp \= Groups.ID
WHERE (Groups.visible\='Y') AND
((xmax>? AND xmin<? AND ymax>? AND ymin<?)
AND (endtime>? AND starttime<?))
AND (series =?)
ORDER BY endtime;
```

Concrètement, chaque agent possède une connexion sur la base de données environnementale MySQL et à chaque pas de simulation, les environnements reliés à l'agent va exécuter les requêtes nécessaires en fonction des capteurs qui sont dans l'environnement.

C'est à dire que chaque agent va, à chaque pas de simulation exécuter la requête ci-dessus pour obtenir les informations nécessaires.

Le simulateur et les resultats de simulation

Le simulateur de MUFFINS est entièrement configurable par le biais d'un fichier de configuration. De cette manière, on peut lancer très facilement une série de simulations, sans interface graphique, sans avoir à intervenir. Le fichier de configuration est de la forme suivante :

```
# Fichier de configuration pour Mufins
# ---

# Agents initial positions (one file)
# ---

positions file = positions.txt

# The type of behavior
# list of behaviors:
# - mem
# - valueMax
# - valueMaxAtDistance
# - multi
# - bilat
# ---

behavior = bilat

# The depth of the exploration behavior by step (number of pixels)
# ---


prof = 15

# Beginning of the simulation (a date with the following format:
# DD/MM/YYYY:HH:MM:SS)
# ---

start time = 10/04/1999:18:30:00

# End of the simulation (a date with the following format:
# DD/MM/YYY:HH:MM:SS)
```

```
# ---

end time = 10/05/1999:16:44:00

# Number of days of simulation (integer)
# ---

days = 30

# Output parameters file
# ---

output parameters file = outparam.out

# Simulation Preferences file
# ---

# preference file = default.pref

# Parameters used in the simulation
# The first parameter is the most important in behavior
# The second is less important the first one, and so on...
# ---

