Université de Caen – Basse-Normandie

UFR sciences

École doctorale simem

Co-tutelle de thèse
entre
l'Université de Caen – Basse-Normandie (France)
et
l'Université de Neuchâtel (Suisse)

(arrêté du 18 janvier 1994)

Thèse

présentée par

M. Sylvain SAUVAGE

et soutenue

le **9 octobre 2003**

pour l'obtention du

Doctorat de l'université de Caen – Basse Normandie

spécialité informatique

(arrêté du 25 avril 2002)

Conception de systèmes multi-agents : un thésaurus de motifs orientés agent

Composition du jury :

Directeurs :	M^me^ Anne NICOLLE	Professeur, université de Caen
	M. Jean-Pierre MÜLLER	Professeur, université de Neuchâtel Cadre de recherche, cirad, Montpellier
Rapporteurs :	M^me^ Danielle BOULANGER	Professeur, université de Lyon 3
	M. Pierre-Jean ÉRARD	Professeur, université de Neuchâtel

Université de Caen/Basse-Normandie
UFR sciences
École doctorale simem

Co-tutelle de thèse
entre
l'Université de Caen/Basse-Normandie (France)
et
l'Université de Neuchâtel (Suisse)
(arrêté du 18 janvier 1994)

Thèse

présentée par

M. Sylvain SAUVAGE

et soutenue

le **9 octobre 2003**

pour l'obtention du

Doctorat de l'université de Caen – Basse Normandie

spécialité informatique

(arrêté du 25 avril 2002)

Conception de systèmes multi-agents : un thésaurus de motifs orientés agent

Composition du jury :

Directeurs :	M^me^ Anne NICOLLE	Professeur, université de Caen
	M. Jean-Pierre MÜLLER	Professeur, université de Neuchâtel Cadre de recherche, cirad, Montpellier
Rapporteurs :	M^me^ Danielle BOULANGER	Professeur, université de Lyon 3
	M. Pierre-Jean ÉRARD	Professeur, université de Neuchâtel

Ce mémoire a été composé avec l'aide de LaTeX 2_ε et gnu–Emacs sur un système debian gnu–Linux.

Sommaire

Remerciements

Je tiens à remercier ici tous ceux qui ont participé à l'avancement de mes travaux, et notamment :

M^me^ Anne Nicolle, professeur à l'université de Caen, pour m'avoir initié à l'intelligence artificielle et particulièrement aux systèmes multi-agents, pour sa direction et ses conseils experts.

M. Jean-Pierre Müller, professeur à l'université de Neuchâtel et cadre de recherche au cirad, pour sa co-direction éclairée, surtout malgré les distances, et ses idées toujours innovantes.

M. Bernard Morand, maître de conférences à l'IUT de Caen, avec qui j'ai eu plusieurs discussions très enrichissantes, sur le génie logiciel en général et sur les motifs de conception en particulier.

Je tiens particulièrement à remercier M^me^ Danielle Boulanger, professeur à l'université Jean Moulin – Lyon 3, et M. Pierre-Jean Érard, professeur à l'université de Neuchâtel, pour avoir accepté d'être les rapporteurs de ce mémoire.

Avant propos :

de l'interaction

Simplement additionnés ou juxtaposés, les atomes ne font pas encore la Matière.
Une mystérieuse identité les englobe et les cimente,
à laquelle notre esprit se heurte, mais est bien forcé finalement de céder.

— Pierre THEILHARD DE CHARDIN, *in* Le Phénomène humain (1955, p. 31)

En réalité, la manière correcte d'aborder l'étude de la structure qui relie,
c'est de se dire qu'elle est, primordialement (quel que soit ce que l'on met derrière ce mot),
une danse d'éléments en interaction, et que c'est seulement de façon seconde qu'elle se trouve
restreinte par diverses sortes de contraintes physiques, et par celles qu'imposent les différents
organismes, chacun de la façon qui lui est caractéristique.

— Gregory BATESON, *in* La Nature et la pensée (1984, p. 21)

La « structure qui relie », comme l'appelle G. Bateson, est le ciment de cette « mystérieuse identité » dont parlait Theilhard de Chardin. C'est ce qui nous fait souvent dire que le tout est plus que la somme de ses parties. C'est l'interaction.

Dans le domaine de l'intelligence artificielle, Aaron Sloman a mis en avant l'inadéquation de modèle computationnel de la machine de Turing, [Sloman]. Peter Wegner a quant à lui démontré l'intérêt de l'interaction face aux algorithmes, [Wegner 96]. Il expose en somme le gain réalisé par l'utilisation d'une approche interactionnelle, amorcée par la programmation objet, face à une approche purement fonctionnelle pour la description et le développement de systèmes d'information.

Or l'interaction est le fondement du paradigme agent. C'est aussi ce que nous étudions.

Introduction

Les sciences n'essaient pas d'expliquer ;
c'est tout juste si elles tentent d'interpréter ;
elles font essentiellement des modèles.
Par modèle, on entend une construction mathématique qui,
à l'aide de certaines interprétations verbales, décrit les phénomènes observés.
La justification d'une telle construction mathématique réside uniquement et précisément dans le fait qu'elle est censée fonctionner.

— John von NEUMANN

L'étude des systèmes multi-agents (SMA) est une discipline expérimentale de cette science expérimentale qu'est l'intelligence artificielle, [Simon 95]. Et notre objectif, lors du travail qui a mené à ce mémoire, a été de trouver un moyen pratique d'aide à la conception de SMA. Ce mémoire présentera donc le résultat de nos recherches.

Notre objectif est aussi d'aider à la conception de systèmes multi-agents en induisant une émergence forte — au sens exprimé par Jean-Michel Sauvage dans [Sauvage 02a] — au sein du processus même de conception. En permettant une meilleure compréhension de ce processus, le concepteur profiterait d'une
boucle rétroactive sur celui-ci qui lui permettrait de mieux le maîtriser. Comme le soulignaient Anne Nicolle et Bernard Morand, [Nicolle & Morand 93], notre souhait est :

> *« [...] que le concepteur garde à l'esprit qu'il est observateur et donc qu'il fait partie de l'objet, du processus et du produit. »*

—★—

Pour étudier la conception de systèmes multi-agents, il nous faut faire un double bilan : celui des SMA d'une part, et celui des techniques de conception actuelles d'autre part. Avant d'aborder cet état de l'art du domaine de la conception des systèmes multi-agents, nous allons poser quelques définitions et quelques concepts présentant notre point de vue sur les SMA et notre but, aider à leur conception.

Après quelques éclaircissements sur ce que nous appelons la conception au sens du génie logiciel, nous reviendrons rapidement sur les caractéristiques des

systèmes multi-agents et de leurs applications. Nous finirons par nos idées sur la conception de SMA.

1. De la conception et du génie logiciel

Note de vocabulaire : Le terme de conception, même réduit au domaine du génie logiciel, pose un léger problème : il désigne à la fois l'ensemble de l'activité de création d'un programme et une partie de celle-ci, qui concerne la modélisation conceptuelle — c'est le mot — qui vient après l'analyse du problème et qui précède la mise en œuvre (ou « implémentation »). En général nous utiliserons la locution de « conception de système » ou d'« analyse–conception » pour parler de l'ensemble de l'activité de conception du logiciel et le terme de « conception » sera réservé pour désigner la phase de modélisation conceptuelle, celle qui est la cible de nos efforts.

—★—

Le développement d'un système d'information est décomposé en plusieurs phases. Ce ne sont pas des étapes. Elles ne sont pas forcément séquentielles dans le sens où chaque méthode d'analyse–conception définit les processus conduisant le développement, [Silvestre & Verlhac]. Selon la norme iso 12207 (afnor Z 67–150), ces phases sont :

1. les objectifs du logiciel (haut niveau, informel) ;
2. la spécification des besoins (l'analyse fonctionnelle) ;
3. la conception (la modélisation conceptuelle) d'une solution ;
4. la programmation et les tests unitaires (la mise en œuvre) ;
5. l'intégration et les tests de qualification ;
6. l'installation ;
7. l'exploitation et la maintenance.

Les méthodes de conception concernent les phases deux, trois, quatre et cinq — on les renomme souvent l'analyse, la conception, la mise en œuvre et la validation — et parmi elles, la conception a une place prépondérante (d'où la métonymie dont nous avons parlé au premier paragraphe).

La phase de conception est la résolution du problème énoncé lors des précédentes phases. Une conception réussie doit satisfaire une spécification fonctionnelle (qui peut être informelle) issue des phases antérieures et en même temps se conformer aux limitations des environnements de mise en œuvre (réalisation) et d'implantation (utilisation).

La phase de conception procède en fait de ce que nous appelons une modélisation conceptuelle, c'est-à-dire modéliser le système d'information qui résoudra le problème en utilisant des concepts informatiques connus ou décrits pour l'occasion, des concepts génériques (comme les objets, les processus, certaines classes fréquentes si l'on utilise la programmation objet, etc.), des concepts domaine (*i. e.*, relatifs au domaine de l'application, comme celui de page ou de figure pour un logiciel de traitement de texte) et des concepts particuliers à l'application (*i. e.*, difficilement réutilisables sans modification dans une autre application).

C'est de la réussite de la modélisation, de son adéquation au problème et aux usagers, que provient la réussite du projet.

2. Des SMA

Pour appréhender les systèmes multi-agents, il faut tout d'abord prendre en compte la pluridisciplinarité du domaine. Les SMA descendent à la fois de l'intelligence artificielle et de la vie artificielle. Mais ils tirent aussi leurs sources et leurs inspirations de domaines techniques — comme les réseaux et les systèmes répartis —, et de domaines plus abstraits — comme les sciences humaines et sociales (psychologie, anthropologie, sociologie, linguistique, éthologie, etc.) et les sciences cognitives (*i. e.*, les sciences de la perception, de l'apprentissage, de la connaissance et du raisonnement ; en d'autres termes, les neuro-sciences et l'informatique, [Varela 96]).

Nous allons essayer ici de cerner le concept de système multi-agent d'abord d'un point de vue programmation, comme un paradigme de programmation, puis d'un point de vue plus systémique, comme un paradigme de modélisation.

2.1. Le paradigme agent : un paradigme de programmation parmi d'autres

La figure 1 représente une sorte de phylum conceptuel des paradigmes de programmation. Les flèches représentent l'idée, le concept ou la propriété qui permet de passer d'un paradigme de programmation à un autre. [1]

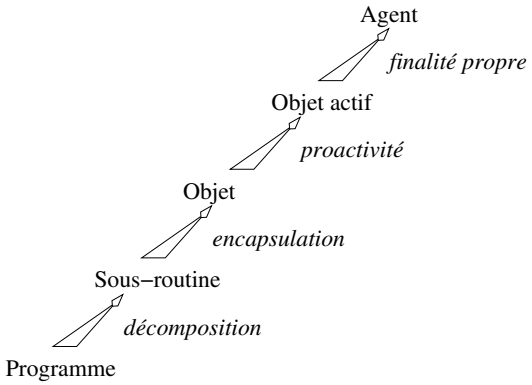


Fig. 1 – Liens entre les différents paradigmes

La décomposition

Le principe de décomposition — « diviser pour régner » — permet ainsi de passer du programme monolithique à un ensemble de sous-routines. C'est cette décomposition, chère aux cartésiens, qui permet de remarquer la réutilisation d'une même sous-routine en divers endroits du programme. Les bibliothèques

1. Il ne s'agit pas pour nous de refaire l'histoire de la conception des différents paradigmes mais d'opérer à un recoupement des propriétés de ces paradigmes pour les ordonner tels que l'on peut ordonner les ensembles $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{R}$ et $\mathbb{C}$, c'est-à-dire en ajoutant une propriété caractéristique à chaque pas. Il n'y a d'ailleurs pas qu'une seule façon de le faire pour ces ensembles mathématiques ; il n'y a sûrement pas qu'une seule façon de le faire pour les paradigmes de programmation. La façon dont nous le faisons ici nous semble convenir à nos propos tout en restant juste, à la fois d'un point de vue conceptuel que d'un point de vue historique ; on peut d'ailleurs trouver des démarches similaires dans [Briot & Gasser 98] ou [Lind 00].

de fonctions, statistiques par exemple, représentent une source de sous-routines réutilisables.

L'encapsulation

Si l'on encapsule les types de données avec les sous-routines (procédures, fonctions ou méthodes), le paradigme objet fait son apparition : les modules s'individualisent et gagnent une certaine indépendance, ils sont non seulement réutilisables tels quels dans le programme mais aussi, et facilement, dans d'autres programmes. Les interfaces graphiques sont une application idéale de ce principe, elles se prêtent bien à cette modélisation et profitent de tous les avantages de la modélisation objet (héritage, composition, délégation, etc.).

La proactivité

La proactivité fait passer des objets aux acteurs (ou objets actifs). La proactivité est le fait que les acteurs sont des objets qui ont leur propre fil d'exécution et un comportement propre. La proactivité nécessite donc le parallélisme. Celui-ci permet aux acteurs de continuer leurs calculs indépendamment les uns des autres. La proactivité subsume le parallélisme en laissant aux acteurs le « choix » du moment ou un message qui leur est adressé est effectivement traité : les messages peuvent être asynchrones.

L'on peut penser aux démons Unix comme exemple d'utilisation de cette propriété (notamment le serveur d'impression qui attend les travaux et qui, parallèlement, traite la communication avec les imprimantes et l'exécution de ces travaux).

La finalité propre

Dans cette continuité, le principal apport du paradigme des systèmes multi-agents est la présence de buts internes à l'agent. Les Gasser, dans [Briot & Gasser 98], appelle cette finalité propre *« structured persistent action »*[2], ce qui rend compte à la fois des aspects de proactivité et d'adaptabilité de l'agent. Ces buts, ou finalité, permettent à l'agent son adaptabilité et une plus grande autonomie.

Il est à noter que ce sont les avantages du parallélisme qui sont parfois utilisés pour qualifier la propriété d'autonomie des agents. Il nous semble pourtant que c'est ce que nous appelons *finalité* qui donne à l'agent toute son autonomie.[3]

L'on dit aussi que l'agent est *téléologique*, en cela non pas seulement qu'il est conçu dans certains buts mais qu'il les porte en lui-même et qu'il peut les manipuler. Il existe une gradation continue dans les possibilités de manipulation des buts qui rend parfois vague le terme d'agent. Dans les éco-simulation utilisant des SMA (branche des SMA issue principalement de la vie artificielle) par exemple, les possibilités de manipulation de ses buts internes par un agent sont souvent nulles car ces buts sont réduits à l'état d'une « simple » fonction de satisfaction. Par contre, l'on peut toujours parler d'agent car le développement d'une telle simulation — notamment dans la phase d'analyse mais aussi lors de la conception — profite des concepts et des techniques des systèmes multi-agents, [Batard 96].

2. Action persistante structurée.
3. Le terme d'*autonomie* est aussi parfois employé pour qualifier cette propriété.

Pour compléter la distinction entre acteur et agent, l'on pourrait aussi ajouter la propriété de localisation (les agents sont situés) et donc celle de mobilité. C'est d'ailleurs cette dernière propriété qui est mise en avant dans les applications des SMA au domaine de l'Internet.

Pour résumer la distinction entre objet, objet actif et agent, nous dirons que l'objet s'occupe du *comment* des services qu'il rend, l'objet actif du *comment* et du *quand* (asynchronicité) et l'agent, lui, s'occupe du *comment*, du *quand* et, surtout, du *pourquoi*.

Des objets

La *paternité* des objets est d'ailleurs confortée par le fait que les agents sont souvent réalisés grâce à la programmation orientée objet ou par le biais des acteurs, [Briot 89a, Gasser & Briot 92, Guessoum & Briot 98a, Agha & Jamali 99]. La proximité de ces paradigmes est assurément d'une aide précieuse ; bien que l'on puisse noter quelques divergences, comme le faisait justement remarquer Les Gasser, [Briot & Gasser 98], la comparaison acteur–agent est bénéfique, les deux points de vue se marient bien et s'appuient mutuellement, mais plusieurs acteurs ne forment pas un système multi-agent : la notion de coordination entre agents ne peut se limiter aux nécessités de synchronisation et de partage des ressources entre acteurs. Bien sûr, les deux domaines bénéficient des avances de l'un et de l'autre, ils représentent deux points de vues différents mais non antinomiques.

D'une autre façon, l'OMG[4], et, par elle, la communauté objet, adopte et intègre de plus en plus les concepts utilisés en SMA : *agent* et *rôle* en particulier. En effet, la souplesse du concept d'agent et la généricité du concept de rôle — un agent étant une entité du système et un rôle un comportement fixé que peuvent avoir différentes entités — permettent leur application à différents niveaux de granularité et permettent donc de spécifier des structures de conception à différents niveaux (cf. les motifs de conception ou les protocoles).

2.2. Système et agent

Un système multi-agent est un ensemble d'agents, évoluant dans un environnement, en interaction les uns avec les autres et avec l'environnement.

Cette définition met en avant les caractéristiques systémiques des SMA. Elle ne donne pas — contrairement à ce que l'on a vu dans les paragraphes précédents — de définition extensive de l'agent. Un agent est défini comme une entité en interaction (avec d'autres agents et avec l'environnement). Un agent est donc défini par les relations qu'il a avec les *autres.*

Cette définition souple permet de voir le SMA comme un système[5] dont les sous-systèmes sont des agents ou des groupes organisés d'agents. L'on obtient ainsi une récursivité dans le modèle système–agent. Cette récursivité est fortement employée dans différents travaux, notamment [Stinckwich 94] ou [Occello & Demazeau 97].

4. *Object Management Group,* organisation regroupant les entités intéressées par la programmation objet (notamment IBM, HP, Sun Microsystems, etc.), `http://www.omg.org`.
5. Au sens systémique pur.

2.3. Caractéristiques du paradigme agent

Comme nous l'avons déjà dit, la finalité des agents est leur caractéristique principale. Ces buts que l'agent doit réaliser demandent qu'il soit autonome, c'est pourquoi l'autonomie est une autre caractéristique du paradigme agent.

L'aspect systémique met en avant deux caractéristiques : le fait que l'agent, en tant que système, est situé dans un environnement, et le fait que les agents forment un système multi-agent.

La notion de système, et donc d'une pluralité d'agents autonomes ayant chacun leurs propres buts, implique les notions de coopération et de compétition. Avec elles, une communication entre les agents et, dans de nombreux cas, une capacité de négociation de l'agent sont nécessaires.

La grande différence entre la programmation fonctionnelle et la programmation orientée objet est qu'en plus de décrire la fonction du système, l'on décrit aussi — et en même temps — sa structure. Or, les trois dimensions d'un système sont sa structure, sa fonction et son cycle de vie (ou son histoire), [Simon 69]. Et, en décrivant les buts de l'agent, l'on lui permet d'être autonome, de gérer son cycle de vie ; l'on lui permet de répondre différemment suivant son « vécu ». Peut-être est-ce là ce qui fait des SMA un véritable paradigme de programmation en cela qu'ils ajoutent la troisième dimension systémique à la description du système.

2.4. Utilisations des SMA

Les différentes applications des systèmes multi-agents relèvent de quatre groupes :

- la résolution de problèmes par émergence ;
- la simulation ;
- le contrôle de systèmes complexes ;
- les environnements d'interaction homme–machine.

Résolution par émergence

L'émergence — dont une définition est donnée dans [Jean 97] — est une qualité intrinsèque des SMA, et des systèmes complexes en général. Nicholas Jennings *in* [Jennings 99] relève parmi les inconvénients des SMA que *« the unexpected individual and group behaviour is considerable »*[6], mais ce comportement social inattendu, émergent, peut servir à faire émerger une solution à un problème complexe à partir de comportements simples. Cette émergence est donc utilisée dans de nombreuses applications de résolution de problèmes (nous en parlerons plus en détail dans le motif *Marques* page 93).

Simulations

L'adéquation de la programmation objet aux simulations n'est plus à démontrer, la naissance même du paradigme objet venant pour partie de ce type d'applications, (cf. le nom même de l'ancêtre des langages de programmation

6. « le comportement social et individuel inattendu est considérable »

objet, simula). Toutefois, le manque d'autonomie des objets ne permet pas toujours de modéliser d'une façon suffisamment fidèle le phénomène que l'on veut simuler. C'est là que les caractéristiques d'autonomie, de proactivité, des agents interviennent.

Les simulations sont donc un autre domaine d'application des SMA, c'est souvent ces applications qui sont mises en avant, surtout lorsque l'on a une culture « vie artificielle ».

Les simulations servent aux experts du domaine (biologistes, sociologues, physiciens, etc.) de deux façons complémentaires. La première utilité est de pouvoir tester et valider les modèles qu'ils ont pu faire du phénomène complexe qu'ils étudient (*e. g.*, des modèles comportementaux de colonies d'insectes à ceux d'une foule dans un hall de gare). La seconde utilité est de pouvoir jouer sur des paramètres difficilement modifiables, voire non modifiables, dans les cas réels pour observer leurs influences.

La définition des agents par leurs relations permet d'appliquer des métaphores sociales à la conception du système, ceci permet là encore de coller plus fidèlement au phénomène à modéliser.

Contrôle

Les systèmes multi-agents sont aussi utilisés pour le contrôle ou le pilotage d'outils ou de composants immergés dans un environnement dynamique, tels les chaînes de production ou encore les robots. Ces systèmes complexes profitent de l'adaptabilité des agents qui les contrôlent pour réagir aux incertitudes de leur environnement (tant dans les perceptions que dans le résultat effectif de leurs actions). Ils profitent aussi de l'autonomie et des qualités de coopération des agents pour résister aux pannes (la faille d'une partie du système peut être compensée par le système).

Là encore, leur conception utilise des métaphores sociales pour la définition des interactions entre les agents. L'on peut aussi résoudre ce genre de problèmes par l'émergence : le système est équilibré par le comportement tropique mais fortement adaptatif des agents, [Müller 98].

Environnements d'IHM

Ces environnements d'interaction homme–machine — comme les EIAH (Environnements Intelligents pour l'Apprentissage Humain), les environnements de développement ou les applications d'assistanat (assistants bureautique ou sur l'Internet) —, avec leurs connotations fortement sociales et interactives, profitent aussi des métaphores sociales des systèmes multi-agents pour leur conception. Le caractère fortement composite et modulaire de tels environnements profite encore au paradigme agent.

Liens avec le génie logiciel

D'une autre façon, si l'on reprend la nomenclature utilisée en génie logiciel, [Printz] :

– les programmes de type S (Spécifié) étant ceux dont on a une spécification parfaitement définie (*i. e.*, un algorithme qu'il *ne reste plus* qu'à transcrire) ;

– les programmes de type P (Problème) résolvant une classe de problèmes avec incorporation de critères d'optimisation dépendant de l'usager (*e. g.*, un

compilateur qui peut produire un code robuste et facilement débogable ou un code hautement optimisé) ;

– les programmes de type E (Environnement) réagissant à des *stimuli* de l'environnement (utilisateur, capteurs, etc.) ;

les SMA s'appliquent par essence aux programmes de type E. Ils peuvent être appliqués à des programmes de type S ou P dans le but d'améliorer leur réactivité ou lorsque l'algorithme précis est trop coûteux, trop complexe ou encore inconnu (pour le type P) ; ils fonctionnent alors comme *approximants*.

Les simulations, les environnements d'interaction et le contrôle des systèmes complexes tombent dans le groupe des programmes de type E, la résolution de problèmes par émergence étant un programme de type S ou P.

3. De la conception de SMA

3.1. Des méthodes

Concevoir des SMA nécessite une méthode (ou plusieurs, une par type de SMA). Du fait de la proximité des paradigmes agent et objet, la première idée serait d'utiliser, à quelques adaptations près, les méthodes orientées objet. Mais cela n'est pas possible, comme le soulignait Alain Cardon, [Cardon 98], ou Nick Jennings dans [Jennings 99] :

– les objets ont une granularité beaucoup trop fine par rapport aux agents ;

– les interactions entre les objets sont trop rigidement définies (cf. la possibilité d'un agent de refuser ou de négocier une tâche) ;

– il n'y a pas de mécanisme capable de gérer les structures organisationnelles intrinsèques à la plupart des SMA.

Il est alors nécessaire de créer une méthode orientée agent. Nous verrons donc dans le chapitre suivant quelques exemples des méthodes agent actuelles. Nous verrons aussi qu'elles sont fondés sur des modèles particuliers ou des architectures particulières (modèles d'agent, d'environnement, d'interaction, d'organisation ou d'espace, architectures d'agent ou de communication, etc.).

3.2. Des outils

Concevoir des SMA nécessite aussi des outils pour aider à leur réalisation. Pour le moment, ces outils prennent forme dans des *plates-formes* agent. Nous verrons dans le chapitre suivant qu'il existe actuellement un grand nombre de telles plates-formes dans la communauté SMA. Nous verrons aussi que toutes les *plates-formes* agent ne sont pas à proprement parler des outils complets de mise en œuvre mais dont l'utilité s'échelonne de la bibliothèque de classes au *framework* (au sens de *proto-application*, cf. chap. I, section 3.2 page 19).

3.3. Idées directrices

Pour notre part, nous n'avons pas retenu ces différents moyens comme objet d'étude. Les méthodes fixent les modèles et contraignent donc le concepteur. Quant aux plates-formes, d'une part elles n'aident à la construction du SMA qu'à la fin du processus de conception, lors de la mise en œuvre, et, d'autre

part, — et bien qu'elles se veulent souvent généralistes — elles demandent aussi d'utiliser des modèles et des architectures particuliers et ne s'appliquent ainsi qu'à des classes de problèmes particulières.

La tâche que nous nous sommes fixée étant de trouver un moyen d'aider à la conception de systèmes multi-agents sans bloquer le concepteur dans des modèles ou des architectures par trop rigides, mais en le guidant dans ses choix, tout au long du processus de conception du système, de la modélisation à l'implantation, nous avons exploré les différentes techniques de génie logiciel qui permettent à la fois la réutilisabilité des connaissances et des expériences — en somme l'expertise de la communauté —et de couvrir au maximum les phases de la conception d'un SMA. Les motifs de conception sont la technique que nous avons choisie et que nous nous appliquons à étudier dans ce mémoire.

4. Approche utilisée

Joël de Rosnay note que « La complexité émerge de la simplicité partagée. C'est une des grandes lois de la nature », [de Rosnay 95, p. 27]. Or, comme le souligne la systémique — d'abord par [Simon 69], puis par [Le Moigne 90] en France —, il ne suffit pas de compliquer les méthodes utilisées sur les problèmes simples pour appréhender un phénomène complexe.

Ces systèmes complexes sont souvent distribués, les données et le contrôle nécessaires à la résolution du problème y sont répartis parmi les éléments du système. Ce sont aussi des systèmes ouverts, car ils demandent une adaptabilité à l'environnement et à leur extension.

Dans [Bateson 84], Gregory Bateson met en garde contre l'envie d'étudier les systèmes complexes par la logique ou par le quantitatif :

> *« Mais il n'existe pas aujourd'hui, en 1979, de méthode établie permettant de faire la description d'un tel enchevêtrement. Nous ne savons même pas par où commencer.*
>
> *» Il y a 50 ans, on aurait admis que la meilleur marche à suivre pour accomplir une telle tâche devait être soit logique, soit quantitative, soit les deux à la fois.*
>
> *» Mais nous allons voir, comme tout élève devrait le savoir, que la logique est précisément impropre à traiter des circuits récurrents sans engendrer des paradoxes, et que les quantités ne sont précisément pas le tissu de ces systèmes complexes communiquant entre eux. »*

C'est en conservant ces mises en garde à l'esprit que nous avons essayé d'étudier le problème de la conception des systèmes multi-agents. Car les SMA sont une approche adéquate des systèmes distribués et des systèmes ouverts. En effet, ils ont une vocation de parallélisme, de résistance aux pannes et aux interférences, d'adaptation et d'extensibilité incrémentale et, surtout, de prise en compte des interactions. Tous ces points sont nécessaires pour régler ce type de problèmes.

5. Contexte scientifique

Nos travaux s'inscrivent dans deux laboratoires au sein desquels nous les avons effectués. Le laboratoire greyc (Groupe de REcherche en Informatique,

Imagerie et Instrumentation de Caen) tout d'abord — nous y avons effectué la majeure partie de nos travaux — qui regroupe de nombreuses disciplines informatiques et qui participe à de nombreuses collaborations avec des disciplines variées, telles la linguistique, la géographie ou l'économie. L'IIUN (Institut d'Informatique de l'Université de Neuchâtel) ensuite — le laboratoire d'origine de notre co-directeur de thèse, Jean-Pierre MÜLLER — dans lequel nous avons effectué plusieurs séjours.

Nos travaux suivent donc la lignée de ceux du feu groupe Codissima du greyc — le premier groupe du laboratoire axé sur l'interaction et les SMA, originellement dirigé par Anne NICOLLE —, et parmi ces travaux, ceux de Benoît DURAND sur la modélisation de SMA dédiés à la simulation, ceux de Bernard MORAND sur l'analyse et la conception des systèmes d'information et ceux d'Anne NICOLLE sur la conception et l'interaction (homme–machine et, plus généralement, agent–agent).

Nos travaux s'insèrent aussi dans ceux du groupe de travail smile (Systèmes Multi-agents Interactifs, Logiques et Évolutifs), dirigé par François BOURDON, et qui a regroupé le pôle SMA du laboratoire greyc, et dans ceux de l'équipe island (Interaction Sémiotique : Langue, Diagrammes).

Les travaux de Jean-Pierre MÜLLER de l'IIUN nous ont aussi fortement guidé, notamment en ce qui concerne l'émergence et les méthodes de conception émergentes.

Bien sûr, les collaborations et les contacts avec d'autres laboratoires nous ont aussi inspiré, notamment à travers les groupes de travail Colline et asa[7] du GDR--PRCI3.

6. Démarche suivie

Dans l'optique des objectifs que nous venons de décrire, nous avons essayé de comprendre la conception des systèmes multi-agents en étudiant les différentes méthodes existantes, et, parallèlement, nous avons étudié le génie logiciel, plus particulièrement le génie logiciel objet et les techniques de réutilisation.

L'étude des génies logiciels agent et objet nous a permis de constater que le génie logiciel agent en est encore à ses balbutiements mais que son frère aîné, le génie logiciel objet, atteint, lui, sa maturité.

Le génie logiciel agent est certes jeune mais il possède déjà une grande masse d'expériences. Nous avons déduit qu'il fallait trouver une nouvelle formalisation de la connaissance agent si nous voulions aider au développement de ce GL agent.

La maturité du GL objet nous a conduit à envisager d'utiliser l'expérience acquise dans ce domaine pour l'appliquer à la formalisation des connaissances empiriques du paradigme agent.

L'étude des techniques de réutilisation, et notamment des motifs, nous a amené à penser que les motifs sont le médium idéal pour diffuser les concepts et les différents modèles utilisés dans la conception et la réalisation des SMA.

De cette démarche, provient le plan en deux parties de ce mémoire. La première partie nous permettra de décrire certains travaux sur la conception de SMA et sur les techniques de réutilisation. Dans la seconde nous exposerons le

7. Agent et Systèmes d'Agents.

corps de notre thèse — l'utilisation de motifs — à travers la description d'un thésaurus de motifs agent.

7. Plan de ce mémoire

—★—

Première partie : état de l'art

Premier chapitre

Dans le premier chapitre, nous aborderons l'aspect génie logiciel de l'état de l'art. Nous y définirons la réutilisabilité et exposerons quelques techniques de réutilisation utilisées en génie logiciel et dans le domaine des systèmes multi-agents en particulier. Nous aborderons notamment les concepts de composant, de *framework*, de motif et de protocole. Nous essaierons aussi de débroussailler les notions intervenant lors du passage d'un modèle à son utilisation, nous y verrons notamment la différence entre instanciation et paramétrage.

Chapitre II

Ce deuxième chapitre constituera un état de l'art des méthodes, des techniques et des outils actuellement mis à la disposition du concepteur de SMA. Nous ferons une description sommaire des différents travaux réalisés en nous fondant sur une grille d'analyse de la réutilisabilité. Nous tirerons de ces descriptions des directives quant à nos propres travaux.

Chapitre III

Le troisième chapitre conclura cette première partie et reviendra en détail sur les avantages apportés par l'utilisation de la technique des motifs pour le développement de logiciel.

—★—

Seconde partie : notre contribution

Chapitre IV

C'est dans le quatrième chapitre que nous développerons la méthode que nous avons choisie : les motifs de conception. Un état de l'art de l'utilisation des motifs pour la conception de SMA nous permettra d'introduire nos propres travaux. La liste des motifs orientés agent que nous avons désignés y est présentée avec leur description extensive. Ce chapitre et la liste des motifs que nous avons découverts constitueront la plus grande part de ce mémoire : un thésaurus de motifs agent.

Chapitre V

Le cinquième chapitre de ce mémoire nous permettra de discuter de l'utilisation des motifs comme outils d'aide à la conception. Il donnera un exemple de mise en application de notre *credo* : utiliser les motifs orientés agent que nous définissons pour développer et documenter une application multi-agent. L'application choisie est la simulation de robots réactifs dont le but principal est de placer des objets à des places précises.

—★—

Un bilan et quelques unes des perspectives envisageables sont présentés en conclusion.

Première partie

De la réutilisabilité en génie logiciel de la réutilisation pour les SMA

Chapitre Premier

De la réutilisabilité en génie logiciel

Ce qui est déjà fait n'est plus à faire.

Si vis pacem, para bellum.
— Proverbes

Les systèmes d'information[1] sont devenus indispensables, leur utilité n'est plus à prouver, et leur usage est plus fréquent chaque jour (pour s'en convaincre, il suffit de voir l'essor du secteur tertiaire depuis son apparition comme secteur des « services » à son état actuel où la principale fonction des métiers qu'il englobe est le traitement de l'information). Leur nombre augmentant, réutiliser, pour en créer de nouveaux, le travail déjà accompli pour leur mise en place est devenu indispensable : d'une part pour des raisons d'économie d'échelle mais aussi pour des raisons de fiabilité (plus un système est utilisé plus il a de chances d'être fiable).

Le but du génie logiciel est d'optimiser et de rationaliser le travail de conception de logiciel et le suivi de celui-ci. L'optimisation comprend plusieurs facettes : minimiser le temps passé, minimiser la complexité du travail à accomplir tant lors de la création que lors de la maintenance, augmenter la qualité et la fiabilité logicielles. La rationalisation consiste à formaliser les étapes de la production de logiciels pour rendre leur réalisation plus aisée, plus rapide et plus automatique, tant par la machine que par l'homme, qu'il travaille seul ou en équipe.

—★—

1. Pour une histoire des systèmes d'information, le lecteur pourra se reporte à la première partie de l'étude — très bien faite — d'Henri Ghesquière, [Ghesquière 98].

Dans ce chapitre, nous allons rapidement décrire les différentes techniques de réutilisation, puis nous nous pencherons plus particulièrement sur quatre d'entre elles : les composants, les *frameworks,* les motifs (ou *patterns*) et les protocoles.

Avant de conclure, nous parlerons des différentes formes d'utilisation d'un modèle (paramétrage, instanciation et utilisation).

1. Réutiliser

La réutilisabilité est un des moyens à la disposition du concepteur pour atteindre les buts du génie logiciel. En effet, la réutilisation réduit les étapes de conception et de test des composants réutilisés, elle permet aussi de simplifier la maintenance et l'évolution des applications. Elle permet aussi de construire des applications similaires dans leur interface, ce qui facilite leur usage et augmente leur qualité pour l'utilisateur final. Nous verrons aussi qu'elle peut aider à la formalisation.

Bien sûr, l'idée de réutiliser n'est pas nouvelle mais il faut atteindre un volume critique d'applications pour qu'elle soit réalisable et réalisée. D'une part, il faut que le coût de l'organisation nécessaire à la réutilisabilité puisse être compensé par le gain apporté par la réutilisation. Il faut souvent opérer une réorganisation pour pouvoir adopter une véritable politique de réutilisabilité, ce qui implique des problèmes aussi bien techniques qu'humains. D'autre part, il faut avoir une expérience et un ensemble d'exemples suffisants pour qu'elle devienne possible ; la réutilisation suppose l'existence de points communs entre les applications, et pour pouvoir détecter ces points communs, il faut à la fois une quantité suffisante d'exemples, une expérience technique et un effort d'abstraction.

2. Historique

Les différentes techniques de réutilisation suivent en fait le phylum conceptuel des paradigmes de programmation dont nous avons parlé dans le premier chapitre (section 2.1 page 3).

Parmi les techniques de réutilisation, la duplication — le célèbre « copier–coller » — est la première, tant par son ancienneté que par son utilisation. Cette technique permet de réutiliser les erreurs, d'application en application. De plus, elle implique une réadaptation du code copié et donc une ré-ingénierie du code. Enfin, elle ne s'applique guère qu'au niveau du codage, ce qui limite fortement son efficacité.

Les bibliothèques de fonctions sont une autre technique de réutilisabilité. Comme la précédente, celle-ci ne concerne que le codage mais, bien que plus sûre, elle ne permet pas véritablement la réutilisation d'algorithmes en cela que les fonctions manipulent des données de structure précise qui sont, trop souvent et pour des raisons d'optimalisation, des cas particuliers des types de données auxquelles l'algorithme pourrait s'appliquer.

La programmation modulaire a été la première tentative pour gérer la réutilisabilité dès la phase de conception.

> *« On dit généralement d'un système qu'il est modulaire lorsqu'il admet la modification, l'adjonction ou la suppression de certains sous-systèmes, sans que son efficacité globale s'en trouve fondamentalement affectée. »*
>
> — Bruno Lussato, 1980

Son évolution naturelle fut la programmation orientée objet qui permit l'apparition de la réutilisation de la structure des objets et de leur comportement fonctionnel — avec les méthodes — mais aussi des algorithmes, par leur définition relativement à des types abstraits et par encapsulation des algorithmes dans des objets (cf. la définition de la STL[2] de C++, [Stroustrup 99] et son utilisation du motif de conception *Command*, [Gamma *et al.* 94, p. 233]).

La programmation objet permet *sui generis* la réutilisabilité grâce aux mécanismes qu'elle introduit : l'encapsulation, la composition, l'héritage, le polymorphisme et le couplage dynamique.

3. Techniques objet

À partir des succès de la programmation par objets pour la réutilisabilité de base (au travers des classes), différentes techniques ont été mises en avant par les processus de conception objet : les composants, les *frameworks,* les motifs *(patterns)* et, dans une moindre mesure, les protocoles.

3.1. Les composants

Les composants sont des modules communicants étendus, des objets à faible interdépendance, à couplage faible. Leur but principal est de permettre de réutiliser une classe en facilitant son intégration dans le logiciel en cours de développement.

Pour qu'un composant soit utilisé dans un certain nombre d'applications, il faut qu'il soit portable et utile. La portabilité nécessite que le composant soit configurable et permette un assemblage et une intégration aisés. L'utilité du composant signifie qu'il comble un besoin de l'application envisagée.

Cette notion de « composant » émerge de deux approches distinctes : les architectures logicielles et les composants binaires.

L'approche des architectures logicielles vient du monde académique, la structure d'un logiciel correspond alors à une interconnexion de composants. Ce point de vue induit un certain flou sur la définition même d'un composant : module Ada, unité Pascal, objet, filtre à la Unix ? toutes ces notions peuvent répondre à cette définition.

L'approche des composants binaires est celle de l'industrie. Ici, c'est la réutilisation qui prime, et, bien sûr, les composants sont distribués sous forme binaire pour satisfaire aux exigences du marché (secret et concurrence). Cette approche repose sur le concept de « boîte noire » : un module réutilisable dont l'interfaçage externe est spécifié mais dont le fonctionnement interne est caché.

En parlant du concept de « boîte noire », il est à noter que c'est le principe de composition qui le fait apparaître. En effet, la composition cache les composants dans le composé : l'interface des objets composant n'est plus visible, seule l'interface du composé est disponible. C'est le contraire qui se produit avec la réutilisation par héritage : il faut connaître beaucoup de choses sur la classe héritée pour s'en servir correctement, il s'agit donc d'une « boîte transparente ».

Nous allons voir les trois technologies de composants concurrentes actuellement sur le marché : com, les EJB et le CCM.

2. *Standard Template Library,* bibliothèque standard de gabarits.

3.1.1 Component Object Model

Microsoft Corporation a proposé ses technologies componentielles : com et com+, qui utilisent une autre technologie maison : *Object Linking and Embedding* (ole et ole2), ole *Control eXtensions* (OCX), et, dans une moindre mesure, ActiveX — une version légère d'ole spécialement prévue pour l'Internet (en réponse aux *applets* Java de Sun).

Il s'agit d'une approche binaire fortement fermée (véritable « boîte noire » dont l'utilisateur ne connaît que les spécifications d'interface). Com+ repose sur la notion d'interface et de découverte à l'utilisation (par des mécanismes réflexifs simples).

L'un des principaux problèmes de cette approche, mise à part sa fermeture, est qu'elle met à bas les principes mêmes de la programmation objet : l'exemple flagrant est l'explosion de l'encapsulation engendrée par la mise en application du mécanisme d'agrégation (*i. e.*, l'accès aux composants de l'agrégation est toujours permis alors que l'encapsulation devrait les cacher).

3.1.2 *Les* JavaBeans *et les* Entreprise JavaBeans

Les *JavaBeans* sont la réplique de Sun Microsystems au com de Microsoft, mais en utilisant une approche moins binaire : l'utilisation de Java ouvre le format, mais, en contrepartie, il permet une grande portabilité.

Il s'agit avant tout de composants graphiques et événementiels dont le modèle de mise en œuvre est séparé du modèle conceptuel : la réflexivité y a une part très importante. Les *JavaBeans* sont distribués sous forme de fichiers d'archives (au format jar) ce qui permet de les utiliser en prenant l'approche binaire.

Les *Entreprise JavaBeans,* ou EJB, sont plus récents et font suite aux *JavaBeans* en développant leur champ d'action pour l'orienter du côté des applications industrielles, en créant des composants applicatifs domaine.

La composition des EJB se fait par l'utilisation des interfaces (*i. e.*, des promesses de services) ou par communication implicite.

3.1.3 Corba Component Model

Le CCM est le modèle des composants standardisé par l'OMG. Du fait que l'OMG est composé en majorité par des industriels, de tous bords, il est à michemin entre les deux modèles concurrents : com et EJB.

L'interface d'un composant est un ensemble de facettes, façon objets distribués (corba), et un comportement événementiel (à la EJB).

Il s'agit donc plus d'un EJB++, faisant un pas vers com, qui ajoute des outils standardisés aux EJB : gestion du cycle de vie, support des transactions, déploiement, etc.

3.1.4 *Conclusions sur les composants*

Com+ et EJB n'étaient pas satisfaisants. La technologie des EJB, par exemple, n'offre ni généricité, ni héritage, ni représentation des aspects relationnels. Ils ont été développés trop tôt (*i. e.*, avant que la communauté ne se mette d'accord, au travers de l'OMG, sur le modèle objet et sa distributivité, avec corba).

Le CCM tend à pallier les problèmes et à faire converger les modèles, mais il est encore récent.

3.2. Les *frameworks*

Un *framework*[3] est un prototype d'applications d'un même domaine, que le développeur n'a plus qu'à compléter — par sous-classage, écriture de fonctions, choix de paramètres — pour créer une application. Le *framework* permet de définir la structure globale de l'application, au niveau de la gestion du contrôle et de la collaboration des différentes classes ; il permet ainsi d'obtenir des applications de structures semblables, permettant un gain tant au niveau de la fabrication que de la maintenance.

Dans une programmation orientée objet, un *framework* est constitué d'un ensemble de classes, une bibliothèque de composants liés par des motifs de conception, dont les interactions sont prévues par le concepteur du *framework* et qui suivent donc des motifs de contrôle bien définis. Donc, on pourrait résumer par la formule suivante (que l'on retrouve dans [Johnson 97]) :

$$framework = \text{composants} + \text{motifs}$$

Le terme *composants* de la formule comporte lui même une partie codage : les composants sont écrits dans un certain langage de programmation.

En programmation objet, l'utilisation du *framework* se fait par instanciation et dérivation des classes le constituant ; en programmation fonctionnelle, il s'agit de donner le corps de certaines fonctions. Le développeur écrit donc les fonctions manquantes qui particularisent son application et qui sont en fait des fonctions client.

En effet, contrairement à ce qui se passe dans une bibliothèque de classes — où c'est le programme que l'on développe qui instancie et donne le contrôle aux classes de la bibliothèque —, dans une application créée à partir d'un *framework*, c'est ce dernier qui définit le contrôle, et les services développés pour cette application particulière seront appelés par le *framework*.

En fait, il arrive parfois que l'on croit utiliser une bibliothèque de classes alors qu'il s'agit d'un *framework* : une autre différence entre ces deux concepts est que, dans un *framework,* l'on est amené à apprendre et à utiliser plusieurs classes ensembles ; l'on qualifie souvent la bibliothèque de « difficile ».

Nous avons parlé dans la section précédente des *frameworks* que sont les technologies ole et *beans*. L'on peut encore citer AWT ou swing comme étant des *frameworks* d'interface graphique en Java. En C/C++, les *frameworks* CDE[4], KDE[5] et gnome[6] sont d'autres *frameworks* d'interface graphique mais aussi d'application (*e. g.*, gnome définit une architecture de communication entre composants/applications, fondée sur corba).

Pour que les *frameworks* soient utiles et utilisés, il faut qu'ils soient très bien documentés. Le développeur a souvent besoin de comprendre comment

3. Charpente, architecture.
4. *Common Desktop Environment.*
5. *K Desktop Environment.*
6. *Gnu Network Object Model Environment.*

fonctionne le *framework* pour l'adapter. La présence de bons tutoriels est indispensable : les *frameworks* sont bien plus compréhensibles et appréhendables par l'exemple.

C'est en fait en partie ce manque de documentation qui laisse penser que les plates-formes SMA actuelles ne sont utilisables que par leurs concepteurs et qui entraîne donc le développement de nouvelles plates-formes *ad hoc.*

Les *frameworks* sont une technique de réutilisation de la conception — puisque l'architecture, les modes de communication et d'interaction, sont conservés — mais qui reste ancrée dans la mise en œuvre, dans le sens où ils sont définis dans un langage de programmation particulier (comme les composants). Cet ancrage est un problème car certains concepts ne peuvent pas être correctement exprimés dans tous les langages et les *frameworks* deviennent de plus en plus spécifiques au langage dans lequel ils sont exprimés.

3.3. Les motifs (ou *patterns*)

Avant propos : de la traduction

Pour la traduction de terme *pattern,* nous avions le choix entre *patron, gabarit, motif, modèle* ou ne pas traduire. *Gabarit* est déjà utilisé pour le terme de *template* (notamment en C++), il désigne un type, ou une fonction, paramétré[7]. *Modèle* est déjà fortement polysémique et nécessiterait l'adjonction d'une explication[8]. *Patron* est en fait l'étymon de *pattern,* il a un sens très proche de *gabarit,* mais il a un second sens en français, celui de « dirigeant », qui peut créer une équivoque, une ambiguïté. *Motif* est la traduction usuelle de *pattern* en informatique : pour les expressions régulières, le traitement automatique de texte, etc.

Motif porte à la fois le sens de figure, forme, répétée et celui de raison, motivation. Or l'utilisation du terme *motif* dans le contexte du génie logiciel ne permet pas de distinguer clairement laquelle des deux acceptions fait le plus de sens ; il est en effet autant probable que l'on parle de répétitions ou de formes abstraites, modèles de plusieurs formes concrètes, que de motivations, mobiles, d'utilisations de techniques, de concepts ou de modèles d'ingénierie.

Toutefois, — et contrairement à celui de *patron* —, le second sens de *motif* (raison, mobile) n'est pas antinomique de la définition de nos *motifs :* comme nous le verrons plus en profondeur, ceux-ci sont une motivation, une raison, de leur propre application et des concepts qu'ils véhiculent. *Motif* nous semble donc un meilleur choix comme équivalent[9] du terme de *pattern.*

Malheureusement, quelques requêtes sur le site de recherche `www.google.fr` sur des pages francophones des expressions possibles nous indiquent que l'usage ne nous suit pas (ou plutôt, qu'il ne nous a pas précédé) :

	singulier,	pluriel,	total
« patron(s) de conception » :	136,	454,	590 pages
« motif(s) de conception » :	33,	43,	76 pages
« pattern(s) de conception » :	23,	121,	144 pages
« design pattern(s) » :	851,	3320,	4171 pages

7. La traduction est en fait très fidèle car ces mots référencent, chacun dans sa langue, une pièce de bois servant à en préparer d'autres (en construction navale).
8. Bien qu'il soit utilisé pour *gabarit* dans [Stroustrup 99].
9. Ou un peu plus, grâce à la notion de motivation que le terme anglais ne porte pas.

Toutefois, l'on peut remarquer que l'utilisation de l'expression en anglais est fortement majoritaire ; la traduction n'est donc pas encore figée. C'est pourquoi nous avons décidé d'essayer de faire basculer l'usage vers l'utilisation de « motif de conception » comme traduction de *design pattern.*

—★—

Avant de donner une définition et un historique de la notion de motif, nous allons voir quelques exemples de niveau différent (conception et mise en œuvre). Puis nous reviendrons sur leur forme, leur présentation, la façon dont ils sont décrits (présentation, forme, catalogage)...

3.3.1 Exemples de motifs

Nous commencerons par présenter deux motifs objet — un motif de conception et un motif de mise en œuvre —, puis nous présenterons un idiomatisme de script.

a) Un motif de conception

La figure 2 montre le motif *Composite*, [Gamma *et al.* 94, p. 163] ; en réalité, il s'agit seulement de la représentation en diagramme de classes UML de sa solution, le motif entier fait plusieurs pages. Ce motif permet de concevoir ses classes de sorte qu'un objet composite soit traité exactement de la même façon que ses composants.

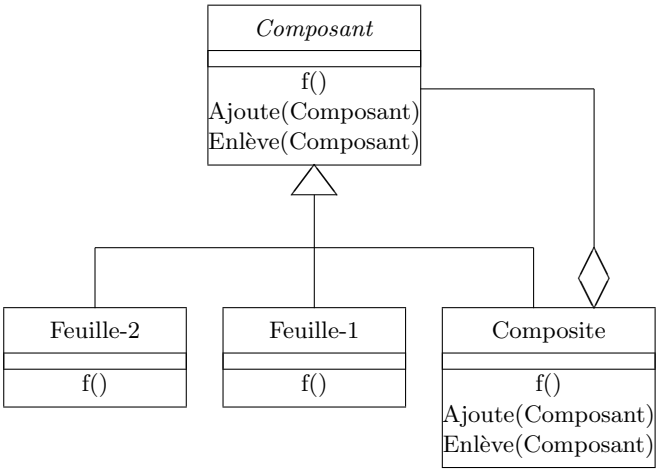


Fig. 2 – Un motif de conception *(Composite)*

La classe abstraite *Composant*, dont plusieurs classes d'atomes héritent (sur notre exemple : *Feuille-1* et *Feuille-2*), permet déjà de manipuler tous les atomes de la même façon (en appelant leur méthode `f()`, redéfinie par chaque sous-classe). La classe *Composite* est une classe pouvant contenir plusieurs atomes mais aussi d'autres objets composites, grâce au lien d'héritage avec la classe *Composant.* La fonction `f()` de *Composite* appelle la méthode `f()` de chacun de ses composants, atomes ou composites. Cette classe est la seule à avoir des fonctions `Ajoute()` et `Enlève()` effectives.

On pourra remarquer que ce motif utilise toutes les caractéristiques de la POO : héritage, polymorphisme, classe abstraite, composition.

La figure 3 montre l'application de ce motif sur un exemple d'objets graphiques : un dessin peut contenir des rectangles, des morceaux de texte ou bien d'autres dessins. Une figure sera représentée par un arbre de structure homogène dont les nœuds sont des dessins (les *composites*) et les feuilles des objets graphiques atomiques.

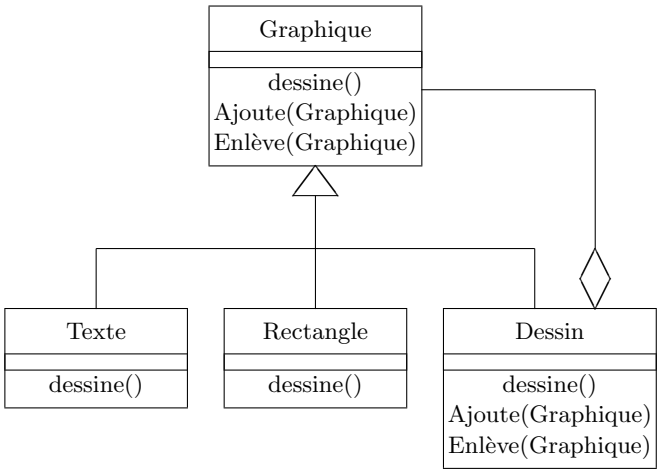


FIG. 3 – Utilisation de *Composite*

b) Un motif de mise en œuvre

Le motif *Composite* est un motif de conception, il est utilisé lors de la phase de conception. L'exemple suivant (figure 4) est un motif de mise en œuvre, un idiomatisme C/C++, qui indique comment coder la recopie d'une suite d'objets terminée par un pointeur nul. Ce motif permet au lecteur de comprendre immédiatement la signification d'une partie de code. En effet, sans l'utilisation de ce motif, il y aurait plusieurs dizaines de façons de coder cette recopie, noyant ainsi le lecteur dans des détails de mise en œuvre.

```
while (*dest++ = *orig++) ;
```

L'affectation a lieu avec les valeurs couramment pointées, puis les pointeurs sont incrémentés. La valeur testée est celle renvoyée par l'affectation, c'est-à-dire la valeur pointée par `orig`*, la boucle s'arrête ainsi dès le premier zéro copié.*

FIG. 4 – Un motif de mise en œuvre (idiomatisme C/C++).

L'on pourrait penser qu'il ne s'agit là que d'un style d'écriture, et il est vrai que les motifs de mise en œuvre en sont très proches et parfois difficilement discernables. Toutefois, ce qui différencie ces motifs d'une simple directive de style, c'est qu'ils sont argumentés et qu'ils prennent leurs racines à d'autres niveaux (comme la conception). Si nous reprenons notre exemple de la figure 4, nous pouvons voir que cette façon de coder une recopie permet, grâce à une habile surcharge des opérateurs [STROUSTRUP 99], de concevoir des itérateurs pouvant se comporter de cette manière, c'est-à-dire celle des « bonnes vieilles » chaînes de caractères du C, ce qui permet de cacher la mise en œuvre de notre suite d'objets et donc une meilleure programmation.

c) Un idiomatisme shell

L'utilisation du motif *Testing Against The Null String*, [Coplien *et al.* 96], permet, dans des langages de script *shell*, de sécuriser les tests sur des variables.

Un test écrit sous la forme suivante :

```
if [ $a = coucou ]; then echo "chaîne \"coucou\""; fi
```

peut être d'une part une source de bogues, et d'autre part une porte ouverte à de graves atteintes à la sécurité du système. En effet, si la variable *a* est vide, le test revient à écrire :

```
if [  = coucou ]; then echo "chaîne \"coucou\""; fi
```

ce qui n'est pas syntaxiquement correct.

De la même façon, en ajoutant des guillemets pour éviter cette erreur, si la variable *a* contient des termes reconnus par la commande `test`[10] comme des opérateurs (*e. g.*, `-a` ou `!=`), le problème continue à se poser.

D'autre part, si le contenu de la variable *a* provient d'arguments passés au script ou de variables modifiables par l'utilisateur du script, l'utilisateur peut en profiter pour demander l'exécution de commandes dangereuses ; surtout si le script est `suid` (*i. e.*, il est exécuté avec l'identité du propriétaire, pas avec celle de l'utilisateur).

Pour vérifier que la variable a correspond à une certaine chaîne, utiliser des tests du genre :

```
if [ x$a = x ]; then echo "chaîne vide"; fi
case x$a in
    xcoucou) echo "chaîne \"coucou\"" ;;
    x)       echo "chaîne vide" ;;
esac
```

Fig. 5 – Un motif *shell* (idiomatisme de script).

—★—

Nous venons de présenter trois motifs s'appliquant chacun à un domaine différent : le premier est un motif de conception objet, le deuxième un motif de mise en œuvre objet et le dernier un motif de mise en œuvre script. Cela montre la diversité d'application des motifs, tant du point de vue du domaine d'application (programmation objet ou script shell) que du point de vue du niveau d'abstraction ou de la phase de développement visée (conception ou mise en œuvre).

Nous aurions pu donner bien d'autres exemples de motifs, dans des domaines très variés, comme les interfaces utilisateur [van Welie & Trætteberg 00], la construction de sites internet[11], l'organisation d'équipes *(groupware)*[12], etc. (voir [Patterns'Page] pour des catalogues en ligne).

10. Rappel : en *shell,* `[` est un raccourci pour la commande `test`.
11. Motifs que l'on retrouve sur `http://www.welie.com/patterns`.
12. Motifs disponibles sur `http://www.groupware-patterns.org`.

Voyons maintenant ce qu'est précisément un motif.

3.3.2 Définition et historique

Une définition courte et générale, mais néanmoins précise, de ce que peut être un motif pourrait être la suivante (et c'est en fait notre définition favorite) :

Un motif est une solution abstraite (partielle et qui a fait ses preuves) à un problème contextuel et récurrent.

Les motifs ont été introduits en 1977 par l'architecte Christopher Alexander ; il s'agissait alors de donner des *indications* pour la bonne conception de plans : à chaque problème ou spécification, une solution éprouvée était proposée. Mais cette utilisation en architecture n'a pas été concluante. L'architecture est peut-être trop une matière de goût, ou les lois physiques sont peut-être trop contraignantes...

Au début des années 1990, la communauté orienté-objet, se trouvant alors en possession d'une expérience certaine de la POO, s'est intéressée aux motifs dans le but d'exploiter et de faire partager cette expérience, [Gamma *et al.* 93]. Depuis, leur utilisation ne fait que croître, notamment grâce aux conférences, [Plop 94], et à l'appui de l'OMG, qui propose de les intégrer à UML.

Comme l'a montré la systémique, [Simon 69, Le Moigne 90], la conception de systèmes complexes est elle-même complexe. On ne peut pas définir *la* solution pour un ensemble de problèmes. Par contre, étant donné la modularité de l'approche objet, il arrive souvent que le concepteur ou le programmeur se retrouve en face d'un problème qu'il a déjà rencontré dans le passé, sous la même forme ou sous une forme légèrement différente. Grâce à cette impression de *déjà vu*, il peut alors essayer d'abstraire la solution de ce problème pour pouvoir la réutiliser. Les abstractions des solutions déjà appliquées sont ce que l'on appelle des motifs.

Par la façon dont ils sont construits, les motifs sont une abstraction, une formalisation, de l'expérience acquise par les concepteurs et les programmeurs. Les motifs permettent aussi de documenter les solutions proposées : il suffit au concepteur d'indiquer les motifs qu'il a utilisés pour qu'un lecteur averti comprenne la solution, sans avoir à reprendre chaque détail pour appréhender l'ensemble.

3.3.3 Présentation

En ce qui concerne leur présentation, les motifs sont décrits sous la forme d'un multiplet. Il s'agit de formaliser leur présentation en la séparant en différents champs aux buts bien définis pour bien guider l'utilisateur, mais en laissant le contenu de certains des champs plus libre — grâce à l'usage du langage naturel — pour permettre à la fois une meilleure compréhension et une souplesse, une adaptabilité, d'utilisation.

Les champs varient suivant les auteurs, mais les principaux sont les suivants :

– *nom :* chaque motif possède un nom (parfois même plusieurs, s'il a été décrit par plusieurs auteurs) qui permet de l'identifier ;

– *contexte :* il s'agit ici d'expliciter la situation dans laquelle le motif a des chances d'être utile ;

– *problème(s) :* le motif peut résoudre un ou plusieurs problèmes, dans leur contexte ;

– *solution(s) :* la ou les solutions possibles au(x) problème(s) ;

– *exemples :* ils permettent de mieux comprendre le motif, celui-ci étant créé de manière empirique ; pour cette même raison, ils sont absolument nécessaires au motif et doivent apparaître au moins au nombre de trois (il s'agit de la « règle des trois ») ;

– *directives :* pour diriger l'application de la solution, expliquer les parties importantes et les parties plus facultatives ;

– *conséquences :* comme chacun sait, une solution peut apporter d'autres problèmes, il est donc utile d'en faire part à l'utilisateur ; elle peut aussi apporter d'autres avantages, indépendamment de la résolution des problèmes visés initialement ;

– *motifs associés :* une liste de motifs applicables dans des situations similaires ou des motifs complémentaires, pour régler les problèmes annexes ou qui se posent à d'autres niveaux (pour la mise en œuvre par exemple).

Les champs choisis dépendent des auteurs et de la façon dont ils ont classé leur motifs, de la façon dont leurs motifs interagissent, mais dans la rédaction même des motifs l'on retrouve des motifs.

3.3.4 Catalogue — thésaurus

Comme il existe des bibliothèques de fonction ou de classes, les motifs sont groupés en catalogues. Ces catalogues ne se résument pas à des énumérations de motifs. Les motifs sont liés : ils se référencent, s'opposent ou s'adaptent les uns avec les autres. Le catalogue forme un véritable thésaurus : chaque motif est un mot et sa description indique comment s'en servir et quels sont les motifs (les autres mots) qui lui sont associés (synonymie, antonymie, dérivation, analogie, etc.).

Cette caractéristique est importante pour bien comprendre la portée de la technique des motifs, c'est pourquoi nous y reviendrons plus en détail dans la section III page 61.

3.3.5 Antimotifs

La technique des *AntiPatterns* ont été introduits par Hays McCormick, Raphael Malveaux, Thomas Mowbray & William Brown, [McCormick *et al.* 98, McCormick & Malveaux 98]. Contrairement aux motifs qui mettent en avant une méthode, une technique ou un usage qui forment une bonne solution, les antimotifs exposent l'application d'une mauvaise technique et proposent une solution pour repenser la résolution du problème. Il s'agit donc de *« re-engineering »*, (*i. e.*, *reconception* ou *ré-ingénierie*).

Lava Flow est un exemple d'antimotif. Il s'agit de remodeler une application, souvent issue du département recherche et développement, dont le code est peuplé de fonctions ou de parties exploratoires, très peu commentées, voire pas du tout, dont l'utilité (et l'utilisation) est à démontrer.

Symptômes :

– fragments de code et variables injustifiables et fréquents ;

– code qui semble important ou complexe mais non documenté et qui ne se relie pas clairement à l'architecture du système ;

– architecture du système très lâche ;

– larges blocs de code en commentaire, sans documentation ;

– nombreuses parties annotées « à remplacer » ou « à revoir » ;

– code inutilisé ou obsolète.

Causes typiques :

– code de test placé en production sans réflexion préliminaire ;

– code écrit par un seul développeur ;

– plusieurs approches testées en même temps ;

– manque d'organisation dans l'équipe ;

– développement itératif (*e. g.* en spirale).

Exceptions :

– de petites applications de recherche & développement, à durée de vie très limitée.

Conséquences :

– si le code n'est pas nettoyé, la « lave » continue à se répandre ;

– si l'origine n'est pas éliminée, l'expansion peut être géométrique avec le passage successif de plusieurs développeurs, pressés et non enclins à vérifier le code déjà écrit ;

– à mesure que la lave durcit, elle devient plus difficile à éliminer.

Solution :

La seule solution pour prévenir l'antimotif *Lava Flow* est de s'assurer que le développement du code soit réalisé après qu'une architecture saine a été établie. Si le *Lava Flow* est déjà présent dans l'application, le remède peut être douloureux, et il faudra un excellent détective pour nettoyer le code. Un principe important est de ne pas modifier l'architecture pendant le développement. Un autre est de clairement documenter chaque partie du système.

—★—

Les antimotifs sont donc aussi des motifs en ce sens qu'ils proposent une solution à des problèmes fréquents. D'un autre côté, leur approche s'oppose à celle des motifs puisque les antimotifs ne s'appuient pas sur des exemples de bonne application d'une solution. Ils se fondent sur des exemples pour proposer une solution de reconception.

3.4. Motifs et protocoles de communication

En informatique, les protocoles de communication sont les règles qui déterminent le format et la transmission de données. Dans le vocabulaire des SMA, où nous nous rapprochons plus des sciences humaines, les protocoles sont la description d'un plan (plus ou moins) détaillé pour la conduite d'un comportement communicateur, notamment pour une transaction.

L'on peut aussi définir plus simplement un protocole comme étant une séquence type de messages. De ce fait, leur représentation sous la forme d'un

diagramme de séquence UML paramétrable semble adéquate (cf. la notation AUML, page 41).

La fipa[13] propose quelques exemples de protocoles ; tels le *Brokering IP,* le *Dutch Auction IP* (enchères hollandaises), ou le fameux *Contract Net.* Pour décrire ces IPS (les *Interaction Protocol Specifications*), il est fait usage de la notation AUML, dont nous parlerons (cf. page 41). L'on peut trouver ces spécifications sur le site de la fipa : `http://www.fipa.org`.

La figure 6 montre un exemple (simple) de protocole que la fipa propose, le *Query IP.* Dans ce protocole, un agent *(Participant)* doit donner une information *(inform)* à l'agent demandeur *(Initiator).* Le demandeur a deux actes de langage possibles (exclusifs l'un de l'autre) : *query-if* et *query-ref* pour demander une information ou la référence d'un objet. L'agent interpellé a quatre actes de langage possibles en réponse, ces actes sont eux aussi génériques dans le sens où ils peuvent contenir des messages différents (*e. g.*, la raison du refus pour *refuse*), c'est l'astérisque après leur nom dans le cadre de gabarit qui dénote cette généricité.