parameters = SST, Vorti, Chl-a

# Geographic Area (to limit the size of the image to be loaded)
# ---

#geographic area =

# Database connection parameters
# ---

host name = bidi
domain name = univ.run
user name = ird
database name = seas3
time zone = Indian/Reunion
```

Les figures 6.12 page ci-contre, 6.13 page 142, 6.14 page 143, 6.15 page 144, 6.16 page 145, 6.17 page 146, 6.18 page 147, 6.19 page 148, 6.20 page 149, 6.21 page 150 sont des captures d'écran de MUFFINS et des résulats de simulation fournis pas MUFFINS

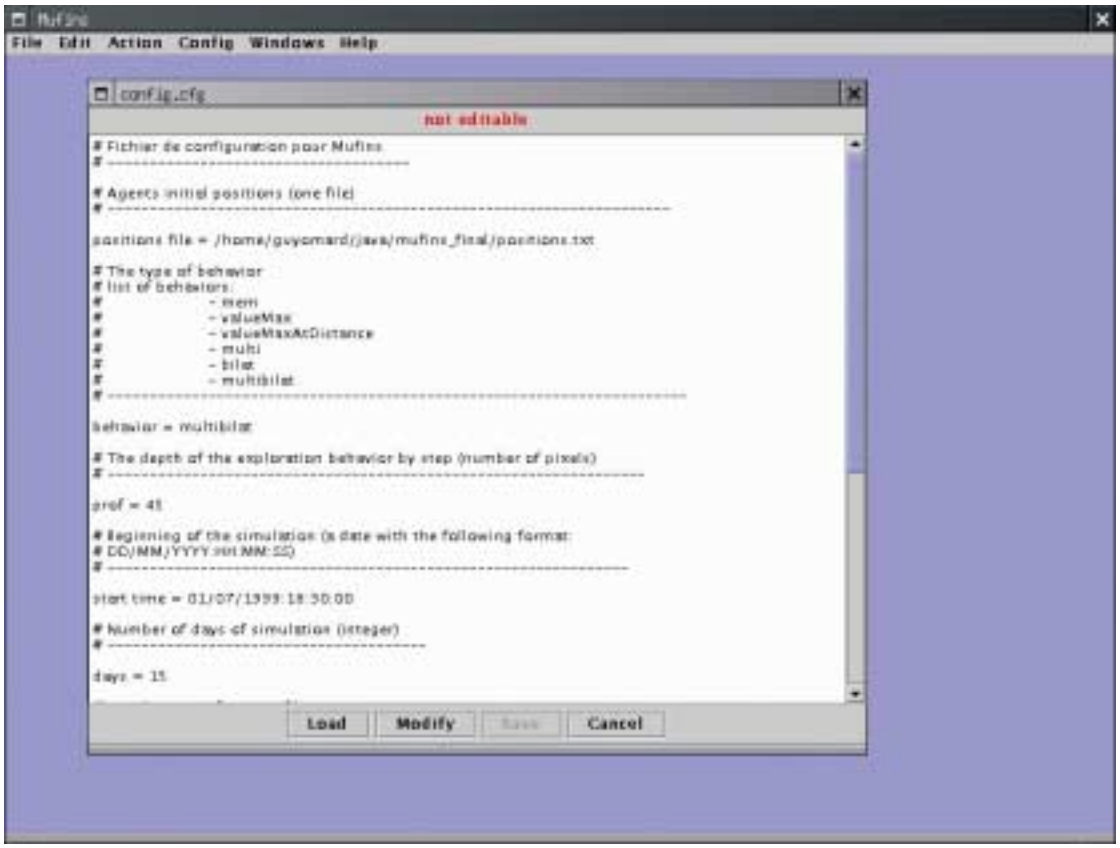


FIG. 6.12 – MUFFINS *avec son éditeur de configuration*

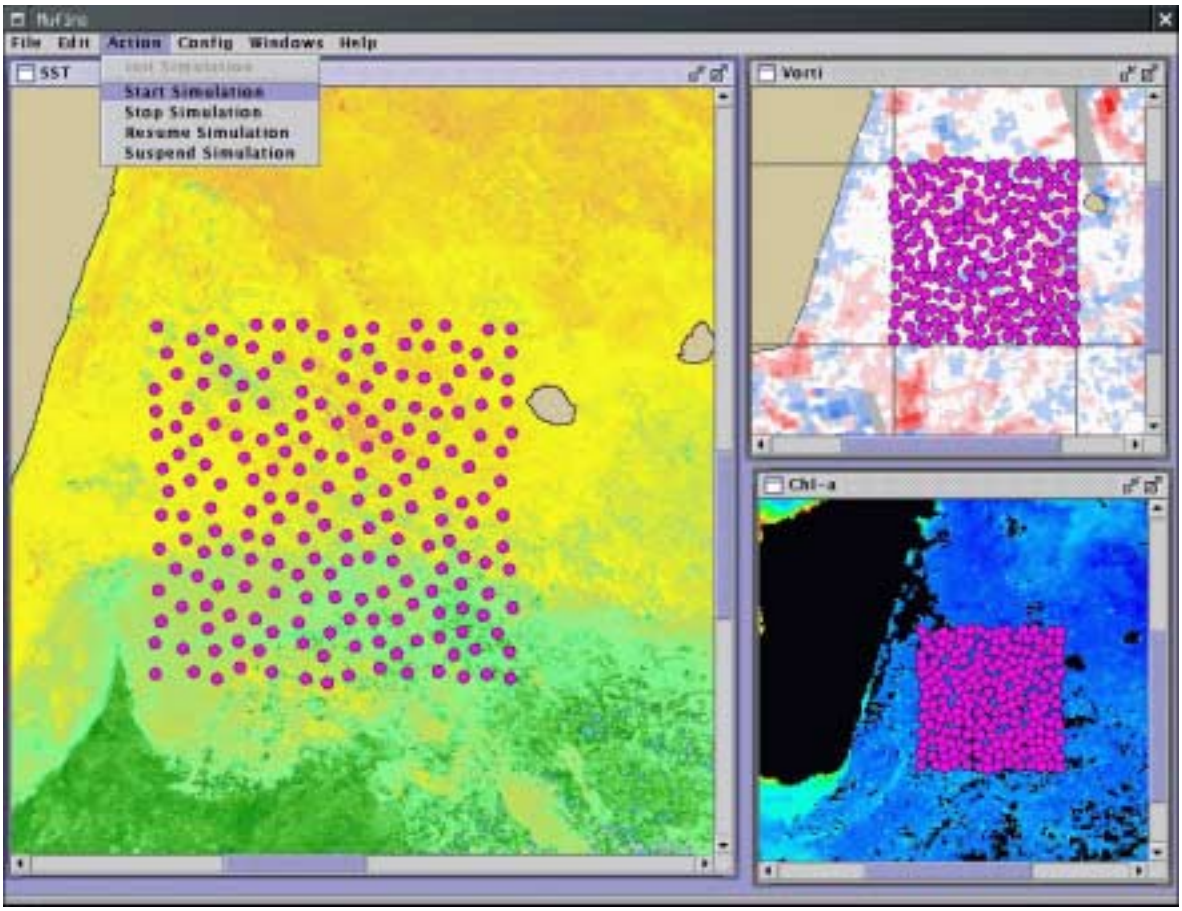


FIG. 6.13 – *Une simulation dans sa configuration initiale*

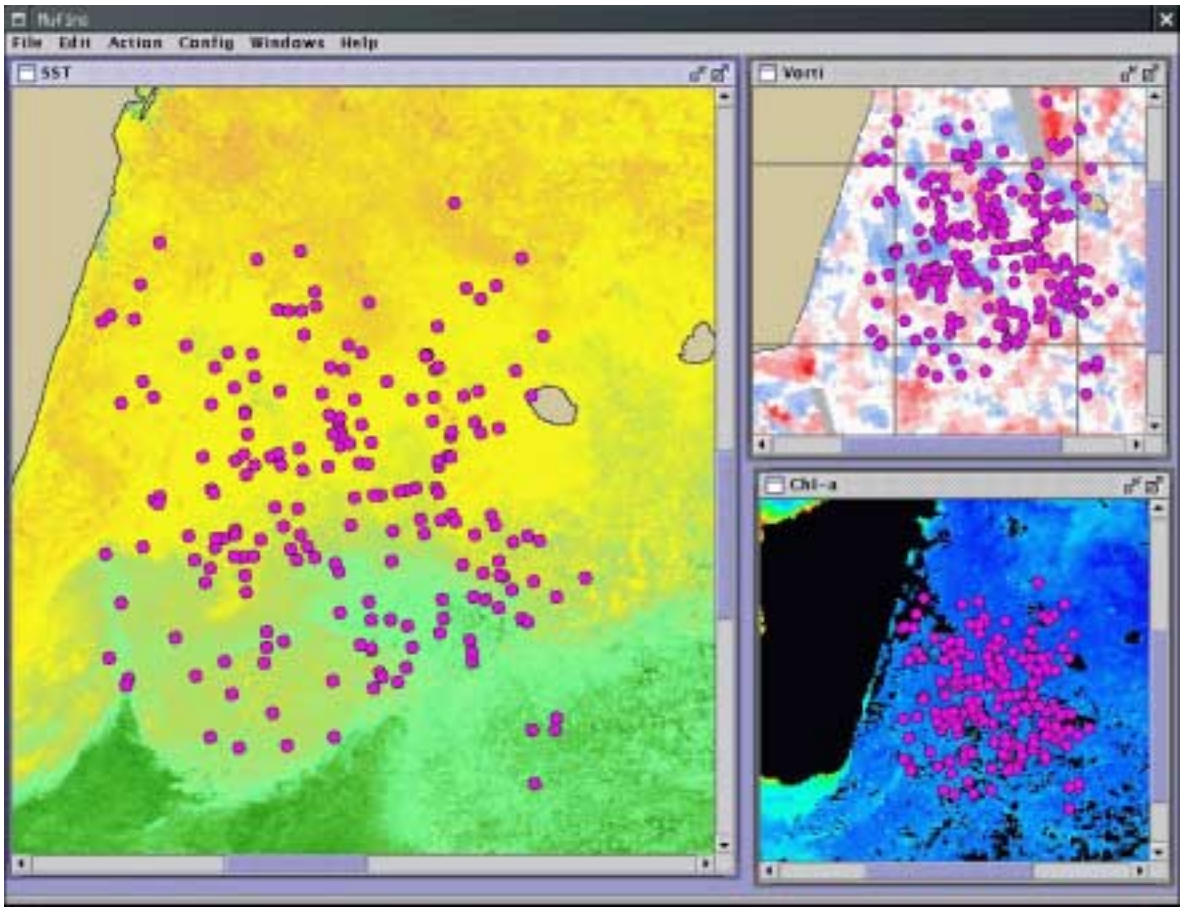


FIG. 6.14 – *Une simulation dans une configuratin finale*

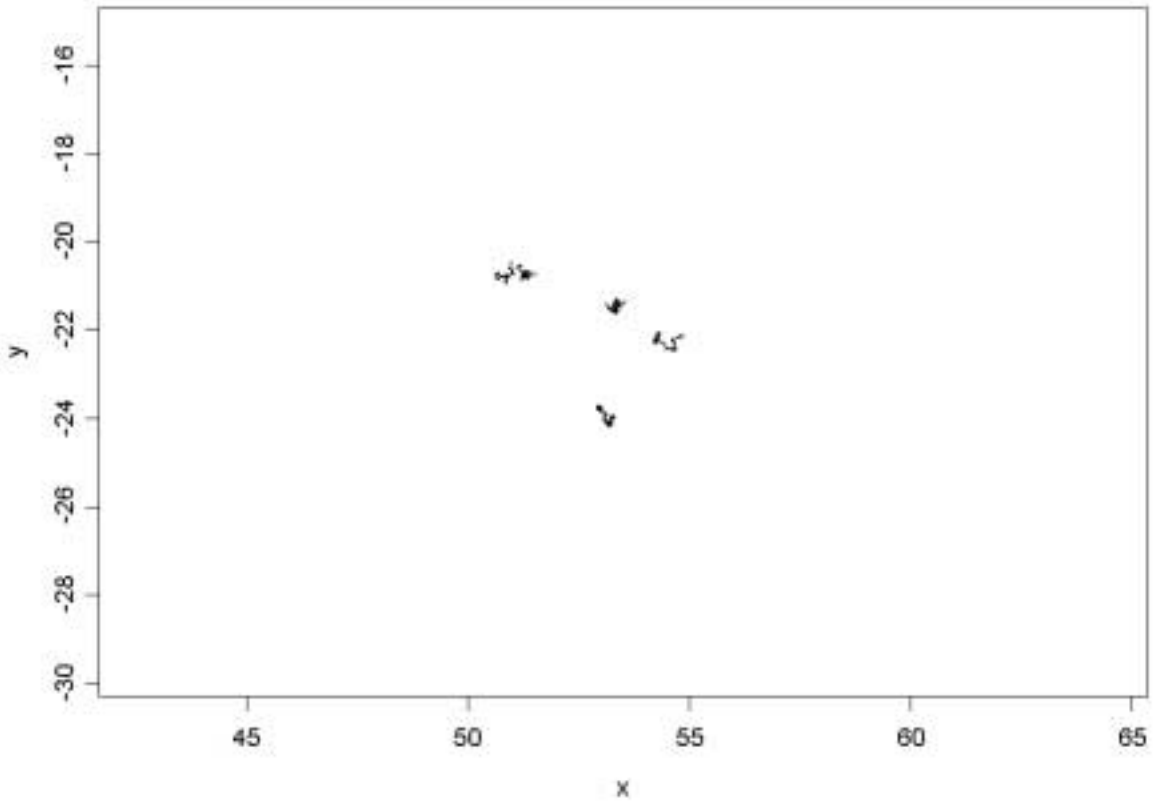


FIG. 6.15 – *Trajectoires d'espadons avec l'approche maximisation du gradient sur une profondeur de 5 pixels*

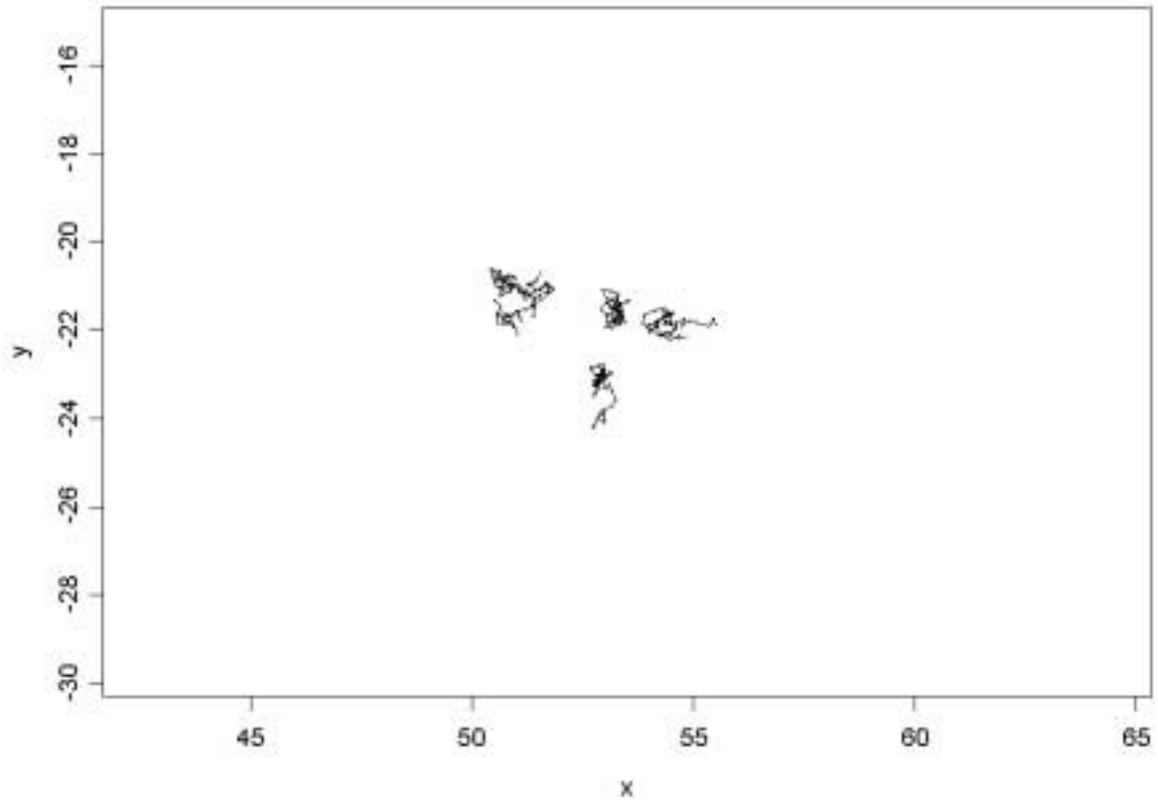


FIG. 6.16 – *Trajectoires d'espadons avec l'approche maximisation du gradient sur une profondeur de 15 pixels*

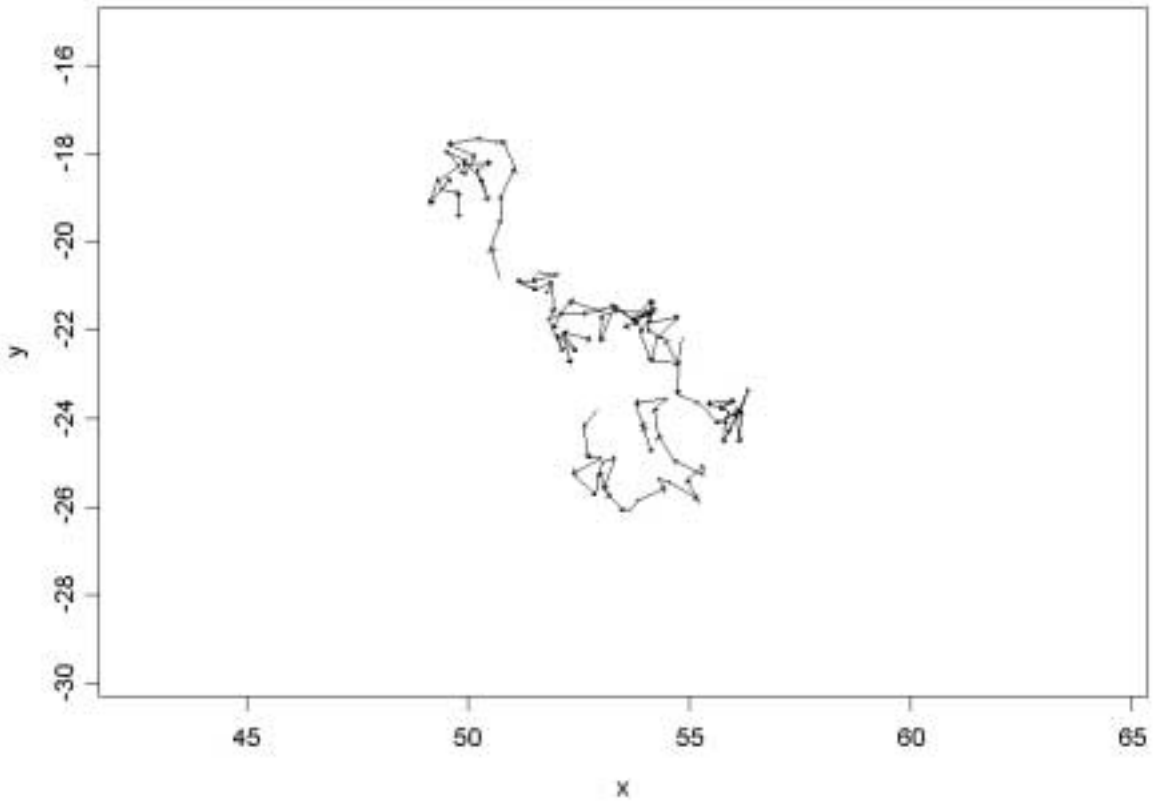


FIG. 6.17 – *Trajectoires d'espadons avec l'approche maximisation du gradient sur une profondeur de 45 pixels*

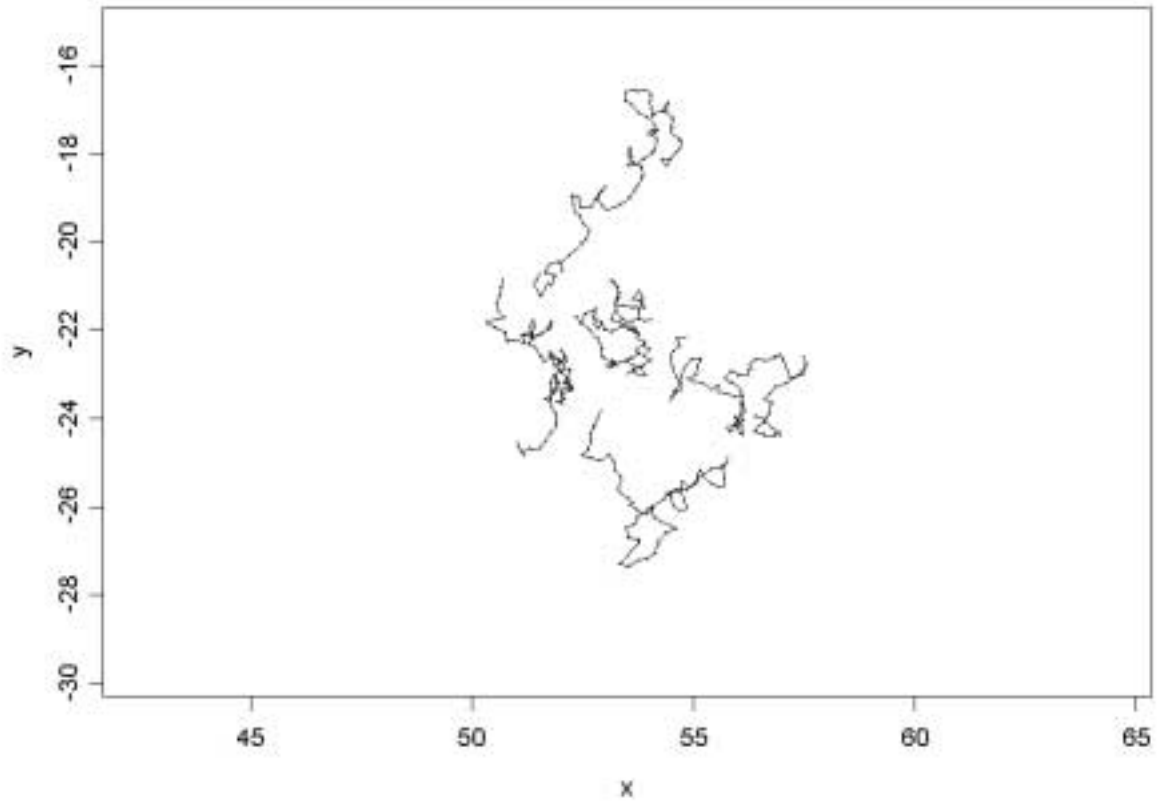


FIG. 6.18 – *Trajectoires d'espadons avec l'approche bilatérale sur une profondeur de 15 pixels*

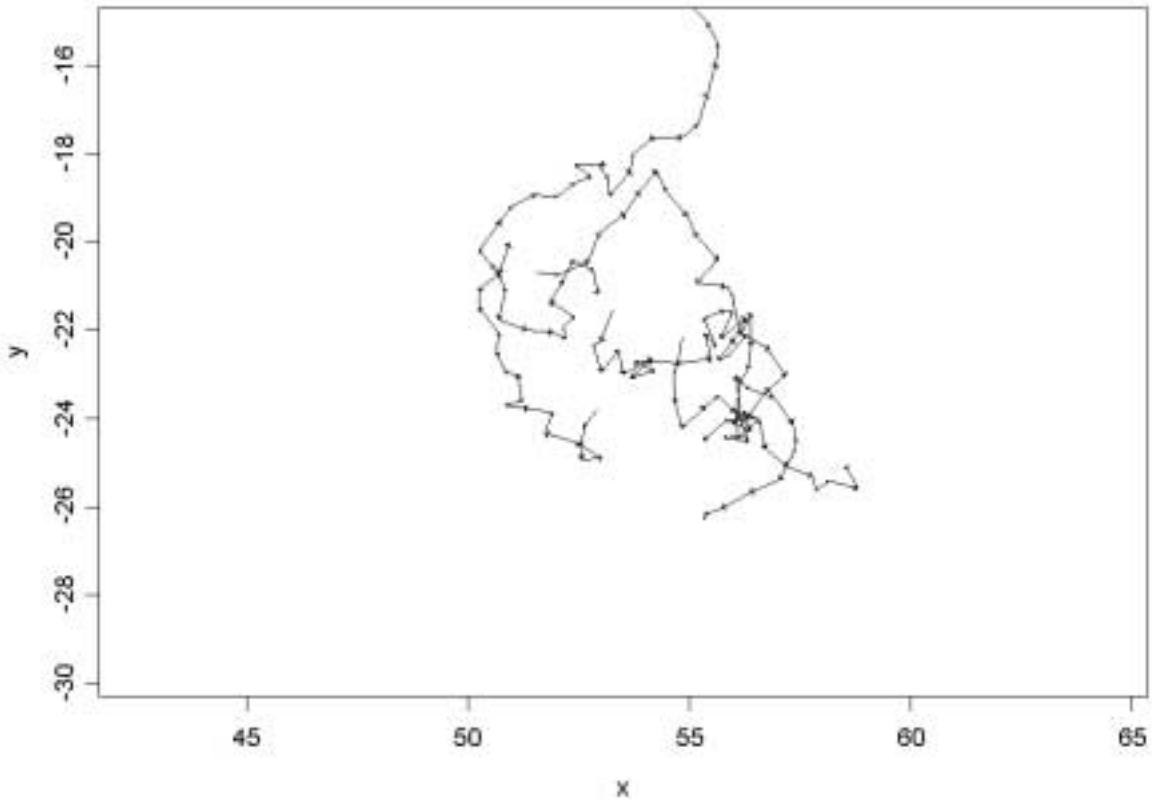


FIG. 6.19 – *Trajectoires d'espadons avec l'approche bilatérale sur une profondeur de 45 pixels*

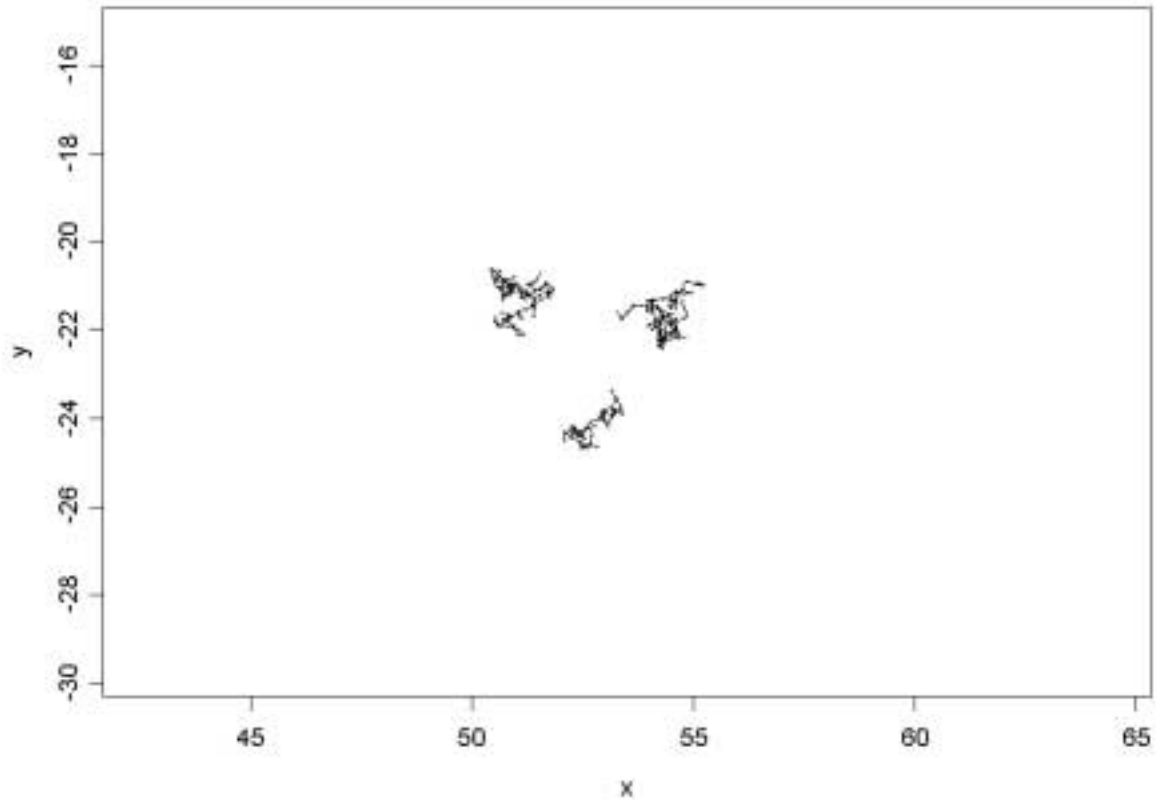


FIG. 6.20 – *Trajectoires d'espadons avec une approche mixant la maximisation du gradient et la marche aléatoire sur une profondeur de 15 pixels*

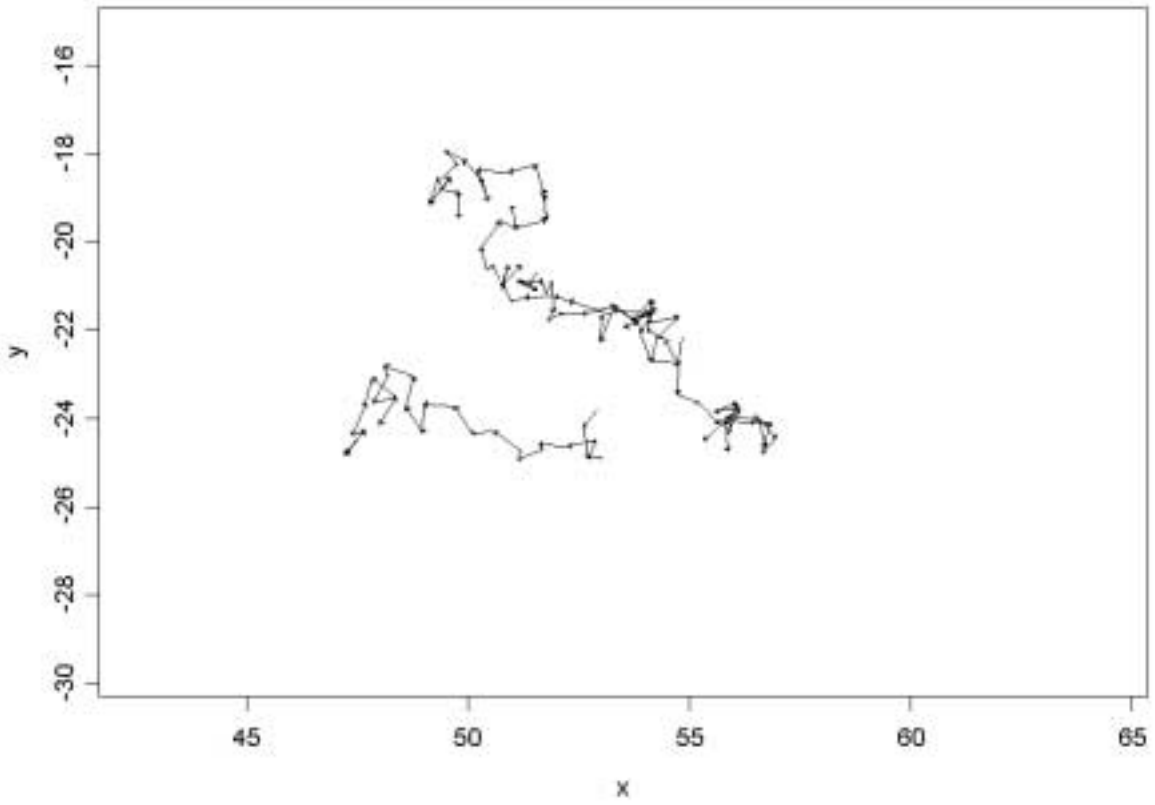


FIG. 6.21 – *Trajectoires d'espadons avec une approche mixant la maximisation du gradient et la marche aléatoire sur une profondeur de 45 pixels*

Conclusion et perspectives

Le point principal de cette contribution est bien sûr le modèle multi-environnements pour les agents. Cette approche nous permet d'appréhender de manière beaucoup plus facile la modélisation et la simulation de systèmes complexes naturels ou sociaux.

Mais, ce travail montre, également, comment cette approche multi-environnements facilite grandement la tâche de modélisation du système. Par cette modélisation, on s'astreint à garder cette décomposition pour tous les environnements et par conséquence pour toutes les entités du système qui sont en rapport avec ces environnements. Le fait de se baser sur les types des environnements entraîne naturellement ce travail de décomposition.