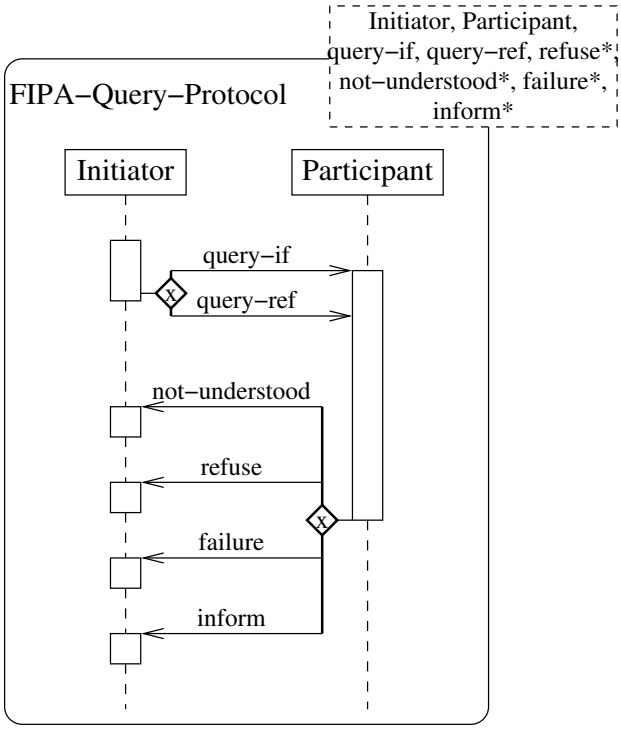


FIG. 6 – Le protocole d'interaction *Query* de la fipa.

L'utilisation du terme *agent* n'est pas anodine : il est clair, comme nous l'avons souligné au premier chapitre[14], que les notions d'agent et de rôle — en relâchant le lien entre le porteur du rôle et son type, sa classe structurelle — permettent une plus grande généricité d'application de ces protocoles, que ce soit dans un SMA ou dans une conception purement orientée objet.

L'utilisation d'un catalogue de protocoles, comme le propose la fipa, per-

13. *Foundation for Intelligent Physical Agents.*
14. Plus précisément page 5.

met, en plus de la réutilisabilité des protocoles, et comme pour les motifs, l'utilisation d'un vocabulaire commun entre les différents systèmes, et donc une meilleure interopérabilité.

Toutefois, il est important de bien comprendre qu'un protocole, même paramétré (*i. e.*, décrit comme un gabarit de protocoles, comme l'exemple de la figure 6 page précédente), n'est pas un motif. Ce qui pourrait être un motif, c'est la description des cas de bonne utilisation de tel ou tel protocole ; par exemple, quand faut-il choisir des enchères anglaises ou un *contract net* itéré. Le type d'instanciation n'est pas le même et c'est le pourquoi de la section qui suit : différencier les moyens de passer du modèle à son utilisation.

4. Du modèle à son utilisation

Note préliminaire : la programmation orientée objet s'est accaparé le terme d'*objet*, il devient difficile, surtout en informatique et encore plus en génie logiciel, d'utiliser ce terme sans une connotation programmation orientée objet. Toutefois, dans cette section, nous utiliserons ce terme d'objet dans son sens large (nous essaierons d'être vigilant lors de son utilisation au sens d'instance de la programmation objet).

L'informatique, en tant que science, crée des modèles et les manipule, mais sa particularité est qu'elle a introduit leur traitement automatisé. Pour ce faire, les modèles et les objets qu'ils représentent sont réifiés en mémoire, ils sont engrammés. Aux objets du monde sont associés d'autres objets du monde, représentations des premiers. Aux modèles du monde sont aussi associés des objets, des données engrammées dans la mémoire de l'ordinateur.

Les concepts du modèle manipulés sont aussi des objets. Ainsi, les objets manipulés par l'ordinateur sont-ils à la fois des engrammations des objets individuels du monde et des engrammations de leur essence, de leur idée, de leur modèle — le terme exact est difficile à trouver. Par exemple, si l'on représente un stock de meubles, chaque table, chaque chaise du stock — les objets du monde — auront chacune une engrammation, mais l'on pourra aussi représenter l'idée, le modèle, de *chaise* — le concept décrit et utilisé dans le modèle pour représenter l'ensemble des objets de *type*[15] chaise.

C'est l'idée de *classe* de la programmation orientée objet : la découverte de Kristen Nygaard d'intégrer la catégorisation des objets dans le langage de programmation (simula, à l'époque) en décrivant des classes à partir desquelles les objets sont créés, initialisés et activés.

Ce concept d'*idée*, de *catégorie*, de *modèle*, de *moule*, de *classe*, est fondamental de la conception et de la réutilisation.

Il existe différentes façons de passer du *modèle* à son utilisation, l'*objet*. Nous pouvons en différencier trois : l'instanciation, l'héritage et le paramétrage.

4.1. L'instanciation

La notion d'instanciation est primordiale en conception objet. Il s'agit de créer un individu (*instance* ou *objet*) à partir d'une idée abstraite représentant

15. Quel que terme que l'on choisisse (idée, modèle, classe, catégorie, type, sorte, espèce, etc.), l'on rencontre toujours un domaine où il a un sens plus précis (philosophie, paradigme de conception ou de programmation, philologie, biologie, etc.).

un ensemble d'individus (une *classe*). Jean-Pierre Briot et Pierre Cointe le soulignaient particulièrement avec le modèle Object/Class d'ObjVlisp, [Briot & Cointe 86].

L'instanciation permet à différents objets de partager des structures et des comportements par le biais d'une classe-modèle commune.

4.2. L'héritage

La notion d'héritage est arrivée peu après la notion d'instanciation — à tout le moins en programmation orientée objet. Il s'agit de catégoriser les classes les unes par rapport aux autres par les relations de spécialisation et de généralisation.

L'héritage permet à différentes classes de partager des structures et des comportements par le biais d'une classe-mère commune.

4.3. Le paramétrage

La notion de paramétrage est parallèle aux deux autres. Paramétrer consiste à donner certaines valeurs à certains *attributs*, au sens large de « trous » pouvant prendre une valeur, pas au sens restreint de « champs d'une classe ».

Le paramétrage, c'est utiliser, dans un cas plus spécialisé, un objet (une fonction, un type ou tout autre objet conceptuel) générique, un prototype.

Le paramétrage permet à différentes fonctions, à différents types, de partager la même description. Chaque fonction, chaque type, devient une spécialisation de son générique.

En fait, si l'héritage permet un polymorphisme d'exécution, le paramétrage permet un polymorphisme de compilation[16].

4.4. Quelques exemples

Voyons maintenant quelques exemples d'instanciation, d'héritage et de paramétrage.

Les composants (EJB ou com+) sont instanciables quand ils sont distribués sous forme de classes. Une fois instanciés, ils sont paramétrables : on peut affiner leurs comportements et leur apparence.

Les classes des bibliothèques de classes sont, le plus souvent, utilisables par instanciation, et, généralement, elles sont réutilisables par héritage.

Les *frameworks* sont paramétrables : l'utilisateur écrit le corps des fonctions attendues par le *framework*. Ces fonctions sont parfois encapsulées dans des classes utilisateur (*e. g.*, quand il s'agit de définir une classe réalisant une classe abstraite ou une interface), ces classes seront alors instanciées dans l'application engendrée par l'utilisation du *framework*.

Les gabarits (en C++) et les génériques (en Ada ou en Eiffel) sont paramétrés, les paramètres peuvent être des valeurs ou des types. Un gabarit représente un

16. Bjarne Stroustrup qualifie les gabarits d'« artifices de compilation » car ils permettent le typage fort à la compilation tout en n'écrivant qu'une seule description pour plusieurs utilisations, [Stroustrup 99].

type, ou une fonction, générique qui sera spécialisé lorsque l'on fixera tout ou partie de ses paramètres.

Les protocoles peuvent être instanciés en des séquences de communication. On peut aussi spécialiser un gabarit de protocole pour qu'il devienne moins générique (comme les gabarits de type en C++) en donnant des valeurs à ses paramètres (cf. le motif *Protocoles*, page 87).

Les motifs sont utilisés, il ne sont ni instanciés ni paramétrés. Il s'agit d'appliquer leur solution. L'on peut dire que cela relève à la fois de l'instanciation et du paramétrage : on crée un nouvel exemple d'utilisation de ce motif (instanciation) et l'on remplit les vides laissés (paramétrage), mais l'on adapte aussi les solutions proposées.

5. Conclusions

5.1. Notre choix

Les composants sont une très bonne idée pour la réutilisation, il s'agit d'objets de granularité plus élevée que les fonctions ou les classes, donc d'une plus grande abstraction, donc d'une plus grande généricité.

Par contre, les composants sont souvent orientés métier, ce qui rend difficile la création de composants utiles pour un grand nombre d'applications ; tels qu'ils doivent l'être si l'on veut aider à la conception de SMA. De plus, il n'y a pas encore d'architecture commune susceptible de les accueillir, et utiliser com+ ou les EJB demande un investissement important.

Les *frameworks* sont construits chacun pour une classe d'applications particulières et sont un peu trop fermés pour le but que nous nous sommes fixé. Créer une n^{e} plate-forme générique ne nous semble pas une bonne idée si l'on n'y intègre pas un modèle spécifique d'agent, d'environnement ou d'interaction.

Les protocoles décrivent le fonctionnement d'un système, son interface externe et les interfaces de communication de ses éléments. Ils sont une partie de la spécification du système mais ils ne le définissent pas.

Par contre, les motifs, bien que plus abstraits, restent fondés sur un processus expérimental. Ils sont génériques et adaptables tout en restant applicables (car appliqués). De plus, ils peuvent être utilisés en parallèle avec d'autres outils. Ces raisons font d'eux l'objet de la partie suivante, partie principale de notre travail : l'étude de leur application au paradigme agent.

5.2. Réutiliser

Il y a différents niveaux de réutilisation. L'on peut réutiliser des concepts, des processus (méthodes et algorithmes) ou des structures. Pour ce qui est des concepts, il s'agit de réutiliser une ontologie, le résultat d'une analyse ou d'une modélisation. C'est aussi ce que l'on fait dans la programmation objet avec les notions de classe et d'héritage. En ce qui concerne les processus, l'on peut réutiliser les algorithmes utilisés pour manipuler les données. L'on peut aussi réutiliser les méthodes utilisées pour lors des diverses phases du développement. Enfin, l'on peut vouloir directement réutiliser les structures, les objets réalisés à l'issue de développements précédents plus ou moins similaires. Il s'agit ici de

réutiliser des parties de code déjà écrites et, aussi, la structure organisationnelle de tout ou partie du logiciel (architecture).

Les techniques de réutilisation que nous venons de voir se placent différemment sur ces trois niveaux. La duplication est purement une réutilisation de structure, dans le sens où le code est une donnée. Les bibliothèques de fonctions sont une réutilisation des algorithmes et du code de ces algorithmes. C'est donc principalement une réutilisation de processus et, aussi, de structure. Les composants sont une réutilisation de structure (le code, la structure des classes et leurs relations), de processus (le fonctionnement interne des composants) et de concepts (les classes). Les *frameworks* sont principalement une réutilisation de structure (le code et l'architecture de contrôle du logiciel) et de processus (le fonctionnement de l'application). Les motifs permettent de réutiliser des concepts (classes et modèles), des processus (chemin du problème à sa solution, comportement du logiciel) et de structure (exemples de codes et, surtout, architecture de contrôle).

Il existe donc un type de réutilisation qui n'est pas présent dans ces techniques : la réutilisation des méthodes de développement (analyse, conception, mise en œuvre).

Nous allons voir, dans le chapitre suivant, dans quelles mesures la réutilisation est appliquée dans le développement de SMA. Nous verrons qu'elle repose principalement sur les structures (bibliothèques de classes, *frameworks*) et les processus (méthodes).

Chapitre II

Les propositions actuelles

A method of solution is perfect if we can foresee from the start, and even prove, that following that method we shall attain our aim.

— Gottfried Wilhelm LEIBNITZ

Les programmes dépendent des méthodes utilisées pour les construire.

— Kristen NYGAARD, Conférence invitée à OCM'2000, Nantes (18 mai 2000)

Les efforts actuels de la communauté agent portent sur la recherche de méthodes pour l'analyse, la conception et le déploiement de systèmes multi-agents. Ce domaine de recherche n'est en fait pas récent (déjà aux premières journées francophones IAD & SMA : [Ovalle & Garbay 93]), mais il prend de plus en plus d'intérêt, [Iglesias *et al.* 98]. Ces efforts prennent deux directions opposées (mais complémentaires) : la première peut être qualifiée de descendante et la seconde d'ascendante.

Le premier groupe de travaux, utilisant une optique descendante, essaie de concevoir des méthodes d'analyse et de conception des SMA. Le second groupe, l'approche ascendante, tente de fournir des outils de développement ; ces derniers ne sont pas tous exempts de méthodes ou de directives d'analyse–conception mais leur optique a un but pratique de développement.

Dans la suite de ce chapitre, nous allons considérer quelques exemples des travaux actuels de la communauté. Nous commencerons par exposer quelques caractéristiques des plates-formes et outils de développement de SMA, puis nous examinerons l'approche descendante des méthodes de conception agent.

1. Les plates-formes et outils

Encore plus que les méthodes ou les modèles, les plates-formes sont pléthore, il suffit de consulter les descriptions des plates-formes et outils de développement francophones collectées dans [Boissier *et al.* 99] pour s'en convaincre. Nous allons analyser et compléter cette liste par quelques outils et plates-formes actuellement disponibles.

Pour ce faire, nous avons classé ces propositions en quatre catégories (non exclusives, chaque outil étant placé dans la catégorie qui le définit le plus à nos yeux) :

1. *les langages de programmation :* ce sont des langages programmation, ou des augmentations de langages, intégrant des concepts agent ;
2. *les bibliothèques de classes :* ce sont des ensembles de classes, de fonctions ou de composants facilitant la mise en œuvre de SMA ;
3. *les outils de simulation :* conçus pour la simulation de systèmes complexes avec de nombreux agents ;
4. *les solutions :* environnements de développement intégrés, ils proposent à la fois des bibliothèques, des architectures (d'agent, de communication, etc.), des méthodes et des outils de réalisation.

1.1. Les « langages » de programmation

Une façon de réutiliser les concepts est de les intégrer directement dans le langage de programmation. C'est ce qu'essaient de faire les outils suivant.

Jack Intelligent Agents™, [Hodgson *et al.* 99, Howden *et al.* 01], augmente Java de quelques nouveaux mots-clefs (*e. g.*, *agent, plan* ou *event*). Les agents de jack sont, à la base, de type BDI et l'utilisateur crée ses agents en faisant hériter leur classe des classes fournies par jack. Le modèle temporel utilisé est discret. Des outils de développement et de surveillance sont fournis (d'autres sont prévus). Jack propose aussi un modèle de coopération intra- et inter-groupe d'agents appelé *SimpleTeam.*

Jack, bien qu'imposant le modèle BDI, se révèle un bon outil pour prototyper un SMA communicant.

Dima, [Guessoum & Briot 98a, Guessoum & Briot 99], est une plate-forme agent à base d'acteurs. Le principe fondateur de dima est d'utiliser les acteurs comme base de mise en œuvre pour les agents. Dima fournit donc les classes et mécanismes nécessaires en Smalltalk.

Concurrent MetateM, [Fisher & Wooldridge 96], est un langage de programmation multi-agent, fondé sur une logique modale (temporelle et de croyances), qui permet une spécification et une vérification formelles du système. Les agents sont des moteurs d'inférence sur une logique temporelle, ils infèrent des faits pour le présent et le futur à partir de faits dans le passé et le présent.

L'outil est en cours de développement et de validation...

1.2. Les bibliothèques de classes

Dans les bibliothèques de classes, c'est à la fois le code et les concepts véhiculés par les classes que l'on essaie de réutiliser, sans trop d'*a priori* sur l'application visée.

GTMas[1], [Chevrier 93], a été développée pour simuler différents types d'organisation d'un SMA, dans le but de les comparer. Bien que, par ses buts et par les outils qu'il fournit, GTMas soit en fait un environnement de simulation, il a été conçu de façon modulaire pour que chaque module puisse être utilisé indépendamment des autres ; c'est pourquoi nous l'avons placé dans cette section.

GTMas propose différents types de communication : point à point, diffusion *(broadcast)* ou par tableau noir. Bien que ne faisant aucune pré-supposition sur l'architecture interne des agents, GTMas propose un moteur d'inférences facilitant la création des agents. Un autre intérêt de GTMas provient des outils de suivi qu'elle propose : sélection des informations utiles, visualisation de ces informations après l'exécution de la simulation, statistiques, etc. Cette plate-forme est une des premières plates-formes agent francophones, elle a été réutilisée et augmentée par la suite.

Javama, [Foisel *et al.* 97, Foisel 98a], cette plate-forme utilise en fait la plateforme GTMas (que nous venons de décrire) en y ajoutant une optique magma (cf. la grille *Voyelles* page 55) et en y implantant le modèle de réorganisation automatique que Rémy Foisel a développé dans sa thèse, [Foisel 98b]. Ce modèle de réorganisation automatique est d'ailleurs son principal intérêt.

Geamas, [Guerrin *et al.* 98], est une plate-forme conçue par l'équipe mas[2] de l'iremia de l'université de l'Île de la Réunion. Il s'agit en fait d'une bibliothèque de classes Java qui facilite la construction des agents, la gestion des interactions et de l'environnement. Il est à noter que geamas propose de distinguer différents types d'agent suivant leur niveau d'abstraction :

- des agents réactifs au niveau micro ;
- des agents cognitifs au niveau médium ;
- une société d'agents au niveau macro.

Cette plate-forme propose aussi un éditeur intégré et des outils de surveillance du système.

Maleva, [Lhuillier 98], est une plateforme componentielle, c'est-à-dire qu'elle est fondée sur le concept de composant (cf. chapitre précédent). Son principe est de composer des éléments de comportements, des composants attachés à une tâche précise, des composants de contrôle, de combinaison d'autres composants, des composants pour les effecteurs et les capteurs, pour construire les agents. Maleva propose un éditeur/constructeur graphique d'agents et de composants et un environnement de simulation. Maleva a été implantée dans différents langages : Smalltalk, Pascal (Delphi), PtitLoo et Mering.

1. *General Toolkit for MultiAgent Systems.*

Comet, [Peschansky *et al.* 00], a été développée au lip6, en compagnie de Maleva mais sur un principe différent : les composants qu'elle propose communiquent par événements asynchrones.

L'intérêt de ces deux dernières plates-formes est bien évidemment l'utilisation de la technique des composants et donc une facilité de réutilisation des composants déjà construits dans d'autres SMA. Mais l'apprentissage et la prise en main ne sont peut-être pas très aisés.

Jatlite[2]**,** [Jeon *et al.* 00], du CDR de l'université Stanford, propose un dispositif de distribution d'agents sur un réseau grâce à un agent routeur spécial, le jatlite *agent message router,* qui achemine les messages KQML entre les agents Java, rien de très nouveau donc.

GrassHopper, [Bäumer *et al.* 99, GrassHopper 99], est une plate-forme développée par GMD fokus et ikv++, fondée sur le standard masif[3] de l'OMG et les spécifications de la fipa[4]. Elle supporte plusieurs moyens de communication : corba[5] IIOP[6], maf IIOP[7], Java RMI[8] ou *sockets.* Elle propose tous les services décrits par masif ainsi que tous ceux définis par la spécification *Agent Management System* de la fipa.

Cette plate-forme répond donc entièrement à tous les standards actuels, même si ceux-ci se limitent aux aspects communicatoires des SMA.

Hive[9]**,** développée aux MIT AI *labs,* [Minar *et al.* 99], est axée sur l'utilisation d'agents pour rendre le *web* dynamique. Cette plate-forme est écrite en Java et utilise la technologie RMI pour la distribution et la communication des agents.

La principale utilisation de cette plate-forme sont donc les applications d'assistanat sur l'Internet.

Jini, `http://www.sun.com/jini`, de Sun Microsystems, est une technologie qui n'a jamais été présentée comme une plate-forme ou un outil agent mais qui utilise les concepts agents dans son modèle. Cette technologie repose sur la technologie RMI et sur les concepts de service de nommage et de *proxy*, cela permet la distribution des resources et des services, leur découverte, définition et chargement dynamiques.

1.3. Les outils de simulation

Les outils de simulation sont de véritables *frameworks.* Leur but est de réutiliser la structure du logiciel à travers son architecture de contrôle ainsi

2. *Java Agent Template, Lite.*
3. *Mobile Agent System Interoperability Facility.*
4. *Foundation for Intelligent Physical Agent,* `http://www.fipa.org`
5. *Common Object Request Broker Architecture,* le standard pour la distribution d'objets de l'OMG, voir [Geib & Merle] pour une introduction.
6. *Internet Inter-ORB Protocol,* le protocole de communication sur l'Internet entre deux sites corba.
7. Spécialisation d'IIOP définie par le standard masif.
8. *Remote Method Invocation,* la technologie de distribution d'objets de Java, voir [Bellot & Matiachoff] pour une introduction.
9. La « ruche ».

que son code.

Swarm, [Minar *et al.* 96], développée au *Santa Fe Institute*, utilise le langage de programmation *Objective C*. Le principe de cette plate-forme est d'exécuter en concurrence un ensemble d'agents situés dans un environnement commun. Cette simulation est à temps discret. Cette plate-forme se destine surtout à l'éco-simulation.

Cormas, [Bousquet *et al.* 98], est une plate-forme de simulation où l'espace est géré par un automate cellulaire. Elle a été développée par le cirad de Montpellier avec le modèle swarm comme inspiration. Elle est écrite en Smalltalk (une version Java est prévue). Les agents y ont une dynamique propre, mais discrète, et chaque cellule de l'environnement peut contenir un autre automate cellulaire représentant un point de vue d'une granularité plus fine de l'espace.

Cette plate-forme peut s'appliquer à toute simulation spatialisée. Elle possède aussi des outils d'observation du système (notamment des interactions entre les agents : liens d'accointance, proximité interactionnelle, etc.).

Mobidyc, [Ginot & Le Page 98], est elle aussi une plate-forme de simulation destinée avant tout à l'écologie. L'espace y est discrétisé et son évolution est prise en charge par un automate cellulaire. Mobidyc est, dans l'esprit, le petit-frère de Cormas et de swarm.

1.4. Les solutions

AgentBuilder, `http://www.AgentBuilder.com`, est une plate-forme commerciale de développement de systèmes multi-agents. Elle propose des outils de développement (bibliothèque, débogueur) et une plate-forme d'exécution Java pour des agents ontologiques communiquant par messages KQML[10].

Aglet[11]**,** [Clements *et al.* 97], propose de construire des agents internet (*i. e.*, légers, mobiles, persistants et événementiels) en Java. L'idée principale est la mobilité du code, en opposition avec la transmission de requête proposée par les modèles objets (RPC[12], RMI, corba). La plate-forme propose donc un environnement graphique pour construire des applications d'agents mobiles, un serveur d'agents (le noyau de communication) ainsi que la spécification d'un protocole de transfert d'agents (ATP) pour la communication entre différents serveurs.

Able[13]**,** `http://www.alphaworks.ibm.com/tech/able`, prend la suite de la technologie des *aglets*. Il s'agit d'une solution complète pour le développement d'agents cognitifs et apprenants. Des éditeurs permettent de créer des agents — ou des composants d'agents, les *AbleBeans* — en combinant des composants

10. *Knowledge Query and Manipulation Language,* un « language » de communication entre agents (plutôt une formalisation de la structure des messages) fondée sur la théorie des actes de langage, [Searle 69].
11. Formé sur les mots *agent* et *applet.*
12. *Remote Procedure Call,* pour une introduction, voir [Riveill *et al.*].
13. *Agent Building and Learning Environment.*

JavaBeans (cf. la section 3.1 *Composants* du chapitre I, page 17) dans un éditeur. La bibliothèque de composants est très fournie et propose des composants permettant aux agents de gérer des arbres de décisions, d'utiliser les logiques floue et booléennes, etc.

Zeus, [Nwana *et al.* 97, Nwana *et al.* 98], est un environnement de développement multi-agent comprenant une plate-forme et une méthode de conception des SMA (cf. section 2.2 page 48). C'est une plate-forme orientée programmation visuelle : une interface utilisateur permet de composer ses agents et son SMA en majeure partie par des manipulations simples (clics souris, choix dans des menus, etc.).

Elle est avant tout dédiée aux applications requérant peu d'agents (moins de vingt), ces agents étant des agents plutôt cognitifs que réactifs. La plate-forme n'est pas suffisamment rapide pour gérer ces derniers : le pas de temps moyen d'une application (*i. e.*, le temps écoulé entre deux synchronisations entre les agents) est de trente secondes. Elle correspond très bien aux applications des SMA aux échanges sur l'Internet (commerce, recherche, assistanat, etc.).

Volcano, [Ricordel 01], est une plate-forme doublée d'une méthode de développement *Voyelles* (cf. *infra*) proposées par P.-M. Ricordel. La méthode développement se veut couvrir toutes *(sic)* les étapes du cycle de vie d'une application logicielle — analyse, conception, développement (*i. e.*, mise en œuvre) et le déploiement — en les découpant grâce à la grille de lecture *Voyelles.*

Développée d'abord à partir de mask (cf. *infra*), la méthode s'est vue pourvue d'une plate-forme : Volcano. Les principes fondateurs de Volcano sont le découpage *Voyelles,* les composants et les langages de description d'architecture, qui permettent de décrire des composants et leurs interactions. Volcano propose différents outils — notamment des éditeurs —, pour les différentes étapes couvertes par la méthode (*Vésuve, Madel, Riba* ou *Lava*).

MadKit, [Gutknecht & Ferber 98, Gutknecht & Ferber], implante le modèle aalaadin « agent–groupe–rôle » (cf. *infra*) et permet de gérer facilement la communication entre les agents. Chaque agent a son propre fil d'exécution, celui-ci étant géré soit par la machine virtuelle Java, soit par un autre agent (ce qui permet d'avoir beaucoup de ces agents légers, pour une simulation par exemple).

Le principe de MadKit est de proposer un micro-noyau qui fournit tous les services nécessaires aux agents (gestion des groupes et des rôles, réalisation des communications, migration, etc.). Les services rendus par le micro-noyau sont vus comme des agents, ce qui permet de les remplacer par ses propres agents et donc de changer le comportement du système. MadKit permet aussi, et très facilement, de distribuer les agents sur différentes machines en lançant un micro-noyau sur chacune ; les différents micro-noyaux pourront alors communiquer grâce à des rôles spécialisés (*communicator* pour gérer la communication elle-même et *synchronizer* pour synchroniser la gestion des groupes et des rôles).

MadKit propose un environnement graphique qui permet de visualiser et de contrôler ses agents, sur le modèle des *JavaBeans :* chaque agent fournit un descriptif de ses propriétés (état, fonctions) à cet environnement qui permet alors de les manipuler.

Pour le moment (version 3.0 de la plate-forme), la gestion du modèle organisationnel est limitée : les rôles sont de simples noms et les groupes des ensembles de rôles. Il n'y a aucune vérification sur les permissions qu'un agent pourrait avoir de prendre un rôle ou non. L'utilisation du modèle aalaadin sert donc seulement à l'analyse du système.

Dans moca[14]**,** [Müller *et al.* 01, Amiguet 03], Matthieu Amiguet augmente MadKit pour lui permettre d'intégrer et de vérifier le modèle organisationnel du système, en utilisant le formalisme de Vincent Hilaire (cf. *infra*).

L'utilisation du formalisme de Hilaire permet de décrire des organisations (*i. e.*, dans le vocabulaire du motif *Schémas d'organisation*, des schémas d'organisation), des rôles et des relations entre les rôles.

L'utilisation de MadKit permet d'instancier tous ces concepts par le biais des groupes, des rôles et des accointances.

Moca vérifie et, le cas échéant, effectue les modifications nécessaires pour que l'agent possède le comportement associé aux rôles qu'il prend. Moca utilise un modèle componentiel pour les comportements et ses mécanismes vérifient et assurent aussi que les composants associés à un comportement sont disponibles pour l'agent.

De plus, lorsque les rôles sont décrits par des *state charts,* moca peut gérer, de façon automatique, les conflits qui pourraient exister entre ceux-ci.

En ce qui concerne la communication entre les rôles, moca utilise le motif *Influences* (cf. cet ouvrage, page 105). Les *influences* pouvant aller du *stimulus* réactif aux actes de langage.

Moca semble donc être un modèle et une plate-forme très prometteurs pour les systèmes multi-agents organisationnels.

Mask[15]**,** [Occello & Koning 00], fournit un ensemble de boîtes à outils pour une conception utilisant la grille de lecture *Voyelles :* des architectures d'Agent, des organes sensorimoteurs pour le point de vue Environnemental, des fonctions pour gérer les protocoles d'Interaction et des fonctions pour gérer l'Organisation des agents. Pour chaque compartiment, mask fournit des éditeurs qui permettent d'affiner les systèmes multi-agents d'une manière déclarative. Elle utilise le C++ comme langage de base, accompagné d'une interface en Tcl/Tk[16].

Son principal intérêt est donc de se fonder sur la grille de lecture *Voyelles* et sur les divers outils développés dans le groupe magma.

***Generic Agent* & MAS DK**[17]**,** [Gorodetski *et al.* 02], sont deux outils complémentaires. *Generic Agent* est une bibliothèque de classes *Visual* C++ et Java pour la construction d'agents. MAS DK propose des éditeurs pour décrire facilement, grâce à des langages formels (logique modale en particulier), des systèmes multi-agents : ontologies, classes d'agents, scénarios comportementaux, gabarits de messages KQML, etc. Les agents sont des machines à états finis et à base de connaissances (une évolution vers le BDI est envisagée).

14. Modèle Organisationnel et Componentiel de systèmes multi-Agents.
15. *Multi-Agent System Kernel.*
16. *Tool Command Language/ToolKit.*
17. *MultiAgent System Development Kit.*

L'intérêt de ces outils est l'intégration des langages formels, ce qui les rend propres à intégrer les modèles développés dans le cadre des agents formalisés (notamment BDI et logique temporelle).

1.5. Conclusions sur les outils

Comme on peut le voir par ce rapide survol, les outils sont tout aussi nombreux que les méthodes, les modèles ou les architectures d'agent. Ils supportent une ouverture variée : certains, comme *Concurrent MetateM* ou jack, forcent les modèles d'agent, les modèles interactionnels ou les modèles d'environnement ; d'autres, comme les kits de communication, ne font aucune supposition sur l'architecture interne des agents et déclarent donc supporter tout type d'agent.

Ainsi, pour ces derniers outils, nous rangeons-nous du côté de Les Gasser lorsqu'il dit :

> *"All the talk about tools like Agent-TCL, Java — most if not all of the proliferating 'agent building frameworks' currently available — ignore this issue of persistence of action and the basic requirement for a search-oriented approach. It seems to me that what many current agent frameworks do is already available as distributed objects — there's no really fundamental advance and probably not a quantum leap of added value for system building either."*

Le désir d'universalité de ces outils les empêche d'intégrer les mécanismes nécessaires à la mise en place de l'apport réel du paradigme agent : la finalité (ou « l'action persistante »), elles se reportent donc souvent sur les mécanismes de distribution et de mobilité et n'apportent alors pas grand'chose aux techniques existantes (notamment corba).

D'autre part, une bonne partie de ces plates-formes est orientée agent — quand ce n'est pas seulement objet —, pas SMA ; c'est-à-dire que les concepts utilisés sont individualistes.

—★—

Voyons maintenant la seconde approche, l'approche descendante, qui tente de décrire des méthodes d'analyse–conception agent.

2. Les méthodes de conception

Par un souci de clarté, nous avons classé les différents exemples que nous traitons en quatre groupes :

1. les méthodes utilisant la notation UML[18] ;
2. les méthodes organisationnelles, fondées sur les notions de rôle et d'organisation ;
3. les méthodes formelles, qui utilisent des formalisations mathématiques poussées ;
4. les autres méthodes.

18. *Unified Modeling Language,* [Booch *et al.* 01]. Pour une introduction, nous conseillons [Fowler & Scott 00]

2.1. Les méthodes utilisant UML

UML est le produit de la fusion des méthodes de conception objet OMT (James RUMBAUGH), booch (Grady BOOCH) et oose (Ivar JACOBSON). Cette notation a été créée pour permettre à ceux qui utilisaient les notations des différentes méthodes de conception objet (OMT [RUMBAUGH *et al.* 97], booch, etc.) de pouvoir communiquer. De cette façon, UML intègre les différents diagrammes présents dans ces méthodes et, bien qu'il n'y ait aucune obligation ou même suggestion de le faire, l'utilisation de ces différents diagrammes, et notamment ceux des cas d'utilisation, incite à utiliser une méthode proche de celles des auteurs (G. BOOCH, J. RUMBAUGH et I. JACOBSON). UML est un langage, ce n'est donc pas seulement une notation, c'est avant tout une description d'un modèle objet. Ceci implique qu'en utilisant ce langage, on ne peut complètement oblitérer le modèle objet intrinsèque.

Depuis l'apparition d'UML en 1997, l'envie d'utiliser une notation normalisée et claire pour décrire les systèmes multi-agents s'est faite plus urgente. UML est donc le premier candidat puisqu'il est dorénavant la norme pour la programmation objet. Plusieurs travaux ont vu le jour pour vérifier l'applicabilité d'UML à la description de SMA, et pour, le cas échéant, proposer des augmentations ou des spécifications d'UML pour le rendre apte à décrire des SMA.

L'extension AUML, [ODELL *et al.* 00], se propose d'utiliser et d'augmenter UML pour permettre la gestion des protocoles et plus précisément les protocoles d'interaction entre agent (AIP, *agent interaction protocol*). Ces AIP sont des modèles de communication composés d'une séquence potentielle de messages entre les agents et des contraintes sur les concepts utilisés par ces messages. AUML étend donc UML :

– en spécifiant les protocoles de communication comme des gabarits de paquetages *(package templates)* qui contiennent des diagrammes de séquence paramétrés (par des types de message ou par des sous-protocoles) ;

– en ajoutant des symboles pour le diagramme de séquence pour indiquer la concurrence (a), le choix, inclusif (b) ou exclusif (c), dans les actes de communication des agents (cf. figure 7 page suivante) ;

– en étant plus clair sur la notion de rôle : en indiquant les rôles des agents dans les diagrammes ;

– en spécifiant les objets/agents par leur rôle plutôt que par leur classe structurelle ;

– en utilisant des agents comme points d'interface entre les paquetages plutôt que des opérations (de fait, les paquetages représentant des protocoles, ce sont en effet les agents qui sont l'interface entre les protocoles) ;

– en ajoutant la notion de mobilité au diagramme de déploiement.

Il s'agit donc avant tout d'une meilleure utilisation d'UML (spécifier mieux et davantage dans les différents diagrammes) qui se traduit par une augmentation du modèle (notions de rôles et de protocoles) plutôt que d'une augmentation de la notation.

La méthode MaSE, [WOOD & DELOACH 00], fondée sur le modèle des cas d'utilisation se décompose en sept phases, trois d'analyse et quatre de conception (cf. figure 8 page 43). La méthode est itérative, c'est-à-dire que l'on peut

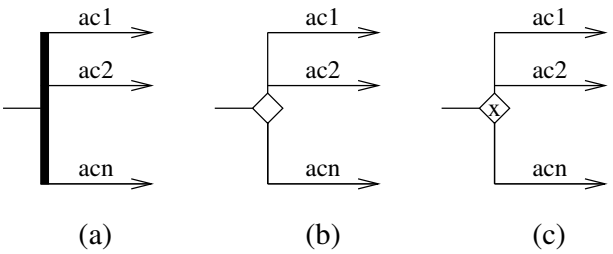


FIG. 7 – Symboles ajoutés par AUML aux diagrammes de séquence

la ré-appliquer autant de fois que nécessaire pour que tout soit spécifié. Elle utilise aussi la notion de *rôle,* ce qui aurait pu la classer dans le groupe suivant, mais l'aspect *use-case* est prépondérant.

À partir du cahier des charges, la phase de capture des buts fait l'analyse fonctionnelle du système. Ensuite, le concepteur construit les *conversations* entre les agents — bien que ceux-ci ne soient pas encore définis — à l'aide de cas d'utilisation. La phase d'affinage des rôles fait d'abord correspondre un rôle avec un but pour ensuite fusionner en un seul rôle les rôles compatibles (*i. e.*, à buts non antagonistes). Vient ensuite la création des classes, c'est-à-dire l'élaboration du diagramme de classes dans lequel les classes sont associées par les liens décrits dans les conversations ébauchées plus tôt ; de la même façon que les rôles précédemment, les classes sont fusionnées lorsqu'elles sont compatibles (généralisation). Les conversations sont alors mises en forme en tant que protocoles de coordination entre deux agents. L'assemblage des agents se fait alors de manière componentielle, c'est-à-dire à l'aide de composants prédéfinis (cf. la section 3.1 page 17) ce travail est facilité par le fait que, pour WOOD & DELOACH, il existe cinq types d'architectures : réactive, BDI, planificatrice, à base de connaissances et *user-defined (sic).* La dernière phase de cette méthode consiste en l'écriture du diagramme de déploiement des agents sur les machines supportant le système. L'outil *agentTool* faciliterait l'application de cette méthode, surtout dans ses dernières phases.

Cette méthode pose plusieurs conditions à son application :

– le monde est fermé, il n'y a pas de création, de destruction ou de déplacement d'agent pendant l'exécution du système ;

– il n'y a pas de possibilité de messages en *multicast*, ce qui peut toutefois être simulé par le point à point ;

– cette méthode deviendrait difficilement applicable au delà de dix classes d'agents.

À ces conditions d'utilisation posées par les auteurs, nous devons adjoindre certaines remarques. Les conversations se limitent à deux agents, bien qu'une conversation à trois agents ou plus ne soit pas interdite, elle n'est pas envisagée dans cette méthode. De plus, l'assemblage des agents n'est pas décrit et est laissé à l'appréciation du concepteur, qui choisira son type d'architecture et donc ses composants. Le problème ici est que nous ne pensons pas que le modèle interne de l'agent (son architecture) soit si peu important dans la conception du système et des interactions.

Les règles de transformation de graphes, [DEPKE *et al.* 00], sont à la base d'une autre méthode de modélisation orientée agent utilisant la notation

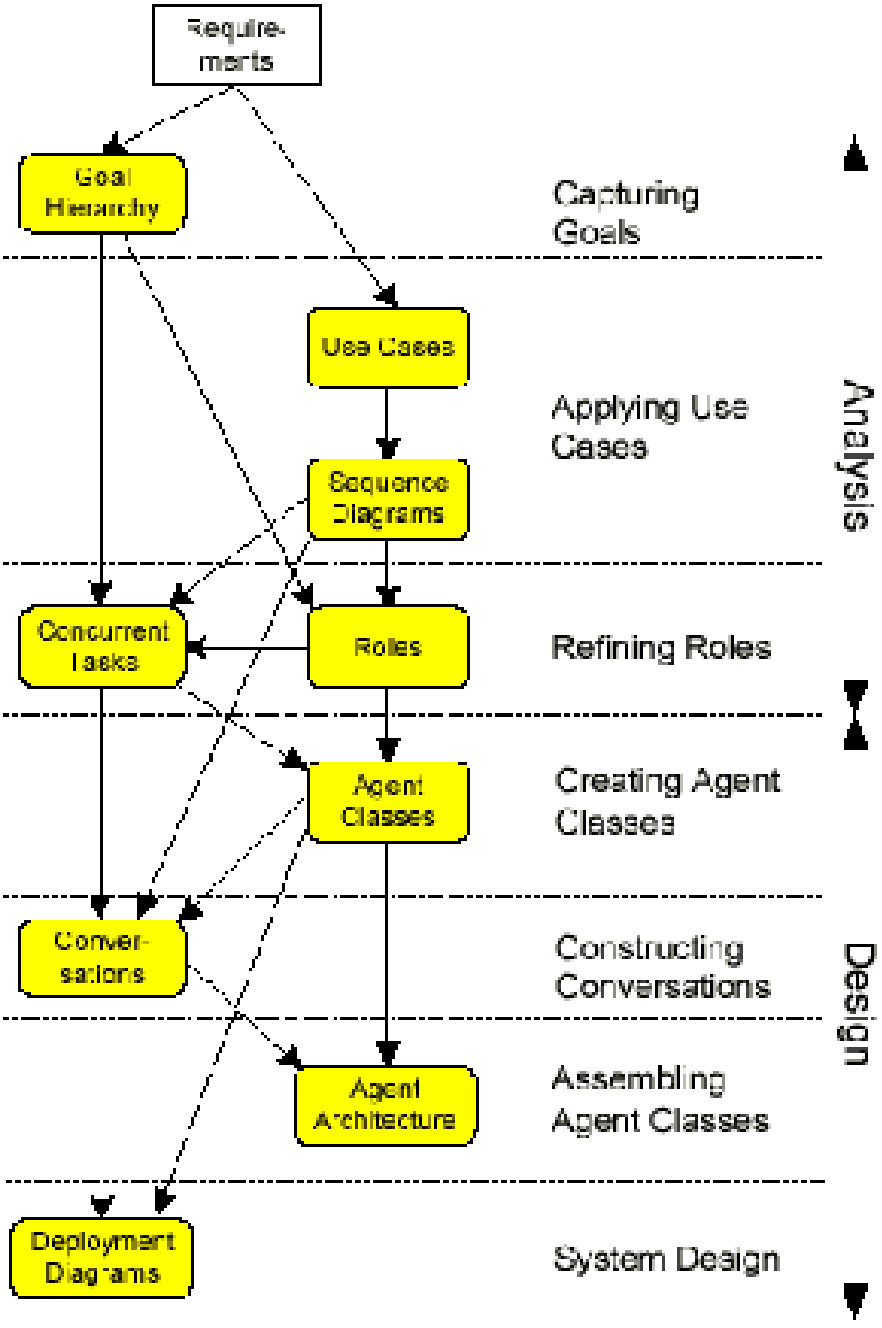


FIG. 8 – La méthode MaSE, [WOOD & DELOACH 00, fig. 1]

d'UML. Ces règles de transformation de graphes sont des règles *si a alors b* où *a* et *b* sont deux graphes représentant, pour *a*, une partie de l'état courant du système (un motif à reconnaître) et, pour *b*, le nouvel état du monde après application de la règle.

La méthode elle-même consiste en les trois phases classiques :

a) Analyse des besoins : cette phase utilise l'approche des cas d'utilisation. Elle décrit le système sous forme de scénarios globaux, représentés par des cas d'utilisation, des diagrammes de séquence et des diagrammes de transformation de graphes, représentant, sous la forme de diagramme d'instances, les modifications des états du système à l'application de ces scénarios.

b) L'analyse comporte trois modèles :

– *un modèle structurel,* spécifiant les types d'agents, leurs attributs et leurs associations sous la forme d'un diagramme de classes ;

– *un modèle fonctionnel,* représentant un affinement des scénarios décrit dans la première phase, utilise des diagrammes de transformations de graphes (ces règles peuvent encore rester incomplètes et vagues pendant cette phase) ;

– *un modèle dynamique,* utilisant des diagrammes de séquence pour représenter les flux de messages de chaque scénario.

c) La phase de conception, qui comporte elle aussi trois modèles :

– *le modèle structurel* est un affinement du diagramme précédent ;

– *le modèle dynamique* utilise des diagrammes d'états pour représenter les protocoles d'interaction ;

– *le modèle fonctionnel* complète et spécifie entièrement les diagrammes de transformations de graphes du modèle fonctionnel de la phase d'analyse.

L'extension apportée à la notation par cette méthode concerne les diagrammes de cas d'utilisation et les diagrammes de classes. Dans les cas d'utilisation, les agents sont représentés par des acteurs à tête carrée avec leurs buts dans des petits nuages au dessus de celle-ci. Les classes d'agents, quant à elles, possèdent un cadre en gras pour noter le stéréotype « proactif » et une case supplémentaire qui contiendra les messages de ces agents — la différence entre un message et une opération est que c'est l'agent qui choisit les opérations qu'il active alors que les messages lui sont envoyés de l'extérieur (par les autres agents, en mode synchrone) ; les messages sont donc ceux qui apparaissent dans les diagrammes de séquence.

Les auteurs pensent aussi intégrer la notion de *rôle* dans les prochaines utilisations de leur méthode. Les rôles, contrairement aux agents qui les jouent, ne participeront qu'à une transaction à la fois. Cela facilitera le passage de l'analyse à la conception, les interactions pouvant être développées une à une. Il faudra ensuite synchroniser les diagrammes d'états de tous les rôles joués par un même agent, ce qui est le principal problème de la décomposition du comportement d'un agent en automates (problème qui s'était posé à nous lors de l'application du modèle organisationnel de Benoît Durand, cf. *infra*).

Les diagrammes d'activité d'UML sont le nœud central de la proposition de Cristoph Oechslein, Franziska Klügl, Rainer Herrler et Franck Puppe, [Oechslein *et al.* 02]. Ils proposent en effet d'utiliser UML dans son état actuel, celui-ci leur semblant presque suffisant.

Les diagrammes de classes représentent, pour chaque type d'agent, les aspects visibles par les autres agents : les attributs sont les états internes de l'agent accessibles de l'extérieur, les méthodes représentent les interactions possibles de l'agent.

Les diagrammes d'activité permettent de représenter le comportement individuel de chaque agent, chacune des interactions possibles de celui-ci. De plus, UML autorise la transmission de signaux entre les nœuds des diagrammes d'activité. Une utilisation efficace de cette possibilité (avec la participation de commentaires et d'objets-paramètres) permet de décrire les interactions entre les comportements des agents.

Enfin, l'utilisation d'OCL [19], le langage de contraintes intégré à UML, permet de spécifier les invariants et les contraintes du système lors de ces interactions.

Bien que n'adhérant pas entièrement au choix d'AUML d'utiliser les diagrammes de séquence, Oechslein *et alii* ont en fait la même optique que les concepteurs d'AUML, à savoir mieux utiliser UML pour lui permettre d'être utilisé comme notation et modélisation des systèmes multi-agents.

2.2. Les méthodes organisationnelles

Ces méthodes partent du principe qu'un système multi-agent est un ensemble d'agents autonomes, possédant chacun un ou plusieurs buts, et, surtout, socialement situés : ils maintiennent des interactions organisées entre eux. Ainsi, la notion de *rôle* est-elle centrale et commune à toutes ces approches. Nous reparlerons de cette notion dans le motif *Schémas d'organisation*, page 79.

Le modèle organisationnel de B. Durand, [Durand 96], augmenté par [Sauvage 97], est bien plus un *framework* qu'une méthode mais l'utilisation de ce modèle implique l'analyse organisationnelle du système à construire et donc propose une méthode implicite.

Le modèle utilise en fait le métamotif *Schémas d'organisation* décrit en détail page 79. Il est fondé sur les principes de *rôle,* de *schéma d'organisation,* d'*organisation concrète* et de *comportement.*

Chaque point de vue sur une société d'agents est modélisé sous la forme d'un *schéma d'organisation*, qui, instancié, donne naissance à une *organisation concrète*. Un schéma peut être instancié autant de fois que l'on veut, l'effet en sera plusieurs organisations concrètes fonctionnant sur le même modèle, avec des agents différents.

Un agent possédera autant de rôles qu'il y a de schémas d'organisation pour lesquels il est pertinent. Un *rôle* est un ensemble de comportements exécutés dans un schéma d'organisation. Chaque rôle définit un ensemble d'états et un *comportement* pour chacun de ces états. Ainsi, un comportement est-il l'activité d'un agent pour un rôle lorsque celui-ci se trouve dans une situation décrite par l'état en cours de l'agent. Dans ce modèle, un rôle est défini comme un ensemble de comportements. Ces comportements sont de quatre types :

– *proactif :* c'est le comportement de base pour le rôle, exécuté automatiquement par l'agent, c'est son activité de base ;

19. *Object Constraint Language*

– *réactif :* il s'agit des comportements occasionnels, déclenchés par des événements internes à l'agent ;

– *interactif*[20] *:* il s'agit de comportements occasionnels, déclenchés, *intentionnellement*, par des partenaires de l'agent (c'est une augmentation du modèle originel de B. DURAND décrite dans [SAUVAGE 97]) ;

– *fonctionnel :* ces activités sont réalisées par un agent lorsque l'un de ses partenaires exécute un autre comportement auquel ce comportement fonctionnel est lié, c'est un comportement *esclave*[21].

Un *sous-rôle* d'un rôle R est un rôle fonctionnel R' tenu par un autre agent que celui qui tient R, dans le même schéma d'organisation. Un *rôle associé* à un rôle R est un rôle fonctionnel R' tenu par le même agent mais dans un autre schéma d'organisation.

Ces deux relations sont proches du point de vue représentation et mise en œuvre mais elles diffèrent au niveau conceptuel. Tout d'abord, la première relie deux agents différents et la seconde ne concerne qu'un seul agent, ensuite, et surtout, parce que dans la seconde il y a deux schémas d'organisation distincts. Donc, la seconde relation est une passerelle entre deux points de vue différents sur le système alors que la première réifie la cohésion du point de vue décrit dans le schéma d'organisation.

La perception est un domaine important du modèle d'agent. Son mécanisme est primordial dans l'activité du système. En fait, il se décompose en deux parties : la *perception* proprement dite, c'est-à-dire l'activité de *scrutation* des *organes* sensitifs, et son inséparable *alter ego* l'*aspect*, c'est-à-dire l'activité du paraître. Notez qu'il s'agit bien là d'une activité, car on ne laisse voir que ce que l'on veut bien montrer, surtout en informatique où tout doit être stipulé, déclaré, préparé ou prévu, et encore plus en POO où les données sont encapsulées et où les mécanismes d'accès doivent être rigoureusement protocolaires. Ainsi, dans la définition d'un rôle, prévoit-on les différents aspects que montrera l'acteur et que les autres agents pourront scruter.

La méthode induite par ce modèle est donc un découpage du système en organisations, composées de rôles, et qui seront abstraites en schémas d'organisation. Les rôles sont ensuite découpés en comportements. Les agents sont catégorisés en classes suivant les rôles qu'ils peuvent prendre. Les comportements sont alors ajustés, coordonnés, pour qu'un agent puisse les exécuter sans incohérence.

Le principal problème de ce modèle est qu'il est réflexif et donc assez confus. En fait, cette réflexivité vient du fait que Benoît DURAND, ne disposant pas du parallélisme, a voulu que le modèle puisse s'auto-exécuter ; il est donc son propre modèle d'exécution. C'est cette propriété qui a attiré Vincent HILAIRE qui a repris ce modèle (cf. la méthode de [GRUER *et al.* 00], décrite ici en page 52).

Un problème secondaire de ce modèle — mais qui est inhérent aux modèles utilisant des automates — est la difficulté de synchroniser les différents automates représentant les comportements concurrents de l'agent (en particulier la gestion des conflits créés par la présence de différents rôles dans l'agent).

20. Le préfixe « inter » sert à montrer la présence d'un autre agent, il n'y a pas d'interaction à proprement parler, mais ces comportements sont à la base des interactions.
21. D'où une relation maître/esclave entre les rôles (et donc entre les comportements).

Le modèle aalaadin, [Gutknecht & Ferber 98], qui est aussi proposé comme une méthode d'analyse–conception, reprend, en le simplifiant, le modèle de Benoît Durand dont nous venons de donner les grandes lignes. Le modèle proposé sépare les concepts en deux niveaux : le niveau méthodologique et le niveau descriptif (cf. fig. 9).

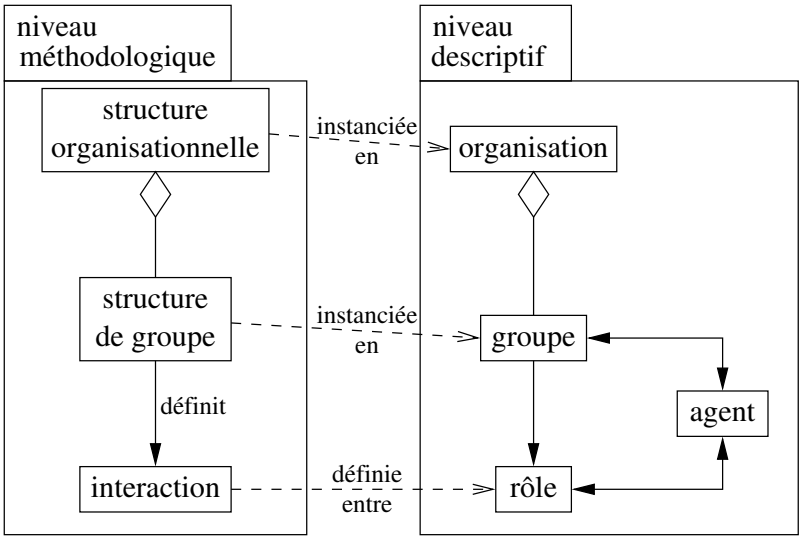


Fig. 9 – Les concepts du modèle aalaadin

Les concepts fondamentaux d'aalaadin sont ceux d'*agent,* de *groupe* et de *rôle.* Un *groupe* est un ensemble d'agents communiquant entre eux — en principe, un agent ne peut communiquer qu'avec les agents des groupes auxquels il appartient ; le *groupe* est donc un concept de niveau instance (ce que les auteurs appellent le niveau descriptif). Une *organisation,* au sens d'aalaadin, est un ensemble de groupes liés entre eux par des agents qui jouent un rôle dans un des groupes et un autre dans un autre groupe (ces agents sont des *représentants*).

Si l'on veut comparer ces concepts avec ceux du modèle précédemment décrit de Benoît Durand, une *organisation concrète* correspond à une *organisation,* décomposée en *groupes.* Et, de la même manière, au *schéma d'organisation* correspondra une *structure organisationnelle* composée de ses *structures de groupe.*

Il est à noter que c'est le concept de *groupe* et non d'*organisation* qui est fondamental pour Olivier Gutknecht & Jacques Ferber ; les organisations se résumeraient à des groupes d'agents qui peuvent communiquer entre eux. Or, comme le fait justement remarquer Anne Nicolle dans [Nicolle 02], les organisations sont plus que de simples groupes.

En ce qui concerne la méthode proposée, il s'agit d'analyser le système comme formé de groupes de rôles. Il faut donc décrire les rôles présents dans le système et les agglomérer dans des groupes. Les structures de groupes permettent d'abstraire les groupes et donc à la fois de les découvrir et de les décrire.

Cette méthode est donc assez embryonnaire pour le moment. De plus le modèle « agent–groupe–rôle » nous semble une simplification excessive des organisations sociales.

La méthode Gaia, [Wooldridge *et al.* 00], repose sur le concept de rôle, un SMA étant vu comme une organisation de rôles. En fait, les concepts utilisés sont individuels :

– *rôle :* il s'agit de l'acception habituelle du terme ;

– *permissions :* ce sont des droits de l'agent sur les ressources utilisées par un rôle ;

– *responsabilités :* ce sont les buts que le rôle implique, elles sont séparées en propriétés de vivacité *(liveness properties)* et de sécurité :

  – les premières concernent les désirs de l'agent, les faits qu'il a envie de voir devenir vrais,

  – et les secondes concernent les invariants qu'il doit maintenir ;

– *activités :* les opérations internes de l'agent ;

– *protocoles :* les protocoles d'interaction nécessaires au rôle.

Les concepts précédents étant considérés comme abstraits, il existe aussi des concepts dits concrets : les *types d'agent,* les *services* et les *accointances,* qui correspondent aux acceptions courantes de ces termes.

La méthode se décompose en cinq modèles (ou plutôt diagrammes) :

– *le diagramme de rôles :* il décrit les différents rôles de l'organisation suivant les attributs que sont les permissions, les responsabilités, les activités et les protocoles ;

– *le diagramme d'interactions :* il décrit les protocoles qui seront utilisés dans le SMA ;

– *le diagramme d'agent :* spécifie quels types d'agents et quelles instances d'agents évolueront dans le système ;

– *le diagramme de services :* il décrit les services nécessaires pour chacun des rôles ;

– *le diagramme d'accointances :* liste les liens de communication qui existent entre les rôles.

Les liens entre ces diagrammes sont résumés par la figure 10 page suivante.

La méthode consiste donc à identifier les rôles du système (prototype du diagramme de rôles), puis les protocoles associés à chaque rôle (diagramme d'interactions). À partir du diagramme d'interactions, le diagramme de rôles est affiné. Cette phase d'analyse est itérée jusqu'à spécification complète des diagrammes d'analyse. En ce qui concerne la conception, les trois derniers diagrammes sont remplis en utilisant les informations contenues dans les deux premiers.

Les points faibles de cette méthode, d'ailleurs soulignés par les auteurs, sont son statisme et sa fermeture à divers types de SMA et d'agents, notamment les SMA évolutifs ou réactifs ou les agents compétitifs. De plus, comme nous le disions plus haut, le modèle sous-jacent est très individuel : il ne s'intéresse aux rôles qu'un par un, il suppose qu'il n'y a qu'une unique organisation dans tout le système, alors que, comme nous le verrons dans le motif *Schémas d'organisation* (page 79), le découpage du système en différentes organisations est bénéfique à l'analyse et à la conception du SMA.

Les *role-models*, forment la base de la méthode d'analyse–conception de la plate-forme Zeus de *British Telecom* (cf. page 38), [Kendall 98a]. Il s'agit ici d'utiliser les prototypes de rôles déjà définis comme base d'analyse du sys-

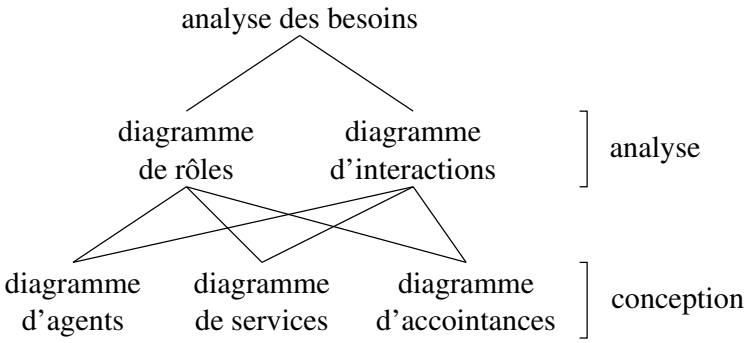


FIG. 10 – Les différents diagrammes de Gaia

tème. Cette méthode est en fait très orientée objet puisqu'elle a été conçue pour la POO : [KRISTENSEN 95, KRISTENSEN & ØSTERBYE 96, KRISTENSEN & OLSSON 96].

La méthode « agent » de Zeus se compose de quatre phases :

1. *analyse domaine :* suivant le domaine du problème, les différents rôles du système sont identifiés ;

2. *conception :* à ce stade, on identifie l'ontologie, les agents, les services et les accointances ;

3. *réalisation :* les agents sont définis, ainsi que leur coordination et l'organisation du système ;

4. *exécution :* la plate-forme fournit des outils de visualisation et de débogage de l'application.

Cette méthode reste donc très objet. Toutefois, le catalogue des prototypes de rôles (les *role-models* du *Role Modelling Guide,* [COLLIS & NDUMU 99]) peut servir de base à un catalogue de motifs organisationnels, tel que nous le décrivons dans le motif *Schémas d'organisation*.

La méthode soda[22]**,** [OMICINI 00], est fondée sur le concept de tâche et est spécifiquement orienté vers les systèmes évoluant sur l'Internet. La conception interne des agents n'est pas abordée, la méthode suppose des agents hétérogènes — et situés.

L'analyse utilise trois diagrammes :

– *le diagramme de rôles :* spécifie les tâches à effectuer pour chaque rôle ou chaque groupe, une tâche est un ensemble de responsabilités (buts), de ressources nécessaires et des compétences requises, les tâches dites sociales seront assignées à des groupes ;

– *le diagramme de ressources :* le système est défini en tant que services disponibles, chacun associé à un ensemble de ressources, de permissions et de moyens d'accès ;

– *le diagramme d'interaction :* les interactions entre les rôles, les groupes, et les ressources impliquées sont modélisées sous la forme de protocoles, eux-mêmes spécifiés par l'information requise et fournie par chaque rôle et chaque ressource et par les règles qui gouvernent les interactions à l'intérieur des groupes.

À partir de l'identification des tâches individuelles ou sociales, les rôles et

22. *Societies in Open and Distributed Agent spaces.*

les groupes de rôles sont à leur tour identifiés. Le résultat de l'analyse est donc la spécification des rôles, des groupes et des ressources.

Pour la conception, trois autres diagrammes sont utilisés :

– *le diagramme d'agent :* chaque rôle y est plaqué sur une ou plusieurs classe d'agents ;

– *le diagramme de société :* chaque groupe est représenté par une société d'agents, une société représente un ensemble d'agents regroupés autour d'un médium de coordination qui englobe les abstractions nécessaires à la coordination interne de la société, — les sociétés sont donc réifiées dans ce modèle ;

– *le diagramme d'environnement :* les ressources sont associées aux infrastructures présentes dans le système, une topologie du système y est spécifiée.

Un des points faibles de cette méthode est qu'elle ne s'intéresse pas aux agents en particulier mais seulement à leurs interactions (ce qui est par contre son point fort, notamment en ce qui concerne la coordination des agents).

Franco Zambonelli, Nicholas jennings & Michael wooldridge, [Zambonelli *et al.* 00b], ébauchent une méthode d'analyse et de conception fondée sur les concepts organisationnels. Les auteurs relèvent bien les déficiences des différentes méthodes organisationnelles proposées actuellement, notamment le manque d'abstraction au niveau organisationnel, les organisations n'étant vues que comme un ensemble de rôles (ce que nous avons relevé ici comme étant l'individualisme des concepts de Gaia ou lorsque nous indiquions le caractère objet des *role-models* de Zeus).

Accompagnés d'Andrea Omicini, [Zambonelli *et al.* 00a], les mêmes auteurs ébauchent une autre méthode de conception de SMA, cette fois-ci fondée sur la coordination, une des lacunes de Gaia. Il s'agit donc d'un mélange de Gaia et de soda (deux méthodes présentées plus haut).

Comme on peut le voir sur la figure 11 page suivante, les diagrammes utilisés proviennent en grande partie de la méthode Gaia (cf. *supra*), la méthode est donc essentiellement la même que Gaia. Par contre, en ce qui concerne les interactions, pour pallier les lacunes de Gaia en matière de coordination, c'est la méthode soda qui est utilisée, grâce à l'adjonction de deux nouveaux diagrammes : les lois sociales et le comportement du médium d'interaction (qui remplace le diagramme d'accointances, moins précis).

Cette méthode utilise un modèle de coordination composé :

– *de coordonnables :* les entités qui peuvent entrer en coordination ;

– *du médium d'interaction :* l'ensemble des abstractions permettant les interactions des agents entre eux, ainsi que la structure autour de laquelle les composants sont organisés (*e. g.*, de manière simple : des sémaphores, ou, de façon plus évoluée : un tableau noir, etc.) ;

– *des lois de coordination :* décrivent le comportement du médium de coordination en réponse aux événements d'interaction.

Cette méthode est donc une bonne intégration des points forts des méthodes Gaia et soda. Toutefois, la conception des agents eux-mêmes n'est toujours pas abordée : seuls leurs comportements sont spécifiés, on ne connaît pas de méthode pour les mettre en œuvre.

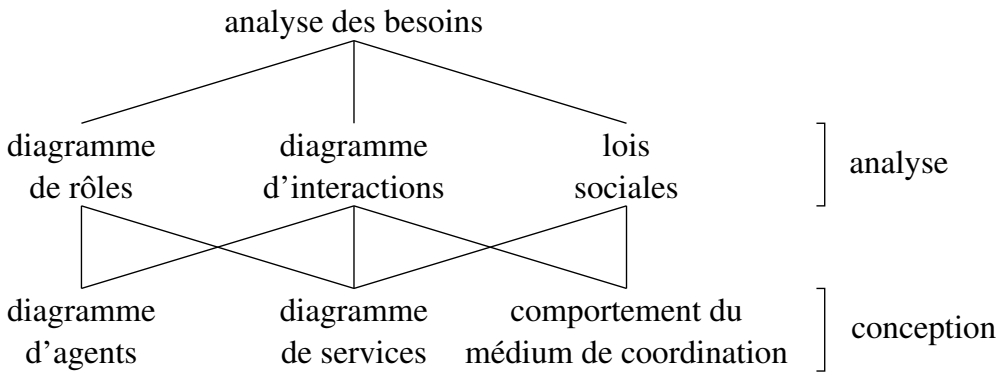


FIG. 11 – Les diagrammes proposés par [ZAMBONELLI *et al.* 00A]

La méthode Cassiopée, [COLLINOT & DROGOUL 98A, COLLINOT & DROGOUL 98B], se spécialise dans le comportement collectif (l'application première ayant été la conception d'une équipe de robots footballeurs pour la compétition *simulation* de la RoboCup). Les rôles des agents y sont de trois types :

– les *rôles du domaine,* ou *rôles élémentaires,* correspondant aux tâches spécifiques à accomplir ;

– les *rôles relationnels* qui traitent des interactions entre les agents ;

– les *rôles organisationnels* qui organisent les agents et leur permettent de coopérer.

La démarche de Cassiopée part de la tâche collective à accomplir, définit les tâches du domaine nécessaires à l'accomplissement de cette tâche collective, décrit la structure organisationnelle (*e. g.*, hiérarchique, distribuée, etc.) du système et les rôles relationnels qu'elle entraîne, puis définit l'évolution de cette structure et donc les rôles organisationnels gérant cette évolution.