De plus, nous avons montré comment, à partir d'une modélisation somme toute assez classique des agents, on peut obtenir facilement un agent qui gère les environnements multiples. Cette décomposition est assez naturelle et ne nécessite pas une prise en main très longue. Nous espérons que, du fait de sa facilité, cette manière de voir les environnements au sein des systèmes multi-agents va être amenée à se développer.

Enfin, le dernier axe de ce travail est, bien sur, celui des implémentations et des simulations. Nous avons montré comment on peut distribuer les données d'un environnement dans le cas de simulations mono-environnementales. Cette première application nous a permis de mettre en évidence le fait que la technologie actuelle permet tout à fait de distribuer les données sur des serveurs distants sans problèmes majeurs. La deuxième application de ce travail a été de mettre en œuvre l'aspect multi-environnemental. Ce noyau générique est à présent fonctionnel et en s'appuyant celui-ci, l'application MUFFINS a été construite. Les résultats de simulation fournis sont encourageant et assez réalistes.

Les perspectives liées à ce travail sont nombreuses. Une première piste à explorer consiste à développer un modèle relationnel pour les SGBD qui puissent gérer des données qui ne proviennent pas uniquement d'images satellites. Le travail consistera donc à étendre et à généraliser ce modèle relationnel en s'inspirant des travaux effectués au niveau des SIG. La deuxième piste à explorer consiste à distribuer les données environnementales sur des machines localisées physiquement à des endroits différents car le volume des informations à traiter peut être éventuellement trop volumineux pour être stocké sur une seule machine. Ce travail pourra être possible une fois que le travail sur la première

piste sera terminé. Au niveau de la distribution sur le réseau, le travail sera facilité par le fait que MySQL gère des connexions réseaux, de ce fait, on pourra très facilement opérer cette distribution. La troisième piste de travail est, quant à elle plus formelle, elle consiste à définir un langage, une algèbre ou une grammaire qui permette d'exprimer facilement les sensibilités des instances environnementales aux modifications des données dans les environnements. Cet apport serait tout à fait judicieux dans la phase de conception du système et permettrait éventuellement de générer automatique le code associé à ces sensibilités.

Bibliographie

[AY84] D.L. De Angelis et G.T. Yeh. – *An Introduction to Modeling Migratory Behavior of Fishes*, chap. " ", pp. 445–469. – Plenum Press, New-York and London, 1984, *Mechanisms of Migration in Fishes.*

[BB83] Pierre Berlioux et Philippe Bizard. – *Algorithmique – Construction, preuve et évaluation de programmes.* – Dunod informatique, 1983.

[BB89] S. Benhamou et P. Bovet. – How animals use their environment : a new look at kinesis. *Animal Behaviour*, vol. 38, 1989, pp. 375–383.