Comme bon nombre des méthodes décrites ici, Cassiopée explicite les différents comportements des agents grâce à une analyse–conception du système mais leur mise en œuvre est laissée aux bons soins du développeur.

La méthode Andromeda, [DROGOUL & ZUCKER 98], construite à partir des expériences de Cassiopée, se différencie de celle-ci par le fait qu'une des contraintes de base au développement d'Andromeda a été l'intégration de l'apprentissage automatique à la conception multi-agent.

Pour cela, Andromeda propose de se fonder sur les buts, et non plus sur les comportements ou les fonctions. Cela devrait permettre une meilleure adaptabilité des agents : leur comportement n'est plus aussi figé ou dirigé, il est plus souple. Considérer les buts plutôt que la façon de les réaliser permet justement de laisser le système apprendre comment les réaliser.

À l'aide de différentes techniques d'apprentissage (apprentissage par renforcement, d'algorithmes génétiques, de réseaux neuronaux, d'apprentissage relationnel, etc.), Andromeda construit les différents rôles du système.

La première phase définit les comportements individuels (rôles du domaine pour Cassiopée). La deuxième phase relève les influences entre les rôles. La troisième phase permet de construire les rôles relationnels. La quatrième phase apprend à grouper les agents (en regroupant les rôles interdépendants). La cinquième phase définit les rôles organisationnels de Cassiopée.

Ainsi, Andromeda reprend-elle les phases de Cassiopée en y intégrant différentes techniques d'apprentissage. D'ailleurs, peut-être utilise-t-elle trop de méthodes d'apprentissage différentes et, comme il est nécessaire de bien les comprendre pour bien les utiliser et qu'il est difficile d'être à la fois expert en apprentissage, en conception de SMA et dans le domaine visé par l'application, peut-être est-elle trop difficile d'accès.

2.3. Méthodes formelles

Desire[23]**,** [Brazier *et al.* 95a, Brazier *et al.* 98c], est présentée comme un cadre d'analyse, de conception et de mise en œuvre de SMA mais elle traite surtout de modèles, sous une formalisation logicienne.

La notion de base y est la tâche. Chaque agent possède entre trois et huit tâches concurrentes :

- le contrôle de son propre processus ;
- l'interaction avec les autres agents ;
- le maintien de ses connaissances des caractéristiques des autres agents ;
- les interactions avec le monde extérieur ;
- le maintien de ses connaissances sur le monde extérieur ;
- le maintien d'un historique de ses observations et interactions passées ;
- la gestion de la coopération (projets, contrats) ;
- les tâches spécifiques de l'agent *(sic)*.

La méthode associée est donc une décomposition hiérarchique des tâches :

- (les tâches sont identifiées ;)
- les tâches sont décomposées ;
- les échanges d'information nécessaires aux tâches sont identifiés ;
- les tâches sont organisées en séquences ;
- certaines tâches sont déléguées d'un agent à un autre ;
- les structures de connaissance nécessaires à chaque tâche sont identifiées et décrites.

Cette méthode est, comme on peut le voir, très orientée tâche et ressemble beaucoup à ce qui se fait dans les KBS[24]. Elle nous semble ne pas prendre en compte un grand nombre des caractéristiques d'un SMA, comme la socialité et la proactivité des agents.

Pablo Gruer, Vincent hilaire & Abder koukam, [Gruer *et al.* 00], reprennent en fait le modèle de Benoît Durand décrit plus haut (page 45), en évitant toutefois sa réflexivité inutile. L'utilisation du langage *Object-Z* pour décrire le système permet à la fois sa vérification automatique et son exécution.

Le principal intérêt de cette méthode est aussi son principal désavantage : le formalisme. En effet, si celui-ci permet la vérification automatique du comportement du système, et même son exécution, il demande au concepteur d'être à la fois un expert du domaine et un expert du formalisme.

23. *DEsign and Specification of Interacting REasoning components.*
24. *Knowledge Based System* ou Systèmes à bases de connaissances.

Giovanna Di marzoserugendo, [Di Marzo Serugendo 00], présente une méthode formelle de développement et de validation fondée sur les réseaux de Petri, la conception par contrats et la logique de Hennessy-Milner. Le méthode est composée de sept phases, chacune engendrant un contrat (un ensemble de propriétés à fournir) :

1. *analyse informelle des besoins :* il s'agit d'une vue informelle du système à réaliser ;
2. *fonctionnalité du système :* le résultat de cette phase est l'ensemble des fonctions à réaliser par le système, il s'agit d'une vision abstraite du système, sans référence aux agents ;
3. *collections du système :* le système est vu comme un ensemble de collections (équipe d'agents) et de relations entre ces collections, le résultat de l'application de cette phase nous donne les fonctions à réaliser par les équipes d'agents et les propriétés de compositions de ces équipes ;
4. *conception des collections :* c'est ici que les collections sont découpées en agents et que les messages et les dépendances sont identifiés, on obtient ainsi la fonction de chaque agent et les propriétés des interactions ;
5. *conception des agents :* cette phase consiste en la description du comportement interne des agents : algorithmes, données internes, etc. ;
6. *moyens de communication :* cette phase est plus concrète que les précédentes puisque l'on y définit les moyens de communication bas niveau (tels RMI, corba, etc.) ;
7. *mise en œuvre :* le contrat de la sixième phase est exprimé dans le langage de communication choisi.

Cette méthode considère donc chaque phase comme une réécriture et un complètement du contrat de la phase précédente. Cette méthode nécessite aussi la maîtrise des réseaux de Petri de type CO-OPN/2 et de la logique modale de Hennessy-Milner. Il est aussi bon de rappeler que les réseaux de Petri ne sont pas dynamiques : l'ajout d'un élément dans le problème nécessite une réécriture totale du réseau correspondant.

2.4. Les autres méthodes

M. Elammari & W. lalonde, [Elammari & Lalonde 99], proposent une méthode orientée agent utilisant les *Use Case Maps* (UCM) comme base de la notation. Un UCM représente un scénario sous une forme graphique (cf. fig. 12, chaque case y représente un agent, un composant du système, le *chemin* y représente l'enchaînement des *responsabilités*, correspondant aux opérations du scénario).

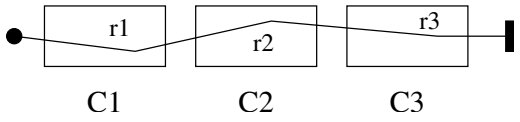


Fig. 12 – Exemple de *use case map*

Cette méthode produit cinq modèles (diagrammes) qui sont dérivables les uns des autres de la façon décrite par la figure 13 page suivante.

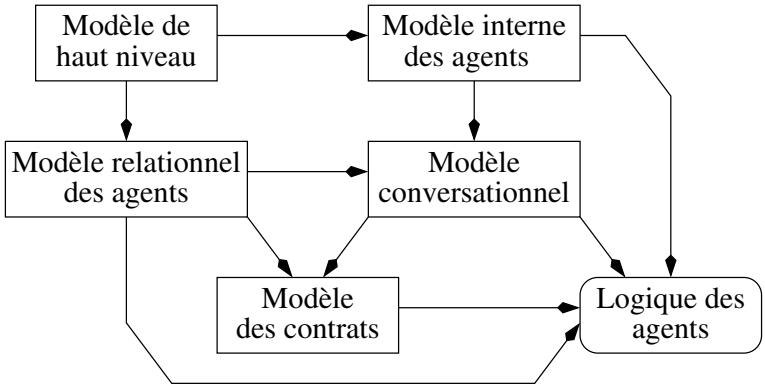


FIG. 13 – Les modèles produits par la méthode ELAMMARI–LALONDE

a) Le modèle de haut niveau permet d'avoir une vue générale du système, il contient les scénarios qui décrivent le comportement fonctionnel. Les agents et les comportements sont identifiés pendant cette phase. Les UCM y décrivent :

– les conditions de garde de toutes les étapes des scénarios ;

– les responsabilités à remplir (les buts à atteindre par ce scénario) ;

– les changements d'état du système suivant la réalisation de ces responsabilités.

Les agents sont identifiés au fur et à mesure de la description des scénarios, en partant des entités émergeant des scénarios. Cinq types d'« agents » *(sic)* sont distingués suivant leur comportement :

– *les acteurs :* représentent les entités physiques (*e. g.*, humains, ressources physiques) ;

– *les gestionnaires :* sont responsables de la gestion de groupes d'acteurs ;

– *les tâches :* les agents simples à la fonctionnalité spécialisée et limitée ;

– *les programmes :* ces agents représentent un programme externe ou leur servent d'interface ;

– *les rôles :* l'acception habituelle du rôle (*i. e.*, une fonction qu'un ou plusieurs agents peuvent prendre dans une organisation).

b) Le modèle interne des agents définit les agents suivant leurs buts, leurs plans, leurs croyances... Chaque agent est décrit dans une table contenant, pour chaque but, ses pré-conditions, ses post-conditions et les tâches et sous-buts nécessaires à sa réalisation.

c) Le modèle relationnel décrit les relations de dépendances et *juridictionnelles.* Les dépendances correspondent aux services rendus par les agents les uns aux autres. Un graphe des dépendances permet de spécifier les agents en cause et s'il s'agit de services correspondant à des tâches ou à des sous-buts, s'ils sont nécessaires (ressources) ou négociés. Le graphe juridictionnel décrit la hiérarchie de délégation des tâches.

d) Le modèle conversationnel s'intéresse à la coordination entre les agents et aux messages qu'ils peuvent s'échanger. Les différents types de dépendance ont chacun des types prédéfinis de messages, à la manière des actes de langage de KQML :

– pour une dépendance négociée, les messages peuvent être des *propositions,*

des *contre-propositions,* des *rejets* et des *acceptations* ;

– une dépendance de ressource implique des *requêtes* et des *informations* ;

– une dépendance de but (un sous-but), provoque une *réalisation* ;

– une dépendance de tâche (une délégation), est réalisée par un message d'*exécution* ;

– une dépendance juridictionnelle (ou d'autorité), utilise des *dires* et des *demandes.*

e) Le modèle des contrats rassemble les engagements que les agents peuvent prendre les uns envers les autres, ces engagements peuvent être pris dès la création des agents ou lorsqu'ils sont utiles, à l'exécution. Chaque dépendance est donc décrite sous la forme d'un contrat qui décrit :

– *les autorisations,* c'est-à-dire les services que chacun des participants met à la disposition de l'autre (pour un certain coût) ;

– *les obligations* que chacun doit remplir pour satisfaire l'autre ;

– *les philosophies,* qui décrivent la qualité, la quantité des services, l'importance qu'ils ont pour chacun, et la façon dont ils doivent être utilisés (*e. g.*, sont-ils obligatoires, facultatifs, que se passe-t-il s'ils échouent, etc.) ;

– *les croyances* que les agents doivent avoir ou partager.

Seul le modèle de haut niveau participe de la découverte du système, les autres phases sont des phases de définition, d'affinement et de réécriture.

Cette méthode et les modèles utilisés se rapprochent fortement du modèle *workflow.*

La grille de lecture *Voyelles*, [Demazeau 95], permet à [Occello & Koning 00] de développer une méthode de conception agent distinguant les points de vue agent, environnement, interaction et organisation. Pour Michel Occello et Jean-Luc Koning, la phase d'analyse consiste à reformuler — suivant des modèles AEIO particuliers et grâce aux principes de récursivité et d'émergence des systèmes multi-agents — le problème considéré en un problème multi-agent. Ensuite, et c'est ici seulement que leur méthode intervient, la conception consiste à résoudre ce problème multi-agent, en utilisant des outils AEIO et les mécanismes de récursivité et d'émergence. Ce point de vue sur le processus de conception est représenté sur la figure 14.

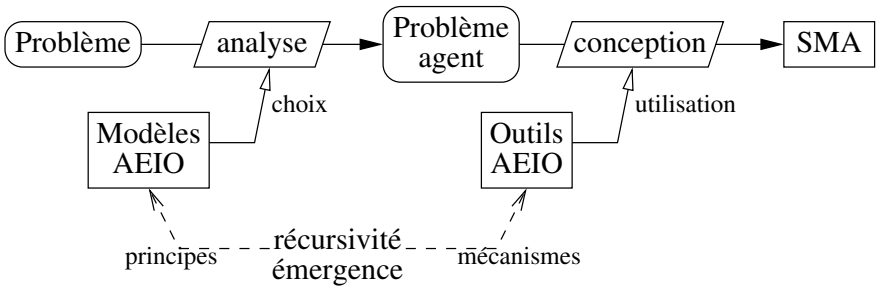


Fig. 14 – Le processus d'analyse–conception façon *Voyelles*

La conception se déroule suivant quatre phases :

1. la première phase consiste à dégager des informations sur le comporte-

ment global du système et à identifier les agents, suivant les modèles AEIO choisis pendant l'analyse ;

2. la deuxième phase s'intéresse aux Agents, de manière individuelle, et à l'Environnement :

– ordonnancement des tâches des agents : création des plans des actions à accomplir pour réaliser les buts,

– représentation de l'environnement et des agents : capacités perceptives et capacités d'action, compétences des agents, représentation des caractéristiques des autres agents ;

3. la société est le *motto* de la troisième phase : expression des Interactions par un ensemble de protocoles, et de l'Organisation par l'ensemble des relations entre les agents ;

4. la dernière phase concerne l'intégration des influences, c'est-à-dire l'ensemble des interactions entre les agents, et avec l'environnement (interactions non prises en compte dans la deuxième phase, ce qui implique une rétroaction sur le travail effectué lors des phases précédentes).

La phase de développement est ensuite facilitée par l'existence de la plateforme mask (cf. *supra*).

L'émergence, est le maître mot de la méthode de conception proposée par Jean-Pierre Müller pour concevoir des systèmes multi-agents de résolution de problèmes, [Müller 98].

Les systèmes visés cherchent à résoudre des problèmes, ou à faire de l'optimisation, en utilisant la dynamique des agents qui structurera l'espace de recherche. La solution à un instant donné sera la structure, l'organisation, prise par les agents à ce moment ; c'est-à-dire que cette solution peut changer dynamiquement, peut s'adapter, si les données du problème changent.

La méthode elle-même propose six étapes :

1. spécifier l'espace de recherche, décomposer sa structure en composantes, déterminer les états de l'espace de recherche et les transitions entre ces états ;

2. déterminer les entités qui produisent ces états, ce seront les agents ;

3. déterminer les buts et la dynamique des agents qui leur permettent de parcourir (potentiellement) l'ensemble des états ;

4. déterminer les contraintes environnementales qui guideront les agents ;

5. déterminer les processus internes de l'environnement : propagation des réactions, évolution propre, etc. ;

6. valider la conception : expérimentalement ou théoriquement suivant la complexité.

Les cinq premières étapes peuvent également se formuler sous forme de questions :

1. Qu'est-ce qui compose la solution recherchée ?

2. Qui crée ces composantes ? ou, plus simplement, Qui agit ?

3. Que veulent les agents ? et comment y parviennent-ils ? Quelles sont les interactions entre les agents qui créent les composantes de la solution ?

4. Quels sont les contraintes environnementales ?

5. Quelles sont les conséquences des actions des agents ?

Ceci permet de noter que le principe de cette méthode est de concevoir

les agents par leurs interactions en tant que créatrices des composantes du phénomène global représentant la solution recherchée.

On peut illustrer cette méthode d'un exemple. Prenons une version simplifiée du problème du voyageur de commerce : trouver un chemin entre des points précis dans un espace euclidien de dimension deux. L'on peut se référer aux différentes expérimentations réalisées concernant la résolution du problème du voyageur de commerce par des agents fourmis, notamment [Bonabeau & Théraulaz 94, p. 216] : bien que les auteurs n'aient pas suivi cette méthode, les résultats obtenus sont proches.

1. Le phénomène global, une solution, est un chemin entre différents points de l'espace, les *villes* ; les composantes de cette solution seront des délimitations locales de ce chemin (des *marques* de la présence du chemin) ;

2. les agents seront des entités qui déposeront ces marques en allant de *ville* en *ville* ;

3. une *ville* sera un pôle attracteur pour l'agent qui n'aura de cesse de les visiter ; les agents auront tendance à suivre les routes déjà tracées et donc à les renforcer ;

4. les marques représentant les portions de route ont une vie limitée ; on pourra interdire (ou rendre obligatoire) un passage en y plaçant des obstacles (ou des marques indélébiles) ;

5. les marques disparaissent avec le temps.

Cette méthode, bien qu'aucun outil de vérification ne soit réellement apporté (comme c'est le cas pour la méthode conçue par Ouiddad Labbani dans sa thèse, [Labbani-Igbida 98], et qui a servi de terreau à celle-ci), capture les principes fondamentaux de la conception de SMA émergentistes :

– le phénomène global émergent est produit par des interactions ;

– les agents sont les entités qui interagissent ;

– ces agents peuvent ne pas être conscients du phénomène global ou en tout cas de leur participation à sa dynamique.

Beaucoup des SMA que cette méthode vise utilisent le motif *Marques*[25] comme principe et processus fondamental.

2.5. Conclusions sur les méthodes

Le but du génie logiciel est de développer des méthodes et de définir des concepts partagés par une communauté d'utilisateurs — souvent industriels, pour encadrer des pratiques de développement. Or, dans la communauté SMA, il semble que chaque équipe crée sa propre méthode — voire plusieurs. La communauté ne semble donc pas très partageuse...

Ceci peut s'expliquer simplement si l'on fait une distinction téléologique :

– le besoin d'un génie logiciel de la part des industriels, leur but étant de répondre à leurs besoins réels en applications ;

– les logiciels d'étude construits par le monde académique — et donc une grande partie de la communauté SMA —, et qui servent à développer un ser-

25. Celui-ci fait partie des résultats de ce travail de thèse, nous n'allons pas le décrire ici. Nous prions le lecteur de se reporter à cette partie du mémoire pour mieux cerner le vocabulaire que nous utilisons ici.

vice pour des besoins non spécifiés, dans le but de faire émerger des propriétés observables.

Ainsi a-t-on d'une part des propositions d'outils généralistes qui ne le sont pas réellement et dont l'application n'est pas toujours aisée et d'autre part des outils spécialisés et contraints mais bien décrits.

De plus, il y a souvent confusion entre modèle et processus. UML, comme nous l'avons rappelé en introduction de cette section, est à la fois une notation et un modèle — objet en l'occurrence. Une méthode, quel que soit le modèle utilisé, est un processus qui mène d'un problème à un système (multi-agent) en différentes étapes, successives ou non, itérées ou non. Dans certains cas, le modèle n'est pas spécifié et les concepts sont donc décrits en même temps qu'ils sont utilisés dans la méthode. Dans d'autres, c'est le modèle qui est décrit et la méthode de conception revient à *identifier* : qui des *rôles,* qui des *tâches,* qui des *protocoles* ; — cela rappelle le début de la conception objet où les méthodes se réduisaient à « trouver les objets », ceux-ci étant évidents puisque « tout est objet ».

3. Conclusions

Étant donné l'âge du paradigme agent, les méthodologies de conception agent sont encore très récentes. Cela ne fait pas longtemps que l'on peut oser une définition des concepts agent — ne serait-ce que celui d'agent lui-même — sans lever une certaine polémique.

Les connaissances sur les systèmes multi-agents sont encore essentiellement empiriques. Ce sont ces connaissances, cette expérience, que les différents outils et plates-formes tentent de capitaliser, avec, comme l'on vient de le voir, plus ou moins de succès.

En ce qui concerne les méthodes, il n'y a pas encore de recul suffisant pour les évaluer, elles sont encore très mobiles (cf. Gaia et soda qui fusionnent à peine testées, ou les méthodes Cassiopée, Andromeda, etc. qui ne sont utilisées que sur une ou deux applications avant d'être au pire abandonnées, au mieux fortement refondues).

Après ces deux conclusions (l'aspect empirique de la connaissance du paradigme SMA et la récence du consensus sur les définitions de ses concepts), nous nous sommes intéressé aux techniques existantes qui permettent :

- de collecter et de formaliser l'expérience acquise ;
- de rendre compte des concepts agent ;
- de les formaliser ;
- d'aider à la conception de SMA.

Comme nous l'avons vu au chapitre précédent, les techniques de réutilisation permettent de collecter les concepts, les processus et les structures. Il nous faut une technique de réutilisation qui conserve :

- les concepts : ceux qui sont propres au paradigme agent ;
- les processus :
  - les méthodes de résolution de problèmes qui peuvent se poser lors de l'analyse (définition des concepts, application de grilles d'analyses), de la conception (relier les concepts, les informatiser) ou de la mise en œuvre (codage) ;

– les comportements, les fonctions des agents ou du système ;

– les structures : l'architecture du système, l'architecture d'un agent, des parties de code.

Les outils et les méthodes que nous venons de voir permettent, chacun dans une certaine mesure, ces réutilisations. Par contre, ils ne sont pas aptes à exposer et à formaliser l'expérience nécessaire à l'utilisation de ces éléments. En effet, manipuler des concepts avec efficience demande une certaine familiarité avec ces derniers. De même, l'abstraction d'une méthode de développement ne facilite pas son application à des problèmes particuliers. De plus, la justification de certains résultats (pourquoi telle architecture ? pourquoi tel codage ?) n'est pas toujours évidente, surtout si l'on a pas l'expérience — au sens de participation — du cheminement qui a mené à ce résultat.

À nos yeux, une seule technique peut permettre cette forme de réutilisation particulière qu'est celle de l'expérience. C'est la technique des motifs de conception, laquelle est le sujet principal de notre travail et de la seconde partie de ce mémoire. Mais, avant d'exposer notre travail, revenons sur les avantages apportés par les motifs.

Chapitre III

Avantages des motifs

If it happens once, it's a bug.
If it happens twice, it's a feature.
If it happens more than twice, it's a design philosophy.

— Anonyme

Avant d'appliquer les motifs au paradigme agent, nous allons développer, étudier plus en détail, quels sont les réels avantages des motifs, comment ils peuvent être utiles aux concepteurs dans le développement de logiciels.

Les motifs ont plusieurs avantages que nous avons déjà esquissés : ils forment un vocabulaire, ils aident à la documentation, à la compréhension et à l'apprentissage, ils peuvent s'appliquer en parallèle à d'autres techniques, et, bien sûr, ils sont d'une aide précieuse en génie logiciel en permettant une meilleure réutilisabilité, en aidant le concepteur tout au long du développement.

1. Vocabulaire de conception commun

1.1. Catalogue — thésaurus

Comme nous avions commencé à le dire lors de la présentation de la technique des motifs au chapitre précédent, les motifs n'ont de sens que lorsqu'ils se trouvent groupés, en un catalogue, voire même ce que nous osons appeler un thésaurus.

Dans un thésaurus de motifs, ces derniers sont souvent classés en différents types. Cette catégorisation dépend des interactions qu'entretiennent les motifs entre eux. Elle dépend aussi des auteurs et de la façon dont ont été décrits et découverts les motifs et dont les auteurs ont envisagé leurs utilisations.

Le Groupe des Quatre (*i. e.*, Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides, auteurs de l'ouvrage de référence, [Gamma *et al.* 94]), tout comme Buschmann *et alii*, [Buschmann *et al.* 95], les classe suivant leur niveau d'abstraction : analyse, conception, mise en œuvre (idiomatismes), architecture, comportement, etc.

Mark Grand, quant à lui, [Grand 98, Grand 99], les classe suivant leur finalité, leur domaine d'application dans le code : interface graphique utilisateur, gestion des responsabilités, organisation du code, optimisation, robustesse, tests.

On peut aussi rapprocher cette différence de catégorisation de celle qui peut exister entre un dictionnaire grammatical et un dictionnaire lexical : les mots peuvent être classés suivant leur catégorie (verbe, nom, adjectif) ou suivant leur parenté (radical, dérivés) ou bien encore suivant leur domaine d'utilisation (comme dans un dictionnaire analogique).

1.2. Vocabulaire commun

Le thésaurus de motifs décrit un véritable langage abstrait qui permet de facilement communiquer autour des abstractions, des concepts et des techniques que décrivent, que portent et que sont les motifs.

Ainsi, ce vocabulaire permet-il aux concepteurs de penser et de communiquer à un niveau d'abstraction plus élevé : chaque motif englobe en un seul terme (son nom) l'ensemble des concepts et des techniques qu'il décrit et qu'il fait interagir. Il englobe donc non seulement plusieurs faits (concepts, structures) mais aussi leurs interactions.

Ce vocabulaire permet aux concepteurs, aux modélisateurs et aux développeurs de communiquer, d'abord entre eux, à leur niveau d'abstraction respectif, mais aussi entre les différents niveaux : un motif d'analyse peut, par exemple, correspondre, interagir, avec un motif de conception ; il en est de même entre les différentes phases de développement.

Grâce aux motifs, la manipulation, le passage et la communication des spécifications, des modèles et des concepts sont grandement facilités.

2. Compréhension des systèmes

Les motifs permettent aussi une meilleure compréhension des systèmes existant. Du fait qu'ils sont construits à partir d'une étude des logiciels existant, les motifs proposent une expertise de ces systèmes. Ils proposent donc aussi une pré-expertise des autres systèmes, comme guides, points de repère pour leur étude.

Un système utilisant les motifs de façon visible, c'est-à-dire en explicitant son utilisation de motifs dans sa documentation, est aussi bien plus facile à appréhender. Les arguments que les motifs utilisés ou écartés apportent aux décisions prises lors du développement du système sont très convaincants et évitent de longues explications pour le concepteur du logiciel (car elles se trouvent déjà dans la description des motifs) et ce sont des arguments extrêmement pratiques pour l'observateur extérieur, cela lui évite d'avoir à entrer dans les détails souvent complexes d'une conception aboutie pour comprendre la structure et les raisons (les motifs !) de la construction de cette structure.

D'un autre côté, les motifs permettent de diffuser les concepts et les techniques de programmation. Ils permettent ainsi un apprentissage du développe-

ment de logiciels au travers de la capitalisation de l'expertise acquise par les autres développeurs ; capitalisation qu'ils représentent par leur description de ces concepts et techniques, par l'appui qu'ils prennent sur des exemples concrets et par les guides et les lignes directrices qu'ils apportent, tant dans la compréhension que dans l'application de ces concepts et techniques. Ceci est d'autant plus vrai que l'on possède déjà une expérience importante dans l'application de ce paradigme mais que le partage des concepts et des techniques de ce paradigme n'est pas facile (il semble d'ailleurs d'autant plus difficile de partager des concepts qu'ils ont un niveau d'abstraction élevé).

3. Outils complémentaires

Un troisième intérêt des motifs est qu'ils peuvent s'ajouter aux autres méthodes de conception. Ils n'empêchent en aucune façon l'utilisation d'autres moyens de conception. Ils sont utilisables parallèlement à ces moyens, que ce soient des méthodes d'analyse, de conception ou des outils de développement. Plus même, ils les complètent.

Les méthodes de conception n'ont pas pu capturer l'expérience des experts. Si l'on prend l'exemple de la conception objet, l'abord en a toujours été difficile et l'entière compréhension des concepts objet est toujours assez longue et ne peut se faire que grâce à l'expérience personnelle. Les motifs comblent cette déficience en montrant l'utilisation des techniques primitives du paradigme utilisé. Par exemple, le motif *Composite*, que nous avons cité précédemment, décrit le concept de composition mais aussi l'utilité de l'héritage à partir d'une classe commune, abstraite de surcroît. C'est un exemple de bonne conception objet et de la bonne utilisation du paradigme objet.

Les motifs sont un moyen de décrire le *Pourquoi* d'une modélisation ou d'une conception (ils sont les motifs de cette modélisation !). Ils sont aussi le compte-rendu des décisions prises, décisions qu'ils ont contribué à prendre en guidant les choix du développeur en regard des contraintes posées.

Les motifs facilitent en outre le passage entre les différentes phases de la conception (cf. l'introduction de ce mémoire). La transition entre deux phases nécessite la transformation d'une description du système en une autre : cela peut consister à passer d'un ensemble de besoins à une modélisation ou, plus généralement, d'un ensemble de contraintes à un modèle, une construction, les intégrant. De la même façon, des constructions décrites durant certaines phases peuvent disparaître dans les phases suivantes, tout comme des constructions apparaissent à mesure que l'on se rapproche des contraintes de mise en œuvre. Les motifs explicitent ces modifications en décrivant le passage des contraintes aux modèles (du champ *Forces* au champ *Solution*) et le passage des modèles aux constructions (par le champ *Mise en œuvre*).

4. Aide à la réutilisation

Déjà comme capitalisation de l'expertise, les motifs représentent une aide à la réutilisation (de cette même expertise). Mais ils sont aussi des outils de réutilisation dans le sens où ils mettent en avant des points du développement, des choix, qui sont primordiaux pour la réutilisabilité. Les motifs permettent de savoir où sont les points variables de la conception, les choix qui risquent

d'être la source de la reconception du logiciel ou, dans un meilleur cas, ceux qui permettent au logiciel d'évoluer sans heurt lors de la modification de contraintes liés aux besoins, au cycle de vie du logiciel.

Le cycle de vie d'un logiciel comporte deux grandes parties : développement et utilisation. Lors du développement, les besoins sont examinés les uns après les autres et la conception utilise les motifs pour répondre aux contraintes qu'ils posent. Lors de l'utilisation (phases de test comprises), de nouveaux besoins peuvent être ajoutés ou pris en compte, de nouvelles contraintes apparaissent. De même, après la mise en production du logiciel, le logiciel est exploité d'une façon plus profonde, d'autres utilisations peuvent apparaître et donc d'autres contraintes avec elles.

C'est un signe de bonne conception lorsque le logiciel est robuste à ces modifications, à l'expansion des contraintes ou à leur modification. Dans cette optique, les motifs permettent aussi de savoir comment cette reconception peut être dirigée, et, dans ce rôle, les antimotifs ont un rôle privilégié.

5. Conclusions

Les motifs permettent de décrire l'existant, de formaliser l'expertise, de la communiquer et de l'appliquer, et ce à tous les niveaux, pendant toutes les phases de la conception de logiciel. Les motifs sont donc une technique de génie logiciel d'une grande valeur.

De plus, ils permettent l'utilisation d'autres techniques de développement en parallèle de leur application. Les méthodes de conception, les plates-formes et autres outils continuent d'être utilisables et ceci est une bonne chose car ils sont très utiles.

Ces raisons nous ont conforté dans l'utilisation des motifs dans le domaine des systèmes multi-agents.

Dans le chapitre suivant, après un état de l'art de l'utilisation de motifs agent, nous proposerons nos propres motifs agents.

Seconde partie

Des motifs SMA

Chapitre IV

Des motifs SMA

No, I will be the pattern of all patience ; I will say nothing.
— William SHAKESPEARE, King Lear (III–2)

If it happens once, it's a bug.
If it happens twice, it's a feature.
If it happens more than twice, it's a design philosophy.
— Anonyme

Le chapitre II nous a montré que la conception orientée agent n'en est qu'à ces débuts. Toutefois, les systèmes multi-agents sont de plus en plus enseignés et utilisés dans le monde académique et, bien que certaines de ses branches soient plus sceptiques, l'industrie aussi vient à l'adopter, notamment dans le domaine des communications.

1. Motivations

Le développement du paradigme SMA est freiné par l'écart entre l'apparente facilité d'appréhender les concepts de base (agent, rôle, organisation, interaction, etc.) et la difficulté de passer d'un problème du monde réel à un SMA qui résout ou aide à résoudre ce problème. En clair, il s'agit de la difficulté de l'analyse et de la conception d'un SMA.

Les recherches actuelles s'orientent donc vers la définition de méthodes de conception. Certains travaux, tendent plutôt vers une méthode formelle, spécifique et d'accès difficile et contraignant (surtout pour ceux qui sont juste intéressés par l'utilisation pragmatique du paradigme agent). D'autres chercheurs ont tendance à vouloir trouver *la* méthode, celle qui s'appliquerait à tous les problèmes. Toutes ces méthodes se ressemblent un peu : l'analyse est fondée sur

les concepts de rôle et d'interaction, soit avec les rôles comme squelette, soit avec les interactions — dans ce cas, il est souvent fait usage des *use-cases* d'UML. Il semblerait aussi que l'on tende vers un UML-agent, qui serait le pendant agent de ce qu'UML est maintenant pour la POO : un modèle conceptuel sous-tendu par un langage et des diagrammes précis — AUML (dont nous avons parlé au chapitre II, section 2.1 page 41) semble en être une bonne ébauche.

Pour aider le concepteur de SMA, l'approche empirique, telle que [Simon 95] la prône, nous semble la plus appropriée dans l'état actuel. Cette idée est d'ailleurs confirmée par le foisonnement d'outils et de plates-formes d'implantation. Mais ces outils ne sont pas réellement pratiques ; leur prise en main est difficile (souvent rendue encore plus difficile par un cruel manque de documentations). De plus, comme ils interviennent à un niveau implantatoire, ils ne permettent pas de diffuser les concepts (la compréhension des petits exemples fournis ne permet pas de pouvoir réutiliser les concepts si facilement).

Or, comme nous l'avons dit dans le chapitre I, à la section III, les motifs sont un moyen d'exprimer à la fois les concepts et l'expérience, ils sont un moyen d'écrire, de décrire, de formaliser et donc de communiquer l'expérience acquise. Nous pensons donc que les motifs sont le médium idéal pour diffuser les concepts et les différents modèles utilisés dans la conception et la réalisation des SMA.

Nous nous sommes donc attaché à l'étude de l'application de cette technique des motifs au domaine des SMA. Après un bref état de l'art de l'utilisation de motifs agent, nous exposerons nos travaux dans le domaine, notamment grâce à la description de onze motifs.

2. Autres travaux sur les motifs agents

Bien que nous soyons parvenu à l'idée d'utiliser les motifs en suivant notre propre chemin, d'autres auteurs ont fait des trajets parallèles qui les ont menés aux mêmes conclusions. Nous allons donc décrire succinctement leurs travaux et analyser leur approche et leurs résultats.

Il faut tout d'abord noter qu'il existe un concept d'« agent mobile » dont la principale caractéristique est la mobilité, et non plus l'autonomie ou la finalité (cf. l'introduction de ce mémoire). Le développement de logiciels à base d'*agents mobiles* se rapproche bien plus de la programmation objet que l'on ne peut le faire avec les systèmes multi-agents en général. Les aspects système et interactionnel des SMA sont souvent négligés, ces logiciels ne comportant la plupart du temps qu'un nombre limité d'agents.

De fait, bon nombre d'utilisations du terme d'*agent* dans les applications sur l'Internet — que l'on peut aussi concevoir comme des applications développées par des spécialistes de la programmation objet utilisant une approche industrielle — devraient en fait parler d'*agent mobile.*

De ce fait, nous classons les différents travaux sur les motifs agents dans l'ordre de leur intérêt croissant vis à vis des SMA ; donc de leur attachement décroissant vis à vis de la seule mobilité des agents.

2.1. Tolksdorf

Robert Tolksdorf, [Tolksdorf 98], décrit quelques motifs qu'il qualifie de motifs agent bien qu'ils aient déjà été décrits en conception orientée objet.

Les auteurs ne semblent pas avoir appréhendé la notion d'agent telle que nous la voyons : dès qu'un programme est utilisé sur l'Internet, il deviendrait un agent !

2.2. Hung & pasquale

Dans [Hung & Pasquale 99], Eugene Hung et Joseph Pasquale décrivent un certain nombre de « motifs agent ».

Nous mettons des guillemets autour de cette expression pour deux raisons : d'une part, les motifs proposés sont très courts, et, d'autre part, le terme d'agent est pris avec la signification particulière d'assistant (*i. e.*, un agent mobile qui représente sur le réseau l'utilisateur du programme auprès d'autres programmes).

Ainsi, les motifs proposés comportent-ils seulement quatre champs : *Description, Key Application, Structure* et *Analysis*, aucun ne dépassant les quelques lignes (ou une simple figure dans le cas du champ *Structure*). L'on notera donc qu'il manque, notamment, le champ le plus important : les exemples au travers desquels le motif apparaît.

L'autre raison du guillemettage de l'expression « motifs agent » vient du fait que les auteurs décrivent les agents comme n'étant que des programmes capables de se déplacer dans un réseau hétérogène, de s'exécuter et de renvoyer des résultats. Cette définition un peu limitée des agents — *quid* du multi-agent ? — limite aussi la vision que l'on peut avoir de leurs applications mais surtout cela réduit fortement les avantages du paradigme agent. Les programmes d'assistanat distribués sont des applications d'un domaine particulier, et, comme tels, ils sont susceptibles d'utiliser des motifs particuliers à ce domaine. Mais cela ne fait pas de ceux-ci des motifs d'un type particulier comme peuvent l'être des motifs orientés SMA. Nous reviendrons sur ce point dans le chapitre V.

Finalement, comme motifs architecturaux, les motifs décrits pourraient bien se révéler former de véritables motifs, à condition d'être retravaillés, approfondis et surtout dotés d'exemples précis. Ils pourraient ainsi former un groupe de motifs portant sur les différentes utilisations des assistants.

2.3. Silva et delgado

Alberto Silva & José Delgado, *in* [Silva & Delgado 98], proposent de voir l'ensemble du paradigme agent comme un motif de conception unique. L'agent y est ainsi un objet actif, autonome, social, réactif et mobile.

Bien qu'il s'appuie sur l'étude de trois *frameworks* agents (Aglets, AgentSpace et Telescript), ce travail nous semble ne reconnaître que superficiellement les applications du paradigme agent puisqu'il ne s'intéresse qu'aux agents mobiles.

2.4. Meira, condeesilva & silva

Dans [Meira *et al.* 00], Nuno Meira, Ivo Conde e Silva & Alberto Silva proposent différents motifs agents. Une partie de ces motifs est la réécriture, de manière plus expressive *(sic),* de motifs déjà décrits par d'autres auteurs (que nous citons ici).

– *Receptionist* : un agent sert de pages jaunes pour un site précis (reprend *Facilitator* d'Aridor & Lange).

– *Secretary* : un agent « agent », envoyé à la place de l'agent, un représentant (reprend *Proxy*).

– *Session* : conservation d'une mémoire conversationnelle (reprend *Communication Session*).

– *Mobile Session* : ici les agents peuvent continuer à converser tout en se déplaçant.

– *Antenna* : un motif couplé avec le précédent, l'« antenne » permet de conserver le lien de communication entre les agents.

– *Private Session* : rendre la conversation sûre en la rendant privée (sans trop de détails d'ailleurs).

– *Meeting with Moderator* : communication à plusieurs sur le même sujet en utilisant un intermédiaire commun (inspiré de *Meeting*).

Les motifs présentés sont assez courts, sans trop de détails (pas de partie sur la mise en œuvre) : la solution tient en un paragraphe, ce qui permet d'intuiter l'idée du motif mais peut laisser dubitatif quant à sa mise en œuvre, voire son applicabilité.

2.5. Aridor et Lange

Yariv Aridor & Danny Lange, *in* [Aridor & Lange 98], cataloguent une liste de motifs de conception orientés agents mobiles.

Ils classent leurs motifs en trois types : mobilité *(traveling)*, tâche *(task)* et interaction *(interaction)*.

Les motifs de mobilité concernent l'encapsulation des concepts nécessaires à la gestion de la mobilité :

– *Itinerary* : réification des itinéraires et de leur routage.

– *Forwarding* : déplacement automatique sur un nouveau site d'un agent nouvellement arrivé.

– *Ticket* : réification des adresses de destination et des services nécessaires au déploiement d'un agent (qualité, quantité, délais, etc.)

Les motifs de tâches concernent la division et l'ordonnancement des tâches :

– *Master/Slave* : il s'agit de la délégation des tâches, ce motif est d'ailleurs déjà bien connu en POO (cf. par exemple [Fischmeister & Lugmayr 99, Massingill *et al.* 99]).

– *Plan* : le nom est un peu générique mais il s'agit en fait de coordonner plusieurs tâches exécutés sur différents sites.

Les motifs d'interaction sont sans doute l'apport le plus important du paradigme agent, c'est la catégorie la plus chargée :

– *Meeting* : la gestion de la synchronisation spatiale et temporelle nécessaire à une interaction locale.

– *Locker* : le « placard » permet à un agent de ranger ses données dans un endroit sûr, cela peut lui permettre d'abandonner un peu de poids lors de ses déplacements.

– *Messenger* : les messages entre les agents sont eux-mêmes des agents.

– *Facilitator* : la gestion d'un annuaire de noms symboliques pour faciliter les communications entre les agents, quelle que soit leur position.

– *Organized Group* : les agents se groupent et se déplacent ensemble.

Les schémas conceptuels sont décrits suivant le modèle objet. Mais les auteurs insistent (et nous les suivons sur ce point) sur le fait qu'il n'est absolument pas nécessaire de s'en servir dans une programmation objet.

Les motifs présentés sont purement des motifs de conception. Dans leur approche des motifs, les auteurs restent proches de la conception orientée objet et ne font pas ressortir toutes les caractéristiques du paradigme agent. Le paradigme agent n'est ici utilisé que pour apporter un niveau d'abstraction plus élevé pour envisager les modules et leurs interactions. Les modules sont manipulés grâce au concept d'agent mobile. Les autres notions portées par les SMA — comme l'autonomie ou la finalité — ne sont pas exploitées.

2.6. Occello & koning

Michel Occello et Jean-Luc Koning proposent d'utiliser des motifs au sein de la plate-forme qu'ils proposent, [Occello & Koning 00].

Chaque architecture d'agent, chaque architecture d'environnement, serait un motif de conception qui permettrait au concepteur utilisant la plate-forme mask de choisir et d'intégrer telle ou telle architecture dans le SMA qu'il construit.

Cela semble une bonne idée de proposer plusieurs architectures différentes dans une plate-forme, ainsi que de donner les moyens de l'utiliser. Le problème ici est double. D'une part, aucun motif n'est réellement proposé. D'autre part, bien que l'on puisse envisager un motif pour décrire les solutions proposées et les problèmes résolus et posés par une architecture particulière — c'est d'ailleurs ce que nous proposons nous-même dans quelques uns de nos motifs —, l'esprit avec lequel cela peut être fait doit être compatible avec la philosophie portée par la notion de motif, en tout cas si l'on veut appeler cela un motif. De ce fait, un tel motif doit rester générique : il doit pouvoir englober le minimum de trois exemples nécessaires et il doit pouvoir être adapté à différentes situations, et sa description doit être critique.

2.7. Kendall *et alii*

Elizabeth Kendall et son équipe sont les précurseurs dans l'utilisation de la technique des motifs.

Dans [Kendall *et al.* 96], ils proposent un « langage » de motifs — ou plutôt, un ensemble de motifs en collaboration —, le *Layered Agent Pattern Language*, qui comporte un motif global, le *Layered Agent*, définissant un motif de conception d'un agent suivant des couches, et cinq motifs, particuliers à chaque couche.

Ces motifs ont été repris et affiné, dans [Kendall *et al.* 97] puis dans [Kendall *et al.* 98], pour arriver à un ensemble de vingt-et-un motifs de conception (sans compter le *motif englobant*). Nous en reparlerons plus loin, dans la section 2.9 page suivante.

2.8. DEUGO *et alii*

Dwight DEUGO et son équipe, [DEUGO *et al.* 99B], relèvent bien les problèmes actuels du développement d'applications utilisant les SMA, ou, plus généralement, les techniques issues de l'IA :

1. le manque d'accord sur les définitions ;
2. les efforts dupliqués ;
3. l'incapacité à satisfaire les exigences de force de l'industrie ;
4. la difficulté d'identifier et de spécifier des abstractions communes au delà du niveau des seuls agents ;
5. le manque d'un vocabulaire commun ;
6. la complexité ;
7. le fait que seuls les buts et les solutions soient présentées (*i. e.*, l'absence des modèles et, surtout, des méthodes).

Ces remarques sont clairement justifiées, mais l'on peut y répondre.

En fait, les points un, deux, quatre et cinq se rejoignent, et, comme les auteurs le développent dans la suite de leur article — et comme nous le soutenons dans nos travaux —, l'utilisation de la technique des motifs et la construction d'un langage de motifs dans la communauté permettra d'y répondre.

Le point trois nécessite des formalismes et des méthodes de preuve et le dernier point tente de mettre en avant le manque d'une méthode de conception claire et définie. Or c'est une des grandes tâches actuelles de la communauté que de pallier ces manques (cf. le chapitre II page 33 de ce présent mémoire).

Quant au point six, concernant la complexité, — nous l'avons déjà plusieurs fois dit dans ce mémoire —, la manipulation de systèmes complexes demande d'aborder la complexité avec des méthodes appropriées.

Les motifs proposés dans la suite de l'article sont des motifs de conception qui tendent à supputer la technologie objet, et, comme nous l'avons signalé pour les travaux précédents, ils ont une propension à réduire le paradigme agent en utilisant principalement les concepts de la programmation orientée objet.

– *Direct Coupling* : en utilisant une technique dérivée de celle utilisée par *Observer*, [GAMMA *et al.* 94], les agents mobiles peuvent rester en contact même lors de leurs déplacements (par enregistrement/notification de la nouvelle adresse).

– *Proxy Agent* : utiliser un faux agent pour que la communication entre les agents mobiles puisse persister.

– *Communication Sessions* : gérer plusieurs communications grâce à la notion de session.

– *Badges* : gérer l'identification des rôles des agents par des marqueurs, cela évite de spécifier un agent réel lors d'une interaction.

– *Event Dispatcher* : la communication entre plusieurs agents mobiles est gérée par un objet intermédiaire, un diffuseur de messages.

2.9. DEUGO, KENDALL & WEISS

Dans [DEUGO *et al.* 99A], Dwight DEUGO, Elizabeth KENDALL et Michael WEISS regroupent les différents motifs décrits par les travaux précédents (notamment ceux cités dans les trois paragraphes précédents). Ces motifs sont classés en quatre types : architecture, communication, mobilité et coordination.

Des motifs architecturaux

– *Layered Agent* : décomposer l'architecture de l'agent en différents modules, agencés en couches.

– *Mobile Agent* : utilisation d'un *proxy,* d'un gestionnaire de sécurité et d'une classe mère encapsulant la mobilité.

– *Resource Manager* : encapsuler les ressources, ou une partie des ressources, dans un gestionnaire qui les contrôlera.

– *Jurisdiction* : création d'agents de haut niveau chargés de gérer les droits aux ressources.

Des motifs de communication

– *Meeting* : les agents se retrouvent dans un « lieu » spécial pour communiquer plus facilement.

– *Ambassador* : un agent s'informe sur une place distante en y envoyant un ambassadeur.

– *Proxy* : *idem* section 2.8, *supra.*

– *Finder* : un agent fixe sert de pages jaunes pour retrouver les agents mobiles.

– *Badges* : *idem* section 2.8, *supra.*

– *Whiteboard* : créer un « babillard » dans chaque place pour que les agents puissent s'y laisser des messages.

– *Translator* : l'agent traducteur permet à deux agents d'ontologies différentes de communiquer.

Des motifs de mobilité *(Travelling)*

– *Itinerary* : permet d'encapsuler l'itinéraire d'un agent mobile, en gérant le plan de déplacement de celui-ci (notamment les exceptions).

– *Ticket* : l'agent envoie un *ticket* avant de se déplacer réellement pour être sûr de la possibilité du déplacement.

– *Router* : attacher des propriétés à l'agent pour qu'un routeur puisse le diriger vers la destination la plus appropriée.

– *Relocator* : la gestion des liens existant entre un agent mobile et d'autres agents, ou avec ses ressources, pour qu'ils subsistent après un déplacement.

Des motifs de coordination

– *Master-Slave* : *idem* section 2.5 page 70.

– *Marketplace* : utilisation du protocole *contract-net.*

– *Specialist* : utilisation d'un tableau noir.

– *Negotiating Agents* : les agents négocient en cas de conflit d'action.

– *Subsumption* : les comportements de l'agent sont choisis suivant une priorité.

Avec l'expérience acquise par ce groupe d'auteurs, avec l'affinage et les perfectionnements successifs apportés à ces motifs, ces derniers gagnent en profondeur dans l'utilisation des spécificités du paradigme agent (cf. l'introduction de ce mémoire), même s'ils conservent encore un fort aspect programmation objet et si quelques uns sont un peu surprenant (notamment *Specialist*) ou déjà bien

connus (*Master-Slave* ou *Proxy*). L'on pourra arguer qu'il s'agit d'un mal nécessaire pour leur acceptation par l'industrie et les développeurs, habitués (non sans difficulté) à la programmation objet et grands apôtres du pragmatisme.

2.10. Aarsten, brugali & menga

Dès 1996, Amund Aarsten, Davide Brugali & Giuseppe Menga ont proposé des motifs orientés agent, [Aarsten *et al.* 96].

– *Compatible heterogeneous agents* : permettre l'utilisation d'agents hétérogènes grâce à la négociation d'interface (recherche des méthodes fournies par l'autre agent).

– *Visibility and Communication between agents* : chaque agent annonce les ressources qu'il fournit à son arrivée dans le système, ce motif propose aussi de définir la visibilité de ces ressources.

– *Leader/Collaborator/Collaboration* : comment gérer une collaboration concurrente entre plusieurs agents, sur le modèle Client–Serveur–Service.

Ces motifs assez complets proposent une mise en œuvre orientée objet (ils utilisent corba) mais restent d'assez haut niveau. On peut regretter que seuls trois motifs soient proposés, restreignant ainsi la vision du paradigme agent.

2.11. Conclusions

Nous avons déjà essaimé quelques critiques sur les directions prises par nos confrères dans l'utilisation des motifs pour les agents et pour les SMA. La principale de ces critiques concerne l'approche faite du paradigme agent et les réductions parfois réalisées, notamment la perte de l'aspect système et interactionnel, qui tend à ramener le paradigme agent à un simple domaine d'application de la programmation objet. Certes, considérer l'agent comme un objet actif, autonome, social, réactif et mobile est nécessaire, mais c'est aussi fortement réducteur.

Toutefois, comme nous l'avons exposé au chapitre précédent, les motifs sont utiles en génie logiciel et nous pensons qu'ils peuvent aussi l'être aux systèmes multi-agents. Les travaux cités ici indiquent un besoin de la communauté agent pour les outils de conception souples que sont les motifs.

D'ailleurs, malgré les remarques que nous avons osées, ces travaux permettent aussi de débroussailler le chemin de l'utilisation de motifs agent mais aussi de l'utilisation du paradigme agent.

—★—

Maintenant que nous savons ce qu'est un motif. Maintenant que nous avons défini nos objectifs et repéré quelques uns des écueils. Nous pouvons essayer d'appliquer notre solution pour l'aide au développement de systèmes multi-agents : les motifs.

3. Nos motifs

Nous allons proposer un ensemble de motifs qui pourront servir de fondement au développement de cette technique parmi les concepteurs de SMA. Nous

proposons différents types de motifs, d'une part pour montrer les divers usages possibles de cette technique et d'autre part pour couvrir une part suffisamment grande du domaine de l'analyse, de la conception et de la mise en œuvre de SMA.

3.1. Types de motifs

Comme nous le disions dans le chapitre III page 61, chaque auteur catégorise ses motifs suivant leurs interactions, la façon dont ils ont été décrits et découverts et l'utilisation envisagée. Nous ne faisons pas exception à la règle et, au cours de la découverte et de l'écriture des motifs que nous proposons, leur classification a évolué pour aboutir à celle présentée sur la figure 15.

Outre les motifs « classiques » (ici, principalement les motifs architecturaux), nous avons déterminé deux autres classes de motifs agent : les motifs métaphoriques et les métamotifs (cf. fig. 15). Les motifs métaphoriques sont des motifs qui tirent leur racines d'un domaine tout autre que celui des SMA ou même de l'informatique ; ils proviennent de l'utilisation systématique d'une métaphore décrivant une technique fortement présente dans un domaine particulier. Les métamotifs sont des motifs plus abstraits que les motifs habituels, ils définissent un ensemble de concepts qui seront utilisables dans toutes les phases de la conception d'un SMA : analyse, conception et mise en œuvre.

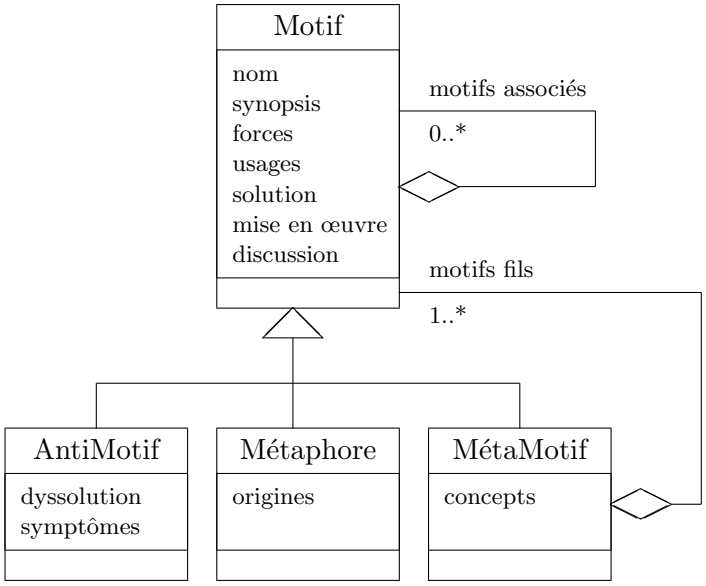


Fig. 15 – Motifs et motifs

Nous proposons aussi trois antimotifs qui, comme nous l'avons dit au premier chapitre, à la section 3.3.5 page 25, sont des motifs particuliers décrivant des techniques récurrentes de reconception, de ré-ingénierie.

3.2. Champs de nos motifs

De la même façon que pour leur classification, le formalisme utilisé pour la description de nos motifs a évolué. En partant des champs principaux que nous donnions dans la section 3.3.3 page 24, nous avons choisi les champs suivants comme étant les champs constitutifs d'un motif :

Synopsis : une explication rapide de ce que fait le motif, une phrase pour présenter les problèmes et leur solution.

Forces : un récapitulatif des contraintes contextuelles, des problèmes, que l'on cherche à régler avec ce motif. Il n'est pas nécessaire que le problème contienne toutes les *forces* décrites pour que le motif soit applicable. Néanmoins, plus il y en a, plus le motif à de chances d'être réellement applicable et plus son application à de chances d'être utile et avantageuse.

Usages : des exemples concrets d'utilisation et d'application de ce motif. C'est à partir de ces exemples qu'à été découvert le motif. Ils sont au moins au nombre de trois. Attention, il ne s'agit pas de donner *tous* les exemples qui utilisent le motif, seulement un échantillon.

Solution : ce champ explique quel est le principe permettant de résoudre les contraintes posées et comment utiliser le motif dans le développement d'une application à base de SMA.

Mise en œuvre : donne quelques directives aidant à la mise en œuvre de ce motif dans un SMA : comment appliquer la solution, quelles sont les détails à prendre en compte dans la réalisation de cette solution.

Discussion : cette partie explique les raisons d'utiliser ou non ce motif, ces avantages et ses inconvénients, les problèmes annexes et les questions qu'il peut poser.

Motifs associés : une liste de quelques motifs à consulter, motifs SMA ou motifs objets, qui aident à la mise en œuvre de celui-ci ou qui proposent une solution alternative intégrant d'autres *forces* ou en excluant certaines.

Pour les motifs métaphoriques, un champ supplémentaire nous a semblé nécessaire à leur description :

Origines : les sources d'inspirations de la métaphore, qui émergent au travers d'exemples de sa présence dans des domaines différents du monde des SMA.

De la même façon, les métamotifs nécessitent une modification de la formulation. Le champ *Concepts,* qui expliquera les différents concepts utilisés et définis dans ce motif, remplacera le champ *Forces* des motifs classiques. De plus, chacun des champs *Usages, Solution, Mise en œuvre* et *Discussion* des motifs classiques est décomposé en trois parties, traitant chacune d'une des trois phases classiques : analyse, conception et mise en œuvre.

Bien sûr, les antimotifs, bien qu'ils soient eux-mêmes des motifs, nécessitent, par leur définition, deux champs supplémentaires :

Symptômes : les effets visibles, les contraintes à résoudre, de l'application de cet antimotif (ou plutôt de la *dyssolution* de cet antimotif). Ce champ remplace le champ *Forces.*

DysSolution : la mauvaise solution ou les techniques conduisant à l'apparition de cet antimotif et de ses symptômes.

Le champ *Solution* des antimotifs consistera ainsi en la description d'une technique, d'un processus, de reconception du SMA qui permettra d'éliminer l'antimotif.

Métamotifs

Ces motifs sont plus que des motifs dans le sens où chacun décrit un principe fondamental du paradigme des systèmes multi-agents. De plus, ces *métamotifs* engendrent chacun une série de motifs *fils*, véritables instances de leurs géniteurs.

Nous avons décrit deux métamotifs :

Schémas d'organisation : utilisation des concepts de *rôle, organisation* et *schéma d'organisation.* Les motifs fils sont des schémas d'organisation.

Protocoles : utilisation du concept de protocole. Les motifs fils sont des motifs expliquant l'utilisation de tel ou tel protocole.

Nous les décrivons en remplissant les champs des motifs énoncés dans le chapitre précédent.

Motif

Schémas d'organisation

1. Synopsis

Les concepts organisationnels peuvent servir à analyser et concevoir un SMA, ils peuvent aussi être réifiés dans sa mise en œuvre.

2. Concepts

Ce motif définit différents concepts tels qu'ils sont utilisés par la communauté SMA. Ces concepts et leurs relations sont représentés sur la figure 16.

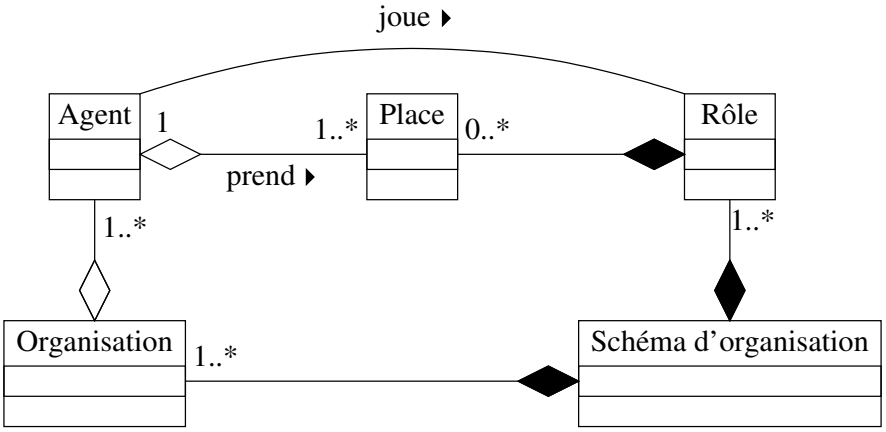


Fig. 16 – Concepts du métamotif *Schémas d'organisation*

– *Agent :* une entité autonome du système, c'est elle qui donne son nom au paradigme de programmation.

– *Rôle :* ce terme désigne une fonction qu'un agent peut prendre dans une organisation ; cette fonction peut être à courte durée (elle est prise en charge puis abandonnée par l'agent) ou pérenne, dans le sens où elle peut survivre à l'agent (au cours du temps, plusieurs agents peuvent se succéder dans la prise en charge du rôle). Chaque rôle définit les comportements de l'agent qui jouera ce rôle et il définit aussi les interactions et les possibilités d'interaction que l'agent aura avec d'autres agents, du fait de leur rôle.

– *Organisation :* c'est une structure qui regroupe plusieurs agents, chacun jouant un ou plusieurs rôles dans le schéma d'organisation dont elle fait partie.

– *Schéma d'organisation :* il s'agit d'une structure abstraite à partir de laquelle les organisations sont instanciées ; un schéma d'organisation est donc une classe d'organisations. En fait, de notre point de vue, le schéma d'organisation définit une classe d'ensembles de rôles en interaction.

Les concepts d'*agent* et de *rôle* ne posent pas de problème et sont communs à tous les auteurs. Par contre, les termes d'*organisation* et de *schéma d'organisation* ne sont pas (encore) standards — il est vrai que ces notions sont plus délicates. Le principal problème vient de la difficulté à appréhender la nature et la place de l'instanciation entre ces différents concepts — car il y a plusieurs sortes, plusieurs niveaux, d'instanciations entre ces concepts (cf. la section 4 du chapitre I, page 28).

a) D'une part, chaque schéma d'organisation peut se réaliser en plusieurs organisations. Une organisation est donc une instance d'un schéma d'organisation. De plus, plusieurs agents peuvent prendre en charge le même rôle, — soit dans des organisations différentes, soit dans la même organisation —, donc, lorsqu'un agent prend un rôle en charge, il réifie ce rôle en une *place de rôle* qui est particulière à l'agent (le rôle prévoit plusieurs places).

b) D'autre part, comme on peut le voir sur la figure 17 page suivante, tous ces concepts sont eux-mêmes instanciés (dans un sens plus programmation orientée objet cette fois : les modèles structurels sont utilisés pour créer des objets informatiques) pour créer des cas concrets de schémas d'organisation ; comme c'est le cas ici avec le schéma d'organisation *Contract-Net* qui définit un type d'organisation particulier mettant en relation deux rôles, celui d'*initiator* et celui de *participant,* dont le comportement sera dirigé par le protocole du *Contract-Net* tel qu'il est défini par la fipa[1].

De ce fait, du côté modèle de classes, nous avons les concepts de *schéma d'organisation* et de *rôle,* et du côté modèle d'objets, nous avons les *agents* qui prennent des *places de rôles* dans des *organisations.* L'instanciation a donc lieu d'une part entre les schémas et les organisations et entre les rôles et leurs instances, et d'autre part entre ces concepts et les objets informatiques et particuliers représentant des cas concrets.

Si l'on reprend notre exemple, l'on peut voir que le schéma d'organisation *Contract-Net* est instancié (au sens *a)*) en une organisation *Mon Contract-Net* qui est composée par trois agents dont un prend en charge le rôle d'*initiator* (en prenant la seule place prévue par ce rôle) et les deux autres celui de *participant* (c'est-à-dire que chacun prend une des places que ce rôle prévoit).

Pour prendre un exemple plus complexe (*i. e.*, avec plus de rôles) et bien

1. Cf. le motif *Protocoles,* page 87, et la section *Protocoles de communication* du chapitre I, page 26.

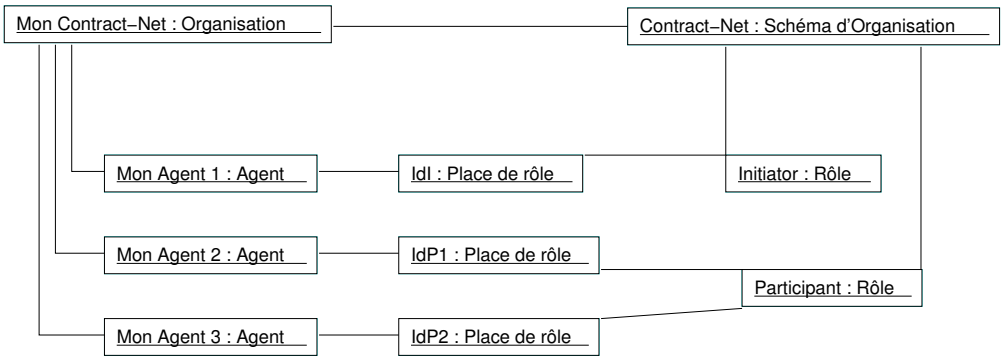


FIG. 17 – Exemple de schéma d'organisation *(Contract-Net)* et d'une organisation concrète attachée *(Mon Contract-Net).*

connu, examinons celui des *associations loi 1901*. La fameuse « loi du 1er juillet 1901 »[2] prévoit (en tant que schéma d'organisation) l'existence d'*associations* (d'organisations) comportant des *membres* et deux *responsables* (les rôles dans l'organisation). Les statuts particuliers de l'association peuvent affiner ces rôles (par exemple en appelant *président* le premier responsable et *trésorier* le second) ou définir d'autres rôles.

À partir de là (*i. e.*, des termes de la loi), une association peut être créée et se voir dotée de statuts et de règlements organiques, des agents peuvent prendre les places prévues par les différents rôles, suivant les statuts établis (*e. g. membre actif* s'il paie sa cotisation, *membre honoraire* s'il est coopté, le *président* doit être élu, etc.)

En particulier, les statuts types[3] prévoient un *président*, un *trésorier*, un *secrétaire*, un *vice-président* et des *membres*. Les rôles de *président* et de *trésorier* prévoient une seule place, obligatoire et les agents qui les prennent sont distincts (*i. e.*, l'association *doit* avoir un président et un trésorier uniques, mais deux agents différents prendront ces places). Les rôles de *vice-président* et de *secrétaire* prévoient une place au maximum mais peuvent ne pas être pris en charge ou bien cumulés avec d'autres charges (*e. g.* le trésorier pourra aussi être secrétaire et vice-président[4]). Quant au rôle de *membre*, tous les autres rôles en découlent : c'est le rôle le moins spécialisé ; prendre la place de président, de trésorier, de secrétaire ou de vice-président nécessite d'être membre de l'association (*i. e.*, de prendre une des places du rôle de *membre*).