[BBaPP98] François Bousquet, Innocent Bakam, Hubert Proton et Christophe Le Page. – Cormas : Common-pool ressources and multi-agents system. *In : Proceedings of the* 11^{th} *International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, éd. par A.P del Pobil et M. Ali, *Lecture Notes in Artificial Intelligence*, volume 1416. pp. 826–838. – Springer Verlag, juin 1998.

[BDT99] Eric Bonabeau, Marco Dorigo et Guy Theraulaz. – *From Natural to Artificial Swarm Intelligence.* – Oxford University Press, 1999.

[Boe86] B.W Boehm. – *A Spiral Model of Development and Enhancement.* – Software Engineering Note, 1986.

[Car89] F.G. Carey. – Further acoustic telemetry observations of swordfish. *In : Proceedings of the 2nd International Billfish Symposium, Planning the Future of Billfishes, Research and Management in the 90s and Beyond,*, éd. par Stroud R.H. pp. 103–122. – National Coalition for Marine Conservation, Inc., Marine Recreational Fisheries Savannah, Georgia, USA, Kalia-Kona, Hawaii, USA, 1989.

[CM97] Stéphane Calderoni et Pierre Marcenac. – Emergence of Earthquakes by MultiAgent Simulation. *In : Proceedings of the 11th European Simulation MultiConference*, éd. par Ali Riza Kaylan et Axel Lehmann. pp. 665–669. – Society for Computer Simulation, Istambul, Turkey, juin 1997.

[CM98] Stéphane Calderoni et Pierre Marcenac. – MUTANT : A multiagent toolkit for artificial life simulation. *In : Proceedings of the* 26^{th} *International Conference on Technologies of Object-Oriented Languages and Systems*, éd. par M. Singh, B. Meyer, J. Gil et R. Mitchell. pp. 218–229. – IEEE Computer Society Press, Santa Barbara, California, USA, août 1998.

[CM99] Stéphane CALDERONI et Pierre MARCENAC. – Mutant : a Genetic Learning System. *In : Proceedings of the* 12^{th} *Australian Joint Conference on Artificial Intelligence (AI-99)*, éd. par Norman FOO, *Lecture Notes in Artificial Intelligence*, volume 1747. – Springer, Berlin, décembre 1999.

[CMG98] Rémy COURDIER, Pierre MARCENAC et Sylvain GIROUX. – Application d'une méthodologie en spirale au développement d'une plate-forme multi-agents en smalltalk. *L'Objet – Hermes*, vol. 1, n° 4, 1998.

[CR81] F.G. CAREY et B.H. ROBISON. – Daily patterns in the activities of swordfish xiphias gladius observed by acoustic telemetry. *Fishery Bulletin*, vol. 79, n° 2, 1981, pp. 277–292.

[Dag94] Laurent DAGORN. – *Le comportement des thons tropicaux modélisé selon les principes de la vie artificielle.* – Rennes, Thèse de Doctorat, ENSAR, 1994.

[DCF93] A. DROGOUL, B. CORBARA et D. FRESNEAU. – Manta : New experimental results on the emergence of (artificial) ant societies. *In : Simulating Societies Symposium*, éd. par C. CASTELFRANCHI. – 1993.

[Des] Martin DESRUISSEAUX. – Seagis Home Page. – http://seagis.sourceforge.net/.

[Dic] Dictionnaire francophone universel en ligne. – http://www.francophonie.hachette-livre.fr/.

[DP99] Alexis DROGOUL et Sébastien PICAULT. – Microbes : vers de collectivités de robots socialement situés. *In : Actes des* $7^{ème}$ *Jounées Francophones en Intelligence Artificielle Distribuée et Systèmes Multi-Agents*, éd. par Marie-Pierre GLEIZES et Pierre MARCENAC. pp. 265–277. – Hermes, Saint Gilles, Ile de La Réunion, novembre 1999.

[Dro93] Alexis DROGOUL. – *De la Simulation Multi-Agents à la Résolution Collective de Problèmes.* – Paris, France, Thèse de Doctorat, Université Pierre et Marie Curie, 1993.

[DS96] J. DENÈCHE et F. SALGÉ. – *Les Systèmes d'Information Géographique.* – Presses Universitaires de France, 1996.

[DSD00] Martin DESRUISSEAUX, Robert SIRON et Hervé DEMARCQ. – *Regrouper les efforts de création d'outils géographiques Java : Proposition d'une avenue.* – Rapport de recherche, Observatoire du Saint Laurent – Canada, 2000.

[Fer94] Jacques FERBER. – Reactive distributed artificial intelligence : Principles and applications. *In : Sixth Generation Computer Technology*, éd. par G.M.P O'HARE et N.R. JENNINGS. pp. 287–314. – Waley-Interscience Publication, New York, USA, 1994.

[Fer95] Jacques FERBER. – *Les Systèmes Multi-Agents – Vers une intelligence collective.* – iia – InterEditions, 1995.

[FG86] R.T.T FORMAN et M. GODRON. – *Landscape Ecology.* – John Wiley and Sons, 1986.

[FG98] Jacques FERBER et Olivier GUTKNECHT. – A meta-model for the analysis and design of organizations in multi-agents systems. *In : Proceedings of the International Conference on Multi-Agents Systems – ICMAS'98.* pp. 128–135. – IEEE Computer Society Press, Paris, France, juillet 1998.

[Fla97] David FLANAGAN. – *Java in a nutshell (version 1.1).* – O'Reilly, 1997, 2nd édition.

[FTPD98] E. FIANYO, Jean-Pierre TREUIL, Edith PERRIER et Yves DEMAZEAU. – Multi-agent architecture integrating heterogeneous models for dynamical processes : The representation of time. *In : Proceedings of Multi-Agent Systems and Agent-Based Simulations – MABS'98.* – Paris, France, juin 1998.

[FTW84] D. FARMER, T. TOFFOLI et S. WOLFRAM. – *Cellular Automata.* – New York, 1984.

[GBH87] L. GASSER, C. BRAGANZA et N. HERMAN. – *MACE: A Flexible Testbed Distributed AI Research*, pp. 119–152. – Distributed Artificial Intelligence, 1987.

[GCC+98a] François GUERRIN, Rémy COURDIER, Stéphane CALDERONI, Jean-Marie PAILLAT, Jean-Christophe SOULIÉ et Jean-Dany VALLY. – Biomas : un modèle multi-agents pour aider à la gestion négociée d'effluents d'elevage. *In : Actes du* 1^{er} *Colloque sur les Modèles et Systèmes Multi-Agents pour la Gestion de l'Environnement et des Territoires.* – CEMAGREF, Clermont-Ferrand, France, octobre 1998.

[GCC+98b] François GUERRIN, Rémy COURDIER, Stéphane CALDERONI, Jean-Marie PAILLAT, Jean-Christophe SOULIÉ et Jean-Dany VALLY. – Conception d'un modèle multi-agents pour la gestion des effluents d'elevage à l'echelle d'une localité rurale. *In : Actes des* 6^{e} *Jounées Francophones en Intelligence Artificielle Distribuée et Systèmes Multi-Agents.* pp. 25–37. – Hermes, Nancy, France, novembre 1998.