Il est à noter que ni la loi dans cet exemple, ni le modèle organisationnel dans sa construction ne supposent qu'un rôle soit joué par un seul agent. En effet, le concept d'agent contenu dans le modèle peut très bien s'appliquer à un groupe d'agents, à une organisation d'agents de niveau plus faible. Pour revenir à l'exemple, rappelons qu'il est légal d'avoir, et même qu'il existe, des associations dont les *responsables* sont en fait des collectifs.[5]

Il existe d'autres concepts associés à ce métamotif :

– *Interaction :* lien de communication, de perturbation ou d'influence entre

2. La loi a été maintes fois modifiée et amendée, voir `http://www.legifrance.fr` pour une copie et un historique complets.
3. Stéréotype des statuts d'une association, pour aider à l'écriture des statuts réels.
4. Évidemment, le président ne peut être en même temps vice-président...
5. Les associations de troc et d'échange de type SEL en sont des exemples.

deux agents. Deux agents sont en interaction quand ils peuvent communiquer (par message ou par tout autre moyen) ou quand l'un d'entre eux peut perturber ou influencer l'autre (*i. e.*, ses actions, ou leurs résultats, peuvent être perçues comme des *stimuli*).

– *Accointance :* il s'agit, pour un agent, d'un autre agent avec lequel il est en interaction (ou a été, ou peut être, en interaction). Le terme d'*accointances* représente aussi, pour un agent, les agents qu'il connaît[6] et qui peuvent le connaître.

3. Usages

3.1. Analyse

Comme pour les sociétés humaines ou animales, les concepts organisationnels peuvent servir à analyser des sociétés existantes. Cette utilisation pour les SMA peut être d'en faire une simulation multi-agent.

La première possibilité d'usage des concepts de ce métamotif est la phase d'analyse du système multi-agent.

Comme nous l'avons vu au chapitre II page 33, et plus particulièrement dans la section 2.2 page 45, parmi les méthodes d'analyse agent, beaucoup utilisent les concepts organisationnels, à des niveaux divers.

Emmanuelle Le Strugeon, René Mandiau & Gaëtan Libert ont proposé, dès 1993, [Le Strugeon *et al.* 93], une méthode d'analyse organisationnelle d'un système : grâce à un catalogue des structures organisationnelles possibles (les schémas d'organisation), la structure adéquate vis à vis de la fonction du système est choisie et appliquée au SMA.

Le modèle organisationnel de B. Durand, [Durand 96], augmenté par [Sauvage 97], demande l'utilisation des concepts de schéma d'organisation, d'organisation et de rôle pour analyser le SMA car ce modèle définit et utilise ces concepts.

Le modèle aalaadin, [Gutknecht & Ferber 98], proposé aussi comme une méthode d'analyse–conception, reprend, en le simplifiant, le modèle de Benoît Durand. Dans sa partie *méthode,* il définit les concepts de *rôle*, de *groupe* (que nous appelons ici *organisation*) et de *structure de groupe* (ici *schéma d'organisation*). Il définit aussi d'autres concepts non présents dans notre motif : celui d'*organisation,* qui représente en fait l'ensemble des groupes existant dans le système, et celui de *structure organisationnelle,* qui représente le type d'organisation (*e. g.*, hiérarchique) qu'aura l'ensemble des groupes du système. Il est à noter que c'est le concept de *groupe* et non d'*organisation* qui est fondamental pour Olivier Gutknecht & Jacques Ferber ; les organisations se résument à des groupes d'agents qui peuvent communiquer entre eux.

La méthode soda, [Omicini 00], est principalement fondée sur le concept de tâche mais l'analyse suppose que chaque tâche est attribuée, non pas à un agent, mais à un rôle, ou à un groupe.

La méthode Gaia, [Wooldridge *et al.* 00], de même que les différentes méthodes dérivées de Gaia et soda, [Zambonelli *et al.* 00b, Zambonelli *et*

6. Dans le sens où il sait comment communiquer avec eux, comment les joindre, ce qu'il peut leur demander.

al. 00a], reposent sur le concept de rôle : un système multi-agent y est analysé comme une organisation de rôles.

La méthode Cassiopée, [Collinot & Drogoul 98a, Collinot & Drogoul 98b], définit elle aussi un système multi-agent comme un ensemble de rôles en interaction. Une typologie tripartite de rôles est établie : rôles liés aux tâches fonctionnelles de l'application, rôles interactionnels (communication) et rôles organisationnels ou coopératifs (gestion de la coopération).

Les *role-models*, [Kendall 98a], utilisés dans la plate-forme Zeus de *British Telecom*, se sert de prototypes de rôles déjà définis comme base d'analyse du système : c'est une sorte d'analyse–synthèse.

3.2. Conception

Après avoir été utilisé pour analyser le système, ce métamotif peut aussi être utilisé pour en concevoir une modélisation informatique, c'est-à-dire pour concevoir un SMA. On peut aussi se servir des concepts organisationnels pour concevoir des SMA opérants munis d'une structure organisationnelle.

Certaines méthodes d'analyse–conception, comme celles que nous venons de survoler plus haut et qui sont décrites au chapitre II page 33, utilisent un modèle conceptuel organisationnel : le système est modélisé sous forme de rôles et d'organisations.

MadKit, [Gutknecht & Ferber 98, Gutknecht & Ferber], met en œuvre le modèle aalaadin « agent–groupe–rôle » (cf. *supra*) et permet donc une conception organisationnelle parallèle à une conception objet ; cette dernière catégorise les agents suivant leurs capacités, leurs fonctions, etc.

L'architecture formelle de Vincent Hilaire, [Hilaire *et al.* 00], tout comme celle de Benoît Durand, [Durand 96], utilise un modèle organisationnel qui est directement une spécification du système. Les diagrammes conceptuels sont donc directement issus de l'analyse.

3.3. Mise en œuvre

Une dernière façon d'utiliser ce métamotif *Schémas d'organisation* est de réaliser, directement au niveau du programme, les différents concepts qu'il transporte.

Dans la plate-forme où B. Durand a appliqué son modèle organisationnel — et avec laquelle il a, par exemple, développé une simulation d'épizootie de fièvre aphteuse —, ainsi que dans la plate-forme que nous avions développée pour mettre en œuvre une variante de ce modèle, les rôles, ainsi que les organisations, sont réifiés. En fait, les organisations y sont même des agents : elles ont un rôle *organisation* dans l'organisation *système,* la tâche attachée à ce rôle étant de veiller à l'exécution, par tous les agents de cette organisation, de leur propre tâche pour un tour d'exécution : les fonctions associées aux rôles qu'ils ont dans cette organisation (appelées *comportements* dans le modèle). Cette réflexivité est un moyen simple, mais confusionnel, d'assurer une concurrence des comportements.

La réification des rôles et des organisations permet aux agents de prendre des rôles en charge ou de les abandonner.

Enfin, MadKit possède les classes *rôle* et *groupe*, et les agents peuvent les manipuler et même en créer dynamiquement. MadKit permet la création dynamique

de groupes car, dans sa mise en œuvre comme dans son modèle, aalaadin, un groupe n'est instance que d'un seul schéma d'organisation, celui de groupe (*i. e.*, un groupe n'est qu'un ensemble de rôles, il ne définit pas les interactions entre les rôles, seul le principe de séparation est défini : deux agents ne devraient pouvoir communiquer que lorsqu'ils ont un groupe en commun, c'est-à-dire lorsque chacun prend un rôle — pas forcément différent — dans un même groupe).

4. Solution

4.1. Analyse

La solution qu'apporte ce motif dans le domaine de l'analyse d'un système est de voir le système comme un ensemble d'organisations, elles-mêmes composées de rôles en interaction. Il faut ensuite reconnaître le type de chaque organisation, c.-à-d. le schéma d'organisation à partir duquel elle peut être instanciée.

Dans ce cadre, un catalogue des schémas d'organisation les plus courants, ainsi qu'un catalogue des rôles types de ces schémas, permettrait une analyse plus rapide. C'est ici que ce motif montre son aspect « méta » : il commande plusieurs autres motifs, un par schéma d'organisation type ; chacun de ces motifs définirait donc un ensemble de rôles en interaction et ses cas d'utilisation.

Le manuel *« The Role Modelling Guide »* de la plate-forme Zeus, [Collis & Ndumu 99], est une ébauche de ce genre de catalogue : les rôles les plus courants y sont répertoriés dans le but d'aider à leur identification, à leur conception et à leur mise en œuvre. La vision un peu trop orientée objet du *role-model* tel qu'il est appliqué dans Zeus ne prend pas en compte les organisations, et encore moins les schémas d'organisation, mais il peut être d'une aide précieuse.

4.2. Conception

Lors de la construction des schémas conceptuels — notamment du diagramme de classes si l'on prend UML comme notation —, les rôles, les organisations auxquelles ils appartiennent et les schémas d'organisation dont sont issues les organisations sont décrits.

Faire apparaître les rôles permet de virtualiser les services : les relations décrites entre les rôles ne dépendent que des rôles en interaction, pas des agents particuliers qui jouent les rôles.

Faire apparaître les organisations, c'est décrire les relations entre les rôles et montrer leur cohésion au sein d'une entité de granularité supérieure : l'organisation.

Faire apparaître les schémas d'organisation, c'est indiquer, pour chaque organisation, le schéma dont elle est issue.

Dans plusieurs des méthodes présentées au chapitre II, les rôles apparaissent dans les diagrammes ; les relations entre les différents rôles sont aussi décrites, l'on peut ainsi dire que les organisations sont décrites. Par contre, les schémas d'organisation apparaissent peu (voire pas du tout). Or, il nous semble que l'on peut tirer avantage de la présence des schémas dans les diagrammes conceptuels, ne serait-ce que pour la documentation ainsi apportée.

4.3. Mise en œuvre

La mise en œuvre de ce métamotif peut se faire sur différents niveaux. Le niveau le plus simple est celui où seules les places prévues par les rôles sont réifiées dans le système, les agents peuvent ainsi les prendre et les abandonner à loisir.

Le niveau suivant demande à ce que les organisations soient aussi des objets du système. Le modèle de B. Durand est un exemple opérationnel de ce niveau. MadKit est aussi à ce niveau, bien que les groupes d'aalaadin ne sont pas véritablement des organisations (cf. la présentation du modèle aalaadin, page 46, et [Nicolle 02]).

Le niveau final voit, quant à lui, les schémas d'organisation à leur tour instanciés ; les agents peuvent donc instancier des organisations sur des schémas, des modèles, qui n'existent pas au démarrage du système. La manipulation de schémas d'organisation est tout à fait comparable à la manipulation de classes (puisqu'un schéma d'organisation est une classe d'organisations), l'intégrer dans un modèle n'est pas chose aisée.

5. Discussion

5.1. Analyse & Conception

Cette section concerne à la fois l'analyse et la conception car la seule distinction entre les deux vient du système considéré : pour l'analyse, il s'agit du système « réel » (qui fait partie du monde), et pour la conception, il s'agit du système logiciel que l'on est en train de concevoir.

L'utilisation d'un modèle de conception organisationnel permet de représenter conceptuellement la connaissance du monde que l'on tire de l'analyse. Le modèle du monde ainsi construit intègre plus de connaissances du Monde et permet donc un niveau d'abstraction élevé.

L'utilisation de ce motif est extrêmement pratique lorsque le système possède plusieurs groupes organisés ; il permet de séparer les points de vue que l'on peut avoir sur le système et d'appliquer ainsi le vieil — mais néanmoins particulièrement utile et fréquemment appliqué — adage « diviser pour régner ».

Ce motif n'est pas exempt de problèmes. Son application a en effet tendance à figer l'organisation du SMA et donc à réduire l'adaptivité et la réactivité que pourrait avoir le système s'il pouvait se réorganiser. En fait, comme le souligne Anne Nicolle, [Nicolle 02], la réorganisation automatique et émergente ne peut effectivement avoir lieu que dans les groupes, les organisations doivent prévoir toute réorganisation.

Un autre problème, plus important, vient de la difficulté de séparer rôles et agents, certains rôles sont très dépendants de l'agent, d'autres sont très imbriqués et peu discernables.

5.2. Mise en œuvre

Après avoir utilisé ce métamotif pour l'analyse et pour la conception du SMA, l'utiliser pour sa mise en œuvre permet de manipuler les mêmes concepts de bout en bout de la création d'un SMA : la spécification du SMA nous en donne son implantation.

La réification des schémas d'organisation permettrait d'avoir plusieurs organisations construites suivant le même schéma d'organisation sans avoir, comme c'est le cas avec la mise en œuvre du modèle de B. DURAND, à les prévoir à l'avance (en fait, l'utilisation d'un langage interprété, comme Airelle pour [DURAND 96] ou clos, permet aussi cela mais de manière moins élégante et surtout moins lisible qu'une simple instanciation).

Le problème vient encore une fois du manque de souplesse induit par ce métamotif : les schémas sont prévus par le concepteur, les libertés de l'agent en sont d'autant réduites.

6. Motifs associés

Tous les motifs décrivant des schémas d'organisation et explicitant leur utilisation sont intimement liés à ce métamotif.

Parmi les motifs de ce présent thésaurus, le métamotif *Protocoles* (page 87), qui définit des phases d'interaction entre rôles, est particulièrement intéressant : il permet de découvrir, en fonction de leur interaction, les différents rôles présents dans le système.

De la même façon, les schémas d'organisation que l'on peut extraire des exemples donnés par le manuel *« The Role Modelling Guide »* de la plate-forme Zeus, [COLLIS & NDUMU 99], sont des motifs d'organisations agent, des motifs organisationnels. Les exemples de rôles sont des motifs comportementaux pour les agents.

Le motif *Architecture récursive* (page 131) peut aider à concevoir les sociétés d'agents, ou plus simplement les groupes d'agents, comme des agents, englobant les agents qui les composent, qui peuvent avoir des rôles dans la métasociété. L'agent-groupe aura un rôle différent et indépendant des rôles de ses agents-composants.

⋆⋆⋆

Motif

Protocoles

1. Synopsis

Utiliser les protocoles de communication (et les protocoles d'interaction) pour l'analyse, la conception et la mise en œuvre de SMA.

2. Concepts

Ce motif définit — ou plutôt redéfinit, de manière plus précise, plus claire — différents concepts déjà utilisés en informatique. Ce motif partage aussi certains concepts avec le motif *Schémas d'organisation* que nous venons de décrire.

– *Agent :* une entité autonome du système, c'est elle qui donne son nom au paradigme de programmation.

– *Message :* outre le sens simple d'information transmise par un agent à un autre, ce concept porte aussi la notion d'acte de langage (cf. les travaux d'Austin, Searle[1] & Vanderveken).

– *Interaction :* lien de communication, de perturbation ou d'influence entre deux agents ; deux agents sont en interaction quand ils peuvent communiquer (par message ou par tout autre moyen) ou quand l'un d'entre eux peut perturber ou influencer l'autre (*i. e.*, ses actions, ou leurs résultats, peuvent être perçues comme des *stimuli*). Dans son acception au sein d'un protocole de communication, il s'agit généralement d'une conversation.

– *Rôle :* ce terme désigne une fonction qu'un agent peut prendre dans une interaction régie par un protocole. Chaque rôle définit les messages que l'agent qui jouera ce rôle pourra envoyer.

– *Protocole :* un ensemble de règles qui permet à deux agents ou plus de se coordonner sans ambiguïté. Cette coordination peut avoir différents buts : échange d'information, demande de service, négociation de ressources, etc. Un protocole définit l'ordre et le type des messages et des actions que les agents doivent se transmettre et opérer.

1. J. L. Searle, *Speech Acts, Cambridge University Press,* 1969.

Il est à noter que, bien que le terme de protocole de communication soit souvent employé dans la littérature (et ici !), celui de protocole d'interaction (pour des tâches physiques) peut bien souvent avantageusement le remplacer.

3. Usages

Les interactions étant un des fondements du paradigme agent, il ne manque pas d'exemples de SMA et de techniques SMA qui utilisent les protocoles d'interaction, à niveau ou à un autre. En voici quelques unes, classées suivant la phase de développement où l'utilisation est faite.

3.1. Analyse

Nombre des méthodes d'analyse–conception agent, telles celles que nous avons parcourues dans le chapitre II, intègrent la notion de protocole de communication dans la phase d'analyse du système. Souvent, la communication entre les agents n'est en fait envisagée que sous le champ plus général des interactions. La communication entre les agents, les messages qu'ils s'échangent, ne sont donc abordées que d'une manière assez indirecte.

Par exemple, dans les règles de transformation de graphes, [Depke *et al.* 00], les protocoles utilisés par le système sont décrits lors de la phase d'analyse, ils y sont même primordiaux puisqu'ils sont les graphes de base du système (ensembles de bi-graphes représentant l'état du système avant et après l'application d'une transition).

Dans le même ordre d'idées, Gaia et soda, [Omicini 00, Wooldridge *et al.* 00], et toutes les méthodes conçues par les mêmes équipes (cf. le chapitre II pour plus de détails), construisent des diagrammes d'interaction lors de la phase d'analyse du système. Ces diagrammes représentent les protocoles utilisés dans le système modélisé.

3.2. Conception

La méthode MaSE, [Wood & DeLoach 00], intègre les protocoles de communication dans la phase de conception, au travers de la construction des « conversations ».

AUML, [Odell *et al.* 00], propose de modéliser les protocoles grâce à quelques modifications de la notation UML. Ce sont surtout les diagrammes de séquence qui sont utilisés. C'est d'ailleurs cette notation qui a été choisie par la fipa pour décrire différents protocoles de communication. En fait, ce sont des gabarits de protocoles qui sont représentés (cf. la section 3.4 *Protocoles de communication*, page 26, du chapitre I de ce mémoire pour un exemple).

De la même façon, l'utilisation d'UML prônée par [Oechslein *et al.* 02] permet aussi de modéliser les protocoles de communication. Mais, cette fois-ci, c'est en l'utilisation des diagrammes d'activité[2] et plus particulièrement la possibilité d'y intégrer des échanges de signaux qui servent à modéliser les différents scénarios d'interaction possibles entre les agents du système. Comme le

2. À la différence des diagrammes d'activité d'UML « pur » qui sont associés à des classes, ceux-ci sont associés à des rôles.

soulignent Christoph Oechslein et ses coauteurs, l'utilisation de la notation UML, et notamment le langage de contraintes OCL, peut permettre le passage (quasi-)automatique des diagrammes vers une mise en œuvre des comportements des agents.

Nabil Hameurlain propose, [Hameurlain 02], de décrire les protocoles d'interaction (donc plus larges que les protocoles de communication, mais il donne les exemples du « contract-net » et de sa version itérée) avec des réseaux de Petri et d'utiliser des composants (*i. e.*, des parties de protocoles) pour construire et composer les réseaux de Petri représentant les protocoles.

4. Solution

4.1. Analyse

La solution apportée par ce motif lors de la phase d'analyse peut prendre deux formes. La première est de penser à repérer les échanges de messages entre les entités du système considéré et à les voir comme des conversations, à essayer de les abstraire pour en tirer des protocoles de communication.

La seconde façon d'utiliser ce motif est parallèle à la première. Il s'agit cette fois de se servir d'un catalogue de protocoles de communication (tel celui proposé par la fipa) comme d'une grille d'analyse du système. Cette grille peut aider à repérer plusieurs éléments du système analysé. Tout d'abord, elle permet d'identifier les protocoles utilisés (ou les types de protocoles utilisés). Ensuite, et grâce à cela, elle peut faire ressortir les différents rôles et comportements qu'ont les agents. Enfin, elle permet un repérage, et même une typologie, des messages échangés lors des conversations qui suivent ces protocoles.

4.2. Conception

Comme il est déjà reconnu en génie logiciel, plus tôt une caractéristique du système est repérée, mieux elle sera traitée. De ce fait, les protocoles reconnus lors de la phase d'analyse seront d'autant mieux décrits lors de la conception.

La solution proposée par ce motif est de représenter les protocoles de communication dans les diagrammes représentant le système conçu, comme c'est le cas pour les méthodes citées plus haut, et de la façon qu'AUML ou Oechslein *et alii* proposent. Cette représentation permet, comme c'est le but recherché dans les éditeurs UML, d'automatiser le passage des diagrammes au code.

Une autre proposition de ce motif est de décomposer la communication des agents en conversations catégorisées. Les échanges auxquels peut participer un agent peuvent être décomposés en instanciations de protocoles de communication connus.

5. Discussion

5.1. Analyse

La prise en compte de l'existence des protocoles de communication lors de l'analyse du système simplifie grandement l'identification des différents rôles,

des différentes interactions et des types de message que les agents peuvent s'échanger.

De plus, à l'instar des motifs, les différents types de protocole forment un vocabulaire qui permet de simplifier l'analyse du système, la documentation du logiciel et la communication à l'intérieur de l'équipe de développement. Il est beaucoup plus simple de parler de « contract-net » ou d'« enchère hollandaise » que d'expliquer à chaque fois le type de conversation que ces protocoles représentent.

5.2. Conception

Un désavantage de réifier les protocoles dans les diagrammes de conception est, comme c'était déjà le cas pour les schémas d'organisation, qu'ils diminuent l'ouverture, les possibilités d'extension et le choix des agents en prévoyant trop de choses à l'avance. Or, en génie logiciel, il est toujours mieux de spécifier tôt. De plus, connaître les protocoles disponibles permet le choix, alors que ne pas les connaître n'étend pas le choix, cela le rend impossible.

6. Motifs associés

Le motif *Schémas d'organisation* (page 79) est bien sûr un motif associé de *Protocoles*, ces deux motifs ayant de nombreux concepts en commun (notamment le très important concept de *rôle*).

Le motif *Influences* (page 105) est aussi lié à celui-ci. En effet, les communications entre les agents peuvent être considérées comme des influences.

Le motif *Marques* (page 93) peut servir de support à ce motif. En effet, il propose un moyen de communication simple entre les agents : le dépôt de marques et leur perception, et un type générique de messages : les marques.

Enfin, les motifs fils de ce métamotif, c'est-à-dire les différent motifs expliquant chacun le contexte d'application et les contraintes résolues par tel ou tel protocole, font aussi partie de cette section.

⋆⋆⋆

Motifs métaphoriques

Ces motifs sont issus de l'application au domaine des SMA de techniques venant d'un domaine totalement différent de celui des problèmes abordés par les motifs. Ces motifs consistent donc en l'utilisation fréquente et détournée (d'où le terme de *métaphores*) de techniques et de concepts validés dans d'autres domaines.

Nous proposons deux motifs métaphoriques :

Marques : applique la métaphore des phéromones et du marquage de l'environnement.

Influence : applique la métaphore des forces physiques et des forces d'action à distance.

Il est à noter que la métaphore sociale utilisée dans le motif *Schémas d'organisation* pourrait aussi nous inciter à le placer parmi ces motifs métaphoriques. Toutefois, le fait que la métaphore sociale est fondatrice du paradigme agent et l'aspect « méta » de ce motif nous ont conduit à le placer dans la section précédente.

Motif

Marques

1. Synopsis

Les agents laissent des traces dans l'environnement pour se servir de l'environnement comme d'une mémoire supplémentaire, partagée et située.

2. Forces

– Les agents ont des capacités mémorielles limitées.

– Les agents ont des capacités communicatives limitées.

– Les agents sont situés : il existe des contraintes concernant leurs positions, leurs déplacements, il y a une temporalité dans le système (évolutivité).

– Il y a des contraintes sur les ressources utilisables par les agents : vitesse, autonomie énergétique, durée limitées. Ils ne peuvent passer leur temps à se rechercher pour communiquer.

– Les informations à stocker ou à partager ont un caractère spatial, elles ont une *localité* (*i. e.*, elles ne sont pertinentes que localement).

3. Origines

L'origine du modèle des marques est avant tout biologique. C'est en effet un principe utilisé par de nombreuses espèces d'insectes et dont l'exemple le plus connu est celui des fourmis : chaque fourmi dépose une petite quantité de substances chimiques (appelées « phéromones ») lui permettant à la fois de marquer son passage (mémoire externe) et d'indiquer à ses congénères le chemin qu'elle a pris (moyen de communication). En fait, les phéromones induisent directement un comportement spécifique chez l'individu qui les perçoit (c'est une *réaction* dans le sens de l'opposition *réactif–cognitif* présente dans la littérature agent). Elles agissent comme les transmetteurs chimiques du système nerveux, les hormones (d'où le nom de phérormone ou phéromone) ; ainsi les fourmis utilisent-elles aussi les phéromones pour communiquer entre elles sur une courte distance.

L'utilisation la plus connue des phéromones est bien sûr celle de la piste, citée plus haut, mais d'autres besoins sont satisfaits par leur usage. Ainsi, les termites s'en servent-elles pour la construction de leurs gigantesques termitières. Chaque termite produit le ciment par régurgitation et y intègre de ce fait une petite quantité de phéromones, l'endroit où ce matériau sera déposé dépend de la quantité de ciment porté par la termite mais aussi et surtout des phéromones perçues par la termite, plus la charge est importante et plus la quantité de phéromones est importante, plus elle a de chances de déposer ce qu'elle porte. Ainsi, si la termite est à sa charge maximale, dépose-t-elle le ciment là où elle se trouve ; si la quantité de phéromones est maximale, elle dépose ce qu'elle porte, quelle qu'en soit la quantité. Avec un nombre suffisant de termites fonctionnant suivant ces règles simples, il apparaît rapidement des piliers dont les extrémités ont tendance à se rapprocher pour former des arches et ainsi un nouveau plancher sur lequel le processus se renouvelle, [Bonabeau & Théraulaz 94, p. 58 sq.]. Il s'agit ici d'une fonction stigmergique : aucune termite ne donne d'ordre ni de plan, il s'agit d'une action collective qui n'a pas lieu s'il n'y a pas assez de participants.

Une autre fonction remplie grâce à des phéromones est celle du recrutement. En effet, lorsque les fourmis déposent des phéromones sur leur chemin elles ne le font pas de la même façon dans les deux sens : lorsqu'elles cherchent de la nourriture, elles ne font que repérer leur chemin, lorsqu'elles reviennent à la fourmilière après avoir trouvé une source de nourriture, leurs traces sont proportionnelles à la quantité et à la qualité de la nourriture qu'elles ont trouvé. Ainsi, les autres fourmis suivront-elles d'autant plus cette piste que la source est importante. La piste sert donc au recrutement d'autres fourmis pour aider l'inventrice à la récolte.

En plus des pistes, le marquage phéromonal permet aussi de délimiter le territoire. Cette fonction est d'ailleurs fortement répandue puisqu'elle est utilisée par des espèces plus évoluées[1] que des insectes, comme de nombreux mammifères (félins, rongeurs, etc.) : ils apposent leur *odeur*, le plus souvent pour délimiter un territoire. Les phéromones sont des substances chimiques assimilables à des odeurs car elles sont perçues par le même genre de capteurs (chémorécepteurs) que ceux qui composent le système olfactif, par contre elles fonctionnent bien sur un mode plus réactif puisque, bien qu'il semblerait que nous (les êtres humains) y soyons sensibles, nous ne nous rendons pas compte de leur existence (nous n'en formons pas d'images mentales, comme nous le faisons des autres odeurs).

Les phéromones sont aussi utilisées pour l'identification individuelle ou sociale. Chaque individu marque son appartenance à un groupe social (chaque fourmilière, termitière...) grâce à un bouquet[2] de phéromones particulier à ce groupe. De cette façon, deux fourmis de la même fourmilière s'accepteront mais rejetteront toute fourmi (même de la même espèce) n'appartenant pas à leur société myrmicienne. De la même façon, certaines espèces utilisent les phéromones pour attirer les congénères de l'autre sexe. Ainsi, certains papillons femelles émettent-elles des phéromones perçues à plusieurs kilomètres de distance par les papillons mâles de la même espèce, leur permettant ainsi de se trouver

1. Nous entendons par là qu'elles ont des fonctions biologiques individuelles plus évoluées (surtout au niveau du système nerveux).
2. Les phéromones sont utilisées en bouquet, cela permet une reconnaissance plus sûre que s'il n'y avait qu'une seule odeur.

(c'est bien plus rapide qu'une petite annonce ;o), dans ce cas, l'information est diffusée, disséminée, par le vent.

Cette fonction d'identification va jusqu'à la communication, *e. g.*, les larves formiques émettent certaines phéromones lorsqu'elles ont faim ou ont besoin d'être toilettées. Cette fonction de communication est notamment utilisée pour la communication d'urgence, utilisée là encore par de nombreuses espèces entomologiques, comme les fourmis ou les abeilles. En effet, dans le venin de ces dernières entre en composition une phéromone qui a la capacité d'énerver et de pousser à l'attaque les abeilles avoisinantes. La rémanence des phéromones permet souvent à ce message d'alerte de survivre à l'individu émetteur.

De la même façon, les miettes de pain du petit Poucet du conte de Perrault ou le mythique fil d'Ariane sont des *marques* et relèvent du même principe : utilisation de l'environnement comme mémoire supplémentaire, collective et située. La signalisation routière utilise les mêmes effets : ce sont des informations placées là où elles sont pertinentes (avertissement de la présence d'un danger, indication de direction, etc.). L'on pourrait même aller jusqu'à l'écriture (*i. e.*, dépôt d'une information sur un support physique dans le but de la communiquer ou de s'en souvenir) mais ce serait par trop généraliser : les écrits sont le plus souvent sur des supports mobiles — quand ils ne le sont pas, ils sont plus assimilables à la signalisation dont nous venons de parler — il s'agit donc d'une communication quasi directe.

Ce principe de marquage est donc fortement répandu et d'autant plus utilisé que les systèmes sont organisés. En effet, comme le remarque Guy Théraulaz dans [Bonabeau & Théraulaz 94, p. 41], lorsqu'il y a peu d'interactions dans un groupe, que ce soit en quantité ou en qualité, il y a peu de coordination entre les agents, peu de spécialisation fonctionnelle et peu d'intégration sociale ; la réciproque étant vraie elle aussi : plus il y a d'interactions, plus le groupe est une société, plus il est organisé.

4. Usages

La communication par marquage et plus particulièrement par phéromones a été étudiée dans [Booth & Stewart 93]. Les performances d'un SMA composé de robots trieurs qui communiquent par une phéromone simple sont comparées au même système dans lequel les agents ne communiquent pas. Les phéromones structurent l'espace et les agents suivent cette structure pour déplacer les phéromones, il y a création d'un cercle vertueux qui permet au SMA de trier l'environnement.

La métaphore des marques est utilisée dans Manta, la simulation d'une colonie de fourmis développée dans [Drogoul 93a]. Il s'agit ici de coller à la réalité entomologique : les fourmis ont un comportement réactif dirigé par des *stimuli*, ceux-ci pouvant être internes ou externes. Les *stimuli* externes sont souvent des messages envoyés par d'autres fourmis, ils ont une portée limitée et une force (un poids) faiblissant avec l'augmentation de la distance entre l'émetteur et le récepteur. Ces *stimuli* externes représentent les phéromones identifiantes des divers objets (œufs, larves, fourmis, reine) et leurs états (affamé, repus, etc.). L'aspect entropique de l'utilisation des marques n'est pas abordé ici. Par contre, [Klügl *et al.* 98], en intégrant la trophallaxie (échange d'énergie entre deux fourmis) et l'existence de fourmis-réserves (fourmis pleines de miel et qui servent à la fois de réservoirs vivants et de nœuds dans le réseau des communi-

cations et de dissipation de l'énergie), démontrent l'utilité d'intégrer une notion d'énergie et d'échange d'énergie dans le système : la fourmilière simulée est plus performante et possède un comportement plus proche de la réalité.

Pour rester dans le domaine des simulations de sociétés d'insectes, nous citerons l'exemple de Mari Nakamura et Koichi Kurumatani, [Nakamura & Kurumatani 96], qui, en introduisant une désensibilisation des organes perceptifs après avoir perçu une dose importante de phéromones, obtiennent une structuration du fourragement simulé par leur SMA : suivant les seuils utilisés, le recrutement et donc la répartition des fourmis sur le terrain varient.

Ndedi Monekosso, Paolo Remagnino et Adam Szarowicz, utilisent les phéromones pour augmenter la vitesse de convergence d'un algorithme de *Q-learning* multi-agent, [Monekosso *et al.* 02]. L'utilisation du motif *Marques* permet aux agents de communiquer, de se coordonner et de coopérer dans l'apprentissage, sans conséquence lourde sur le temps de calcul (la communication ne se faisant pas par envoi de messages). Les agents ont un facteur de croyance pour chaque *marque* qu'ils perçoivent ; cela leur permet d'affiner de meilleure façon leur comportement réactif. Comme dans l'exemple précédent, les auteurs montrent qu'un taux d'évaporation trop bas, ou un taux de dépôt trop élevé de la part des agents, mène à des résultats plus pauvres.

Certaines plates-formes, dont MadKit, proposent un modèle de marquage. Pour MadKit, il s'agit d'un ensemble de classes permettant de gérer facilement la production, la propagation et l'évaporation d'odeurs *(flavors)*. Chaque agent peut créer ses propres odeurs et évoluer dans un environnement qui se chargera de leur devenir. Il existe aussi des méthodes pré-codées permettant à un agent de suivre un gradient d'une odeur particulière.