[GF97] Olivier GUTKNECHT et Jacques FERBER. – *MadKit : Organizing Heterogeneity in a Platform for Multiple Multi-Agents Systems.* – Rapport de recherche n° LIRMM97189, Laboratoire d'Informatique, de Robotique et de Micro-électonique de Montpellier, décembre 1997.

[Gir93] Sylvain GIROUX. – *Agents et Systèmes : une nécessaire unité.* – Thèse de Doctorat, Université de Montréal, Canada, 1993.

[GL98] Vincent GINOT et Christophe LE PAGE. – Mobidyc, a generic multi-agents simulator for modeling populations dynamics. *In : Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, éd. par A.P del POBIL et M. ALI, *Lecture Notes in Artificial Intelligence*, volume 1416. pp. 805–814. – Springer Verlag, juin 1998.

[GMC+96] Sylvain GIROUX, Pierre MARCENAC, Stéphane CALDERONI, David GROSSER et Jean-Robert GRASSO. – A report of a case study with agents in simulation. *In : Proceedings of the* 1^{st} *International Conference on Pratical Applications of Intelligent Agents and MultiAgent Technology.* – London, UK, avril 1996.

[GSML99] David GUYOMARD, Jean-Christophe SOULIÉ, Pierre MARCENAC et Michel LARUE. – Mise en place d'un système multi-agents destiné à la simulation de dynamiques comportementales spatiales environnement/ressource, appliqué à l'espadon (xiphias gladius) dans le sud-ouest de l'océan indien. *In : Actes*

du 4ème Forum de l'Association Française d'Halieumétrie – Les Espaces de l'Halieutique. – Edition IRD, Rennes, juin 1999.

[HW94] A. HUTH et C. WISSEL. – The simulation of fish schools in comparison with experimental data. *Ecological Modelling*, vol. 75/76, 1994, pp. 135–145.

[Jav] The Source for Java™ Technology. – http://www.javasoft.com/.

[JO:82] Journal Officiel du 17 janvier 1982. – http://www.journal-officiel.gouv.fr/, 1982.

[Jon97] C. JONES. – *Geographical Information Systems and Computer Cartography.* – Addison Wesley, 1997, 1st édition.

[Lar00] *Dictionnaire encyclopédique Hachette – Edition 2000.* – Hachette Livres, 2000.

[Lee92] Jan Van LEEUWEN. – *Graphs Algorithms*, chap. 10, pp. 525–631. – Elsevier Sciences, 1992.

[Lem96a] Jean-Paul LEMAIRE. – *Les protocoles de réseaux.* – Hermès, 1996.

[Lem96b] Stéphane LEMAN. – *TREMMA : TRansfert d'Expertise avec un Modèle Multi-Agents. Un modèle multi-agents pour la représentation dynamique des connaissances et des raisonnements d'un apprenant.* – Saint Denis, Ile de La Réunion, Thèse de Doctorat, Université de La Réunion, 1996.

[Mar97] Pierre MARCENAC. – Modélisation de systèmes complexes par agents. *Technique et Sciences Informatiques – Hermes*, 1997.

[MBLA96] N. MINAR, R. BURKHART, C. LANGTON et M. ASKENAZI. – *The Swarm Simulation System : A Toolkit for Building Multi-Agent Simulations.* – Rapport de recherche, Santa Fe Institute, 1996.

[MC99] Pierre MARCENAC et Rémy COURDIER. – *Java Agent-Oriented Development Environment.* – John Wiley & Sons Books, 1999, *Journal of Object-Oriented Application Frameworks, M. Fayad.*

[Met00] Météo-France à La Réunion : Systèmes Dépressionnaires Tropicaux. – http://www.meteo.fr, 2000.

[MG98] Pierre MARCENAC et Sylvain GIROUX. – *GEAMAS : A Generic Architecture for Agent-Oriented Simulations of Complex Processes.* – International Journal of Applied Intelligence, Kluwer Academic Publishers, 1998.

[Mul97] Pierre-Alain MULLER. – *Modélisation objet avec UML.* – Eyrolles, 1997.

[MyS] MySql Home Page. – http://www.mysql.com/.

[NP97] Patrick NIEMEYER et Joshua PECK. – *Exploring Java (2nd Edition).* – O'Reilly, 1997, *The Java Series.*

[O'N98] Joseph O'NEIL. – *Java Beans Programming from the ground up.* – Osborne – McGraw-Hill, 1998.

[PC97] E. PERRIER et C. CAMBIER. – *Une Approche Multi-Agents pour simuler les Interactions entres les Acteurs Hétérogènes de l'infiltration et du Ruissellement d'eau sur une Surface de Sol*, chap. unknown. – Elsevier, 1997, *Tendances Nouvelles en Modélisation.*

[Pel01] *Données encyclopédiques.* – Hachette Multimédia / Hachette Livre, 2001.

[PG97] Christophe Le Page et Vincent Ginot. – Vers un simulateur générique de peuplements piscicoles. *In : Proceedings Des 5èmes Journées Francophones d'Intelligence Artificielle et Systèmes Multi-Agents – JFIADSMA'97*, éd. par J. Quinqueton, M.C Thomas et B. Trousse. – Hermès, La Colle sur Loup, France, avril 1997.

[Pop93] Ion Popescu. – *Construction of Delaunay and Voronoï Diagram. Efficient Sequential and Parallel Implementation.* – Rapport de recherche n° 93-53, Johannes Kepler University, Linz, Austria, RISC-Linz, 1993.

[RBEL91] J. Rumbaugh, M. Blaha, F. Eddy et W. Lorensen. – *Object-Oriented Modeling and Design.* – London, England, Prentice All International, 1991.

[RG89] D. Robson et A. Goldberg. – *Smalltalk-80, The Language.* – Addison Wesley, Publishing Company, 1989.

[SM99] Jean-Christophe Soulié et Pierre Marcenac. – Environmental simulations using multiagent systems. *In : Proceedings of the IEEE International Conference on Information, Intelligence and Systems – ICIIS'99.* – IEEE Computer Society Press, Washington, DC, USA, novembre 1999.

[SMCC98] Jean-Christophe Soulié, Pierre Marcenac, Stéphane Calderoni et Rémy Courdier. – GEAMAS v2.0 : An object oriented platform for complex systems simulations. *In : Proceedings of the* 26^{th} *International Conference on Technology of Object-Oriented Languages and Systems – TOOLS USA'98*, éd. par M. Singh, B. Meyer, J. Gil et R. Mitchell. pp. 230–242. – IEEE Computer Society Press, Santa Barbara, California, USA, août 1998.

[Sou97] Jean-Christophe Soulié. – *Conception et Implantation d'une Plate-Forme d'Agents en Java.* – Thèse de 3ème cycle, Laboratoire d'Informatique d'Avignon, Université d'Avignon et des Pays de Vaucluse, 1997.

[VC98] Jean-Dany Vally et Rémy Courdier. – A Conceptual "Role-Centered" Model for Design of Multi-Agents Systems. *In : Proceedings of First Pacific Rim International Workshop on Multi-Agents – PRIMA'98*, éd. par Toru Ishida, *Lecture Notes in Artificial Intelligence*, volume 1599. pp. 33–46. – Springer Verlag, Berlin, novembre 1998.

[W3C] The World Wide Web Consortium. – http ://www.w3c.org/.

[WJ95] M. Wooldridge et N.R Jennings. – *Intelligent Agents : Theory and Practice*, chap. unkown. – Knowledge Engineering Review, 1995.

[ZF93] K. Zeghal et J. Ferber. – A coordinated collision avoidance system. *In : Proceedings of the European Simulation Multiconference.* – Lyon, France, 1993.

Résumé

Dans le cadre de la vie artificielle, on assiste à un essor de l'utilisation des systèmes multi-agents. Ceux-ci permettent de modéliser et de simuler des phénomènes complexes ou des phénomènes sociaux qui ne peuvent pas être facilement modélisés à l'heure actuelle avec des approches classiques. La problématique développée dans cette thèse découle d'une question toute simple : « comment modéliser et simuler des agents qui évoluent dans plusieurs environnements à la fois ? ». Pour ce faire, il a été construit un modèle d'agent qui permet, dans un premier temps, de modéliser et d'étudier les interactions entre l'environnement et l'agent. Dans un deuxième temps, ce premier modèle est étendu afin de pouvoir utiliser plusieurs environnements simultanément. De ce nouveau modèle découlent un nombre de problèmes à régler : gestion du temps, gestion de l'intégrité des données dans les environnements et répartition des effecteurs et capteurs. Nous montrons comment ces divers problèmes sont résolus par l'adjonction de nouvelle entités dans notre modèle. Afin de mettre en œuvre l'approche multi-environnements, une plate-forme de simulation a été construite. Elle concerne la simulation de déplacement d'espadons dans la zone sud de l'océan Indien à l'aide d'images satellites.

Mots-clés: Systèmes Multi-Agents, Environnement, Modélisation, Simulation

Abstract

In the framework of the artificial life, multiagent systems are more and more used. They allow to model et simulate natural complex phenomenon and social phenomenon. The work presented in this thesis has been defined by a simple question: "how to model and simulate agents that evolves in multi environments at the same time?". First, we have defined an agent model that allows to model and study interactions between the environment and the agent. Second, the first model has been extended in order to take into account multiple environments. We show how we manage the time, the data integrity and the effectors and captors repartition on these environments. Finally, we illustrate this architecture with a simultion of swordfish motion in the Indian Ocean thanks to satellites maps.

Keywords: Multiagent Systems, Environment, Modelling, Simulation