Sven Brueckner, *in* [Brueckner 00], explique l'utilisation et la mise en œuvre de la *Pheromone Infrastructure (PI)* pour la coordination stigmergique d'agents simples à l'intérieur d'un environnement simulé. Il a ainsi développé un modèle formel générique permettant la prédiction de la dynamique des interactions phéromonales entre les agents. Le but est donc de pouvoir concevoir un SMA utilisant la *Pheromone Infrastructure* et dont le concepteur pourrait prévoir le devenir grâce au modèle formel de la *PI*. Il s'agit aussi d'utiliser ce modèle formel pour concevoir les comportements interactifs des agents : l'agent utiliserait ses connaissances du modèle formel pour interagir — pour savoir quand et comment lire les phéromones déposées et quand et comment il doit déposer ses phéromones pour induire tel ou tel comportement chez ses interlocuteurs.

In [Bonabeau & Théraulaz 94, p. 216], Hugues Bersini, Pascale Kuntz & Dominique Snyers utilisent des agents réactifs simples qui manipulent des marques phéromonales pour résoudre un problème du type voyageur de commerce. Cet exemple montre à la fois l'utilité du motif *Marques* et des SMA pour résoudre des problèmes qui n'ont rien à voir avec la simulation ou la vie artificielle.

In [Holland 95], John H. Holland décrit le mécanisme de *tagging* qui consiste, pour les agents, à porter des signes qui provoquent l'attraction ou la répulsion des autres agents. Bien que l'immunologie soit à la base de son idée, il utilise ici la fonction d'identification des phéromones utilisée par les animaux. Cette fonction d'identification peut être poussée jusqu'à permettre à un agent de reconnaître la fonction d'un autre agent au sein du groupe. L'on peut faire le parallèle avec l'utilisation d'insignes et d'uniformes dans les sociétés humaines.

Pascal Ballet, *in* [Ballet 00], utilise le motif *Marques* pour sa capacité à

structurer l'environnement. Dans son application des SMA à la reconnaissance de contours, chacun de ses agents augmente le contraste local de l'image, ce qui permet, grâce aux passages successifs d'autres agents, de faire ressortir les contours présents dans l'image. Chaque agent, en modifiant ou non l'environnement qu'est l'image, induit une modification ultérieure de l'environnement par les autres agents, il y a ici aussi création d'un cercle vertueux qui réalise la fonction du SMA. Pascal Ballet démontre que chaque modification apportée par un agent doit être limitée pour éviter la génération de bruits et bien évidemment non nulle pour que l'image soit modifiée.

Serge Fenet & Salima Hassas, dans [Fenet & Hassas 98], utilisent le motif *Marques* pour équilibrer la charge d'un réseau : des agents messages transportent et déposent des *marques* indiquant la charge des nœuds voisins. De la même façon, dans [Fenet 01], Serge Fenet utilise ce motif pour développer un IDS *(Intrusion Detection System),* dont le comportement est calqué sur les réactions de défense des fourmilières et des ruches.

Nous trouvons donc différentes fonctions des marques utilisées dans des applications informatiques. Les marques servent à optimiser le SMA, soit en lui permettant d'économiser des ressources (l'environnement sert de mémoire partagée), soit en lui permettant de structurer le système (Jacques Gervet appelle cela la « construction d'une structure signifiante » *in* [Bonabeau & Théraulaz 94, p. 133]).

Cette structuration peut concerner l'espace : par ségrégation–identification (avec le *tagging* de [Holland 95]), par l'utilisation d'un gradient (dans [Booth & Stewart 93], [Fenet & Hassas 98] et MadKit) ou par marquage de limites et de pistes (dans [Ballet 00], [Bonabeau & Théraulaz 94] et [Nakamura & Kurumatani 96]).

Elle peut aussi concerner l'organisation sociale : par communication (dans [Drogoul 93a] et [Nakamura & Kurumatani 96]), par coordination (dans [Brueckner 00]) ou par communication & entropie (avec [Klügl *et al.* 98], [Monekosso *et al.* 02] et surtout [Van Dyke Parunak 97] dont nous allons parler séant).

En effet, dans [Van Dyke Parunak 97], Henry Van Dyke Parunak développe un modèle entropique des interactions dans un SMA. L'idée principale est d'intégrer une fuite entropique dans le système. En effet, selon la deuxième loi de la Thermodynamique, les systèmes fermés[3] tendent vers le désordre. Or les différents exemples de SMA construits jusqu'ici ont plutôt tendance à s'organiser d'une façon efficace. L'explication est donc qu'il y a de l'énergie qui y est ajoutée de l'extérieur. Ainsi, pour Henry Van Dyke Parunak, les agents s'organisent-ils au niveau macro car leurs actions sont couplées à une dissipation au niveau micro. La fuite d'entropie est organisée du niveau macro (qui est le seul endroit où le travail du système est réalisé) vers le niveau micro.

Les agents produisent de l'énergie par leurs actions et, concurremment mais sans intention des agents, celle-ci est dissipée (soit par des échanges directs entre les agents, soit *via* l'environnement). Cette dissipation crée des champs de flux qui sont perçus par les agents qui le renforcent et dont le comportement est

3. *I. e.*, dont le cycle énergétique est fermé, pas par opposition à l'ouverture des systèmes complexes. Ainsi, un système peut-il être ouvert, en cela qu'il accepte des interactions non fixées à sa conception, et en même temps fermé au sens thermodynamique, c'est-à-dire qu'il évolue dans une enceinte adiabatique.

modifié par ces champs de flux. Les actions des agents font décroître l'entropie alors que la dissipation la fait croître.

Si la dissipation a lieu via l'environnement (l'énergie est représentée par des phéromones), les agents se trouvent à la fois aux niveaux micro et macro de la figure 18. Si la dissipation a lieu entre les agents (l'énergie est représentée par une *monnaie* — H. Van Dyke Parunak donne en effet l'exemple du marché économique dans lequel l'argent bénéficie à ceux qui le font circuler et qui se disperse des acheteurs aux vendeurs, les entrepreneurs perçoivent alors le flux d'argent et s'orientent dans son sens, il en résulte alors des structures autopoïétiques telles des chaînes d'approvisionnement et des centres économiques), alors les agents appartiennent au niveau micro et seules les structures émergentes sont dans le niveau macro.

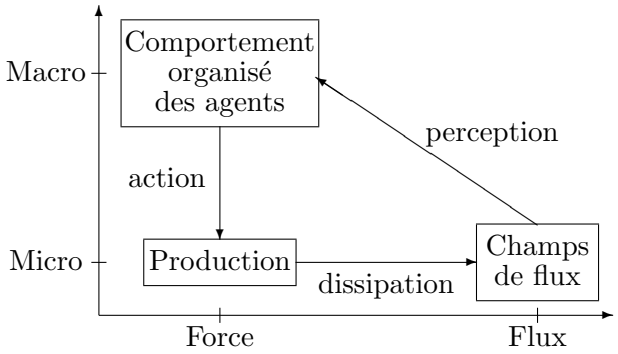


Fig. 18 – Organisation Macro au travers de la dissipation micro, [Van Dyke Parunak 97, p. 21]

Henry Van Dyke Parunak propose ainsi trois caractéristiques nécessaires à ce mécanisme :

– quelque chose doit circuler (matière, information, énergie), soit parmi les agents, soit *via* l'environnement, créant ainsi un gradient ;

– les agents doivent pouvoir percevoir ce gradient et s'en servir pour s'orienter ;

– les actions des agents doivent renforcer le champs (rétromettance positive).

Il est bon de rappeler que la « théorie mathématique de l'information », [Shannon & Weaver 49], mais aussi [Ash 65, Brillouin 59], bien qu'utilisée à de nombreuses reprises, ne peut s'appliquer qu'au niveau physique des moyens de communication (*dixit* Claude Shannon lui-même, cf. l'opuscule cité), et que son rapport à l'entropie[4] telle qu'elle est utilisée en thermodynamique n'est que métaphorique.

Toutefois, la théorie proposée ici a le mérite de proposer une explication et un moyen de reproduire le phénomène d'émergence présent dans de nombreux systèmes, et notamment ceux utilisant la communication *via* l'environnement. C'est d'ailleurs à partir de cette théorie que Jean-Pierre Müller a élaboré une

4. D'ailleurs, Shannon parle d'entropie alors qu'il manipule la néguentropie, en utilisant la lettre H, qui, en général, représente l'enthalpie... Pour des notions claires de thermodynamique, le lecteur pourra se reporter à un bon cours de chimie de niveau terminale.

méthode de conception de systèmes multi-agents, [Müller 98].

5. Solution

L'agent dépose des traces dans l'environnement. Ces traces sont des objets de très faible granularité (*i. e.* au niveau de la mise en œuvre, il s'agit d'objets de structure très simple) et de taille quantique : ces objets ont une taille minimale, voire nulle, par contre, ils ont une certaine force (ou sont présents au même endroit dans une certaine quantité) qui permet à l'agent de le *sentir* à une certaine distance. Une fois déposées, c'est l'environnement qui se charge alors de leur devenir : évaporation, dissémination/propagation, perception.

Ces traces sont perçues dans une zone limitée : le rayon de perceptibilité et les règles gérant l'affaiblissement de la qualité de perception en fonction de la distance sont eux-mêmes fonction de la marque et de la topologie de l'environnement. La perceptibilité est, bien sûr, aussi fonction des organes sensoriels de l'agent : certains de ces objets phéromonaux peuvent ne pas être accessibles de la même façon (quantitativement et qualitativement parlant) à tous les agents, à l'instar des phéromones *naturelles* dont la composition chimique est différente suivant les espèces animales (voire même suivant les groupes sociaux) et qui n'ont donc pas la même signification pour deux individus d'espèces ou même de groupes différents (*e. g.* chaque colonie de fourmi possède sa propre odeur qui signifie « ami » pour les membres de la colonie et « ennemi » pour les autres fourmis ; de la même façon, une phéromone de papillon peut être totalement ignorée par une fourmi).

En outre, l'utilisation des *marques* permet aux agents de communiquer sans s'échanger de messages : déposer une certaine quantité d'une certaine *marque,* c'est modifier l'environnement perçu par tous les agents. C'est par une boucle de rétromettance qu'émergent une communication et une interaction entre les agents. Il s'agit là d'émergence faible, [Jean 97], puisque les agents n'ont aucun moyen de connaître l'auteur du message, non plus qu'ils ne peuvent déterminer si c'est un message — c'est une trace qu'ils perçoivent, ils *réagissent* à la perception de cette trace. Il est à noter que les *marques* sont des objets simples, il n'est pas question dans le motif de base que les agents déposent des messages plus évolués. Cela peut bien sûr être envisagé mais il s'agira alors d'une extension dénaturante du motif *Marques.*

6. Mise en œuvre

Ce motif n'est pas appliqué de la même façon si le système que l'on envisage est une simulation ou si l'environnement du système est le monde réel (robots).

S'il s'agit de faire évoluer des agents dans le monde réel (par exemple des robots), les phéromones sont alors des marques ou des traces de produit que les agents sont capables de percevoir facilement et distinctement. Comme le souligne Jacques Gervet dans [Bonabeau & Théraulaz 94, p. 142] :

> *« Il est peu probable que les mêmes vecteurs soient utilisés dans un système artificiel, et cette différence invite à rechercher les traits qui seront pourtant comparables dans les modes de fonctionnement ; ainsi, l'information chimique est chez les animaux particulièrement adaptée à*

> *donner des informations de type qualitatif, et son utilisation est très répandue pour l'identification individuelle, l'appartenance sociale, voire la reconnaissance topographique de l'environnement (territoire, pistes...). Il est peu probable que cette modalité soit jamais beaucoup utilisée dans un système artificiel ; mais il se peut que la modalité la mieux adaptée à la réalisation de ces fonctions ait à prendre certaines des caractéristiques fonctionnelles que la communication chimique a prises dans le monde animal : réalisation facile de « formes » caractéristiques, permanence du signal après émission, distinction qualitative. Par exemple, le dépôt de traces visuelles permanentes et clairement reconnaissables par un autre agent constituerait un substitut ayant des propriétés analogues et mieux adaptées aux caractères propres des agents artificiels actuellement concevables. »*

Si les agents évoluent dans un monde simulé, alors il faut que l'environnement y soit proactif pour qu'il puisse gérer le cycle de vie des marques : évaporation, dissémination (par le vent, le déplacement par d'autres agents), le masquage (sur-apposition d'autres marques), la disparition (voire l'enlèvement, comme les miettes de pain du petit Poucet).

C'est l'environnement qui doit s'occuper de ces marques, comme cela se passe dans la réalité pour les phéromones. Une solution consisterait à simuler le phénomène d'évaporation par l'inclusion d'une date dans la structure représentant la marque, cette date servirait à l'agent pour juger de la perception de la marque. Mais cette solution n'est conceptuellement pas acceptable car elle déplace l'activité d'évaporation dans l'agent alors qu'elle se situe dans l'environnement de l'agent. D'autres solutions du même type (*i. e.*, insérer des informations dans la structure représentant les marques pour que l'agent décide s'il doit ou non les percevoir et de quelle façon), ne sont pas acceptables pour les mêmes raisons. Le principe est de déléguer à l'environnement la charge de mémorisation et de gestion du cycle de vie de ces mémoires que sont les marques.

Les marques ont un rayon de perceptibilité défini : un agent ne peut percevoir la présence d'une marque à un certain endroit que s'il en est suffisamment proche. Bien sûr, cette notion de proximité dépend fortement de la topologie intrinsèque de l'environnement (au sens mathématique du terme : notions de voisinage et de distance).

Les marques ont une certaine volatilité : la quantité de marques présente en un point diminue en fonction du temps, il y a évaporation. Cela peut se mettre en œuvre de deux façons : activement ou à l'opportuniste. La technique active consiste à ce que l'environnement soit proactif et s'occupe, à intervalles réguliers, de faire diminuer la quantité de marques présente en chaque lieu. La seconde technique, opportuniste, consiste à calculer la quantité de marques accumulée en un lieu défini seulement lorsqu'un agent est susceptible de la percevoir, c'est-à-dire lorsque l'environnement envoie les percepts à l'agent. Cette dernière technique demande un modèle mathématique précis des dynamiques liées à l'évaporation et à la diffusion des marques.

Une caractéristique parfois utile des marques est leur diffusion dans l'environnement : si une quantité importante de marques se trouve en un point, les points voisins[5] récupèrent une partie de cet amas de marques. Cette diffusion peut bien entendu suivre toutes les règles que l'on désire, les plus simples

5. Encore une fois, la notion de voisinage dépend de la topologie de l'environnement.

étant bien évidemment uniformes (*i. e.*, chaque *voisin* reçoit la même quantité de marques) mais l'on peut aussi imaginer d'autres règles : en faisant intervenir le vent (ou un phénomène similaire) par exemple. Cette diffusion permet la formation d'un gradient de la quantité, de la force, de cette marque. Ce gradient peut ainsi être suivi par les agents et le système pourra alors bénéficier des avantages donnés par le modèle de H. Van Dyke Parunak présenté page 97. Cette caractéristique peut, comme la précédente, être mise en œuvre par les mêmes deux techniques : active et opportuniste.

Une dernière caractéristique que l'on peut intégrer est le masquage : une marque peut, si elle est présente en quantité suffisamment importante, masquer une autre marque présente au même endroit en quantité plus faible. Il s'agit d'une règle entrant dans le calcul par l'environnement de ce qui sera donné aux organes sensoriels de l'agent.

7. Discussion

Bien sûr, il existe d'autres solutions que le motif *Marques* pour régler les problèmes de communication et de ressources limitées mais ce ne sont pas réellement des solutions en ce qu'elles ne résolvent pas entièrement le problème, posent d'autres problèmes ou n'intègrent pas toutes les contraintes.

Une solution évidente au problème de la communication serait de faire que les agents se déplacent pour qu'ils se rapprochent et communiquent, mais les déplacements ne sont jamais gratuits, sans parler de la difficulté de se retrouver si on ne peut communiquer à grande distance.

D'autre part, les informations dont l'agent a besoin pourraient être tout simplement rangées dans un coin de la mémoire de chaque agent ; mais cette solution ne prend pas en compte la contrainte de limitation de la capacité mémorielle des agents, encore moins que la structuration possible par l'environnement.

Cette information pourrait aussi être partagée. Mais le partage n'est pas toujours possible, il faut un accès rapide à l'information pour chaque agent et ceci n'est pas réalisable dans le cas de robots ou d'agents réellement autonomes dans le système d'exploitation (les mécanismes de gestion de mémoire partagée sont gourmands en communication). Dans le cas de robots, il est toujours possible d'utiliser leur base de déploiement (fourmilière) mais cela oblige alors les agents à revenir fréquemment à la base ou à utiliser un moyen de communication à distance avec celle-ci.

De plus, ces deux solutions ne permettent pas d'intégrer facilement le fait que ces informations peuvent n'être pertinentes que localement.

Elles posent aussi le problème de la mise à jour des informations contenues dans la mémoire de l'agent ou dans la mémoire partagée. Nous voyons alors que la conclusion de Rodney Brooks : *« use the world as its own model »*, [Brooks 91b], prend ici toute sa signification.

Il y a aussi des problèmes liés à l'utilisation de ce motif, [Drogoul & Fresneau 98] estime que sa mise en œuvre dans le monde réel, c'est-à-dire avec des robots, n'est pas si facile : il y a des problèmes pour trouver le bon *produit* — cela pose des problèmes au niveau des capteurs, de leur sensibilité et de leur précision. L'on pourrait répondre que la robotique est une science qui progresse rapidement, mais il vrai que des capteurs aussi performants que ceux des insectes ne sont pas encore à notre portée (on notera l'approche novatrice

de [Kuwana *et al.* 96] qui utilise, avec succès, une antenne de papillon comme capteur de phéromones pour un petit robot expérimental).

Par contre, comme nous le soulignions plus haut en citant Jacques Gervet, rien ne nous oblige à utiliser des substances aussi subtiles que les phéromones, de simples marquages visuels ou magnétiques suffisent. Enfin, nous estimons que la part d'imprécision ou en tout cas d'indétermination est inhérente aux systèmes ouverts et surtout aux SMA qui sont censés être adaptatifs face à leur milieu et aux bruits et interférences qui peuvent intervenir tant sur leurs organes perceptifs que sur leurs organes effecteurs.

En ce qui concerne les difficultés de la simulation, l'on ne peut évidemment pas négliger les ressources calculatoires et mémorielles nécessaires (cf. la section précédente, la discussion sur la volatilité et la dissémination des marques).

Toutefois, en plus de résoudre les contraintes que nous avons exposées dans la section *Forces,* ce motif apporte de gros avantages. En effet, les marques sont des messages simples — tant par leur forme que par leur manipulation — et le fait qu'elles soient déposées permet d'intégrer une notion de localité dans l'information qu'elles portent. Cette intégration est en outre faite de façon indirecte pour l'agent : il n'y a pas besoin d'un système de coordonnées, les marques sont placées là où elles ont une signification. Cette absence de système de coordonnées peut être primordiale pour l'agent, la localisation précise nécessite des moyens de repérage complexes (*e. g.* le GPS nécessite plusieurs dizaines de satellites) sans compter que le concept même de coordonnées, bien qu'il soit simple et *naturel* en mathématiques et en traitement symbolique, est finalement assez compliqué à mettre en œuvre.

À moins d'être le seul à pouvoir reconnaître une marque, et de savoir qu'il est le seul, un agent ne peut déterminer celui qui l'a déposée. Le mécanisme est détaché de l'identité propre des porteurs et déposants de marques. C'est pour cette raison que ce mode de communication est très particulier et mène plutôt à parler d'intelligence collective : chaque agent est une partie de cette intelligence ; le nombre d'agents influe sur l'aspect chaotique du système : un seul agent et le système est déterministe, plusieurs agents et il devient chaotique et un comportement collectif peut en émerger (on pourra se référer à [Booth & Stewart 93] pour un exemple d'utilisation de marques pour une organisation spatiale du système par des agents simples).

Pour finir, il ne faut pas confondre ou vouloir inclure ce principe de marquage dans le concept de tableau noir. En effet, dans un tableau noir, il y a :

– un contrôle centralisé des ksars[6] ;
– une sémantique importante liée aux données écrites ;
– une répartition spatiale des droits d'écriture et de lecture ;
– une *non-localisation* des ksars ;
– une certaine pérennité des données : elles ne sont que modifiées, jamais supprimées.

Alors que dans le motif *Marques,* il y a :

– un contrôle décentralisé (cf. les exemples de stigmergie comme la construction d'une termitière, etc.) ;

6. *Knowledge Source Activation Records,* les structures permettant d'appliquer les connaissances.

– la possibilité pour un agent de déposer des marques sans qu'il en connaisse la signification, sans même savoir qu'il le fait ;

– la possibilité pour un agent de ne pas réagir à certaines marques ;

– la possibilité pour un agent de percevoir et de déposer des marques à tout endroit ;

– une forte localisation des marques déposées (elles ne sont pertinentes que localement) ;

– un grand rôle pour l'environnement qui modifie les marques, les déplace, les répand, les efface, les transforme.

8. Motifs associés

Le motif *Influences* (présenté dans ce mémoire, page 105) peut profiter à ce motif pour la mise en œuvre de la diffusion des marques dans l'environnement et pour leur perception (une *marque* est une source d'une force d'attraction/-répulsion, une influence positive ou négative).

Le métamotif *Protocoles* (page 87) et ses motifs fils (les protocoles de communication), peuvent aider à l'utilisation de ce motif. Ils peuvent aider à spécifier quand les agents doivent déposer des marques, comment ils doivent y réagir et quels sont les différents types de marques échangées entre les agents (chaque marque correspond à un message, chaque type de marque à un type de message).

⋆⋆⋆

Motif

Influences

1. Synopsis

Découpler les causes et les effets des actions primitives permet de contourner les problèmes de conflits d'actions simultanées des agents sur l'environnement.

2. Forces

– Il existe des influences globales (champs de force dans l'espace).

– Plusieurs agents peuvent agir simultanément sur un même objet (agent ou non).

– Un agent peut influencer ou perturber un autre agent sans être obligé de passer par des mécanismes cognitifs de ce dernier (*i. e.*, pas par envoi asynchrone de message, les niveaux physique et social sont bien distincts).

3. Origines

Le concept de forces tel qu'il est utilisé en sciences physiques est la principale source d'inspiration de ce motif : les objets ne se modifient pas les uns les autres directement, ils génèrent des forces qui, combinées, ont une action.

4. Usages

Ce motif reprend le modèle « actions et influences » exposé et formalisé par Jacques Ferber et Jean-Pierre Müller dans [Ferber & Müller 96].

En 1997, nous avions incorporé le modèle « actions et influences » dans le modèle organisationnel de B. Durand, [Sauvage 97], pour intégrer des comportements déclenchés volontairement par d'autre agents : les *interactions.* Le modèle initial ne possédait que des comportements *proactifs* (volontaires et *habituels*), *réactifs* (en réaction à un stimulus interne) et *fonctionnels* (esclaves d'un comportement d'un autre rôle). Ces *interactions* ont notamment servi à

la mise en œuvre des communications par envoi de message (un message étant une *influence*), les simulations réalisées jusqu'alors étaient dépourvues de communication directe car elles étaient surtout réactives.

Richard Canal, *in* [Canal 98], a étendu le modèle initial des *influences* pour traiter le cas des réactions en chaînes.

Pablo Gruer, Vincent Hilaire & Abder Koukam, dans [Gruer *et al.* 00], utilisent les *influences* dans leur modèle d'interaction entre les agents et entre les agents et leur environnement.

Matthieu Amiguet, *in* [Amiguet 03], a intégré le motif *Influences* dans sa plate-forme multi-agent moca, construite au dessus de MadKit (cf. le chapitre II, page 33). On trouvera un exemple d'utilisation de moca et de son modèle d'influences dans [Müller *et al.* 01], où José Baez développe une simulation de type proie–prédateur grâce au modèle organisationnel de moca et à sa plate-forme.

5. Solution

Pour appliquer ce principe aux SMA, il faut séparer la cause et l'effet de chaque action.

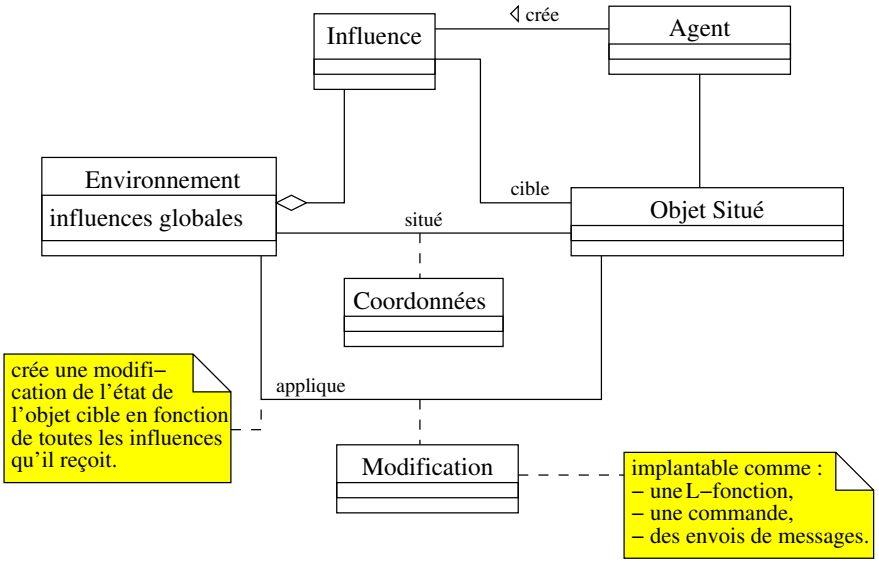


Fig. 19 – Premier diagramme conceptuel du motif *Influences*

La figure 19 présente les concepts en jeu dans ce motif lors de son application dans le domaine des forces physiques (*i. e.*, les influences apportent des modifications physiques aux objets qui les reçoivent).

– *Objet Situé :* définit la classe des objets spatialement et temporellement situés dans l'environnement et pouvant donc servir de cibles.

– *Agent* : exerce des influences sur les objets qui l'entourent.

– *Environnement :* représente l'environnement dans lequel les agents et différents objets sont situés ; en tant qu'objet « global », il connaît les influences globales (gravité, bruit, etc.) applicables aux différents objets ; il détermine aussi

la modification à apporter à l'état[1] des objets du monde en fonction des différentes influences en jeu.

– *Influence :* représente une cause possible d'un changement de l'état[1] du monde, cela va des *forces* dans le sens utilisé en physique aux communications entre agents.

– *Modification :* détermine un changement d'état[1] dans un objet situé — suivant les choix de mise en œuvre, il pourra s'agir d'un script, d'une méthode de la classe Objet Situé à appeler, d'une commande [Gamma *et al.* 93, p. 233], etc.

L'environnement reçoit les influences que les agents produisent et modifie l'état des objets du monde, la transition étant le résultat des différentes influences appliquées aux objets (celles des agents, les influences globales et les contre-influences[2] des objets cibles).

Les influences peuvent aussi s'appliquer à des pertubations autres que les forces physiques. En effet, si c'est la gestion des forces physiques qui bénéficie tout particulièrement du motif *Influences*, on peut tout à fait l'appliquer à d'autres types de « perturbation », comme par exemple les différents *stimuli* que peuvent percevoir les agents ou les messages qu'ils s'échangent.

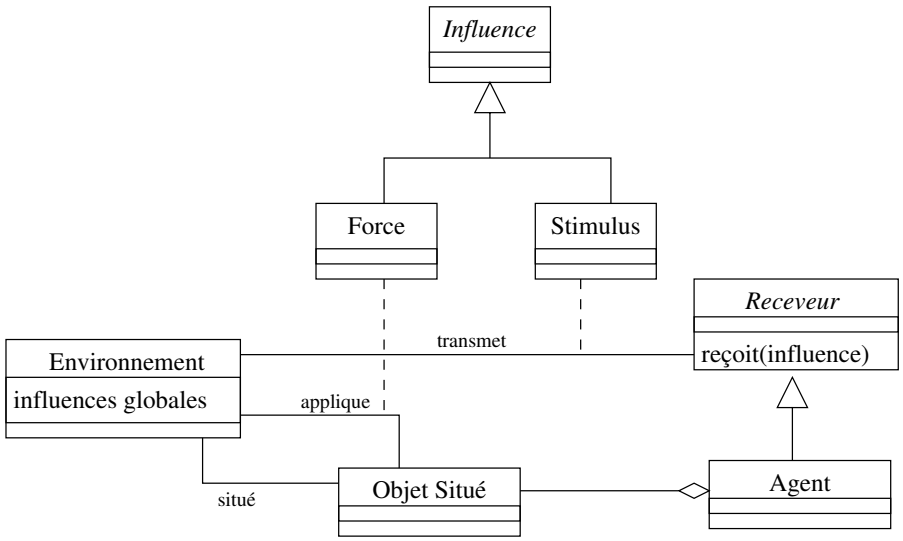


Fig. 20 – Second diagramme conceptuel du motif *Influences*

Dans ce cas plus général, nous avons besoin de deux sortes d'influences : les influences que le receveur subit et celles qu'il traite plus ou moins consciemment — il peut en effet s'agir de *stimuli* auxquels l'agent réagit ou de messages qu'il va analyser. Nous obtenons donc la structure présentée sur la figure 20.

L'environnement applique directement les forces sur leur cible et transmet les autres perturbations (que nous avons ici appelées *stimuli* mais qui peuvent aussi correspondre à des messages) au receveur. Le receveur peut traiter les influences qu'il reçoit de la manière qu'il veut. Cela peut être de manière synchrone

1. *État* au sens large : l'ensemble des valeurs prises par les attributs de l'objet, à opposer au sens strict d'état dans un automate.
2. C'est-à-dire les influences réactives des objets influencés.

(traitement immédiat) ou asynchrone (avec, par exemple, la gestion d'une liste d'attente de messages, etc.).

6. Mise en œuvre

In [Canal 98], R. Canal profite de l'extension du modèle initial développé dans [Ferber & Müller 96] pour exposer un algorithme qui permet, à partir de l'état du monde et de l'ensemble des influences (appelées *propositions* par R. Canal) existant à un instant t, de calculer l'état du monde à l'instant suivant.

Le motif *Interpreter*, [Gamma *et al.* 94], peut aussi permettre de mettre en œuvre ce motif : il permet de reporter l'exécution des actions à appliquer à l'environnement. En effet, dans ce motif, les actions sont décrites dans un langage simple qui sera interprété par le receveur, c'est-à-dire ici l'environnement. Le principal intérêt est d'avoir une couche applicative dans l'environnement (l'interprète) qui pourra combiner les influences et donc calculer leurs effets.

Les effets des influences peuvent aussi être calculés par des gradients, par des équations pour certaines influences — comme les champs de forces —, ou par la technique du gradient à vagues. Sur la figure 21, l'on peut voir une situation simple d'un agent (R) subissant les effets d'influences provenant d'une source (S), le gradient est figuré par le dégradé de gris.

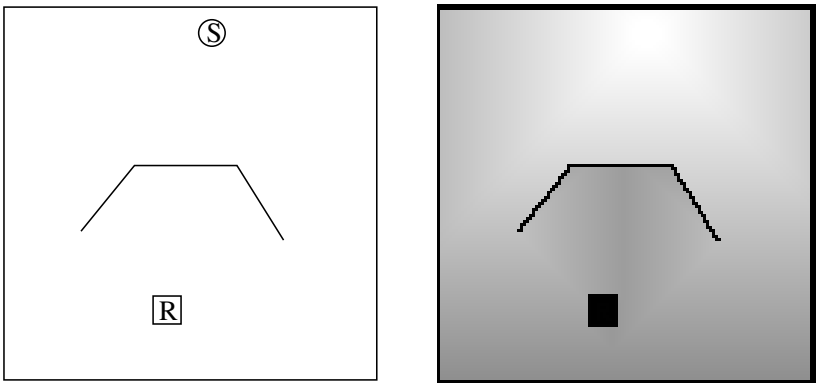


Fig. 21 – Situation et gradient (S est la source du gradient, R un agent influencé par S)

Dans la technique du gradient à vagues, le gradient est calculé par vagues successives, du point culminant aux *minima* globaux. L'espace dans lequel le gradient doit être calculé est discrétisé (seulement pour le calcul du gradient), le point de départ du gradient est le point culminant de l'influence, c'est-à-dire sa source. Pour chaque cellule spatiale voisine de la source, la valeur de l'influence est calculée comme étant légèrement inférieure[3] à celle de la source. Les obstacles à la propagation de l'influence ont une valeur nulle. Les cellules dont la valeur vient d'être calculée servent de sources pour la vague suivante de calculs.

La technique du gradient à vagues est une très bonne technique pour calculer le chemin le plus court dans un monde dynamique avec obstacles, comme on

3. Cela demande à discrétiser aussi les valeurs du gradient ; chaque vague réduit la valeur des cellules d'un *quantum*.

peut le voir sur les figures 21 & 22. Le problème qu'elle engendre vient du temps nécessaire à son calcul ainsi que sa *non-localité* : elle demande de connaître l'espace tout le long du chemin qui sera pris.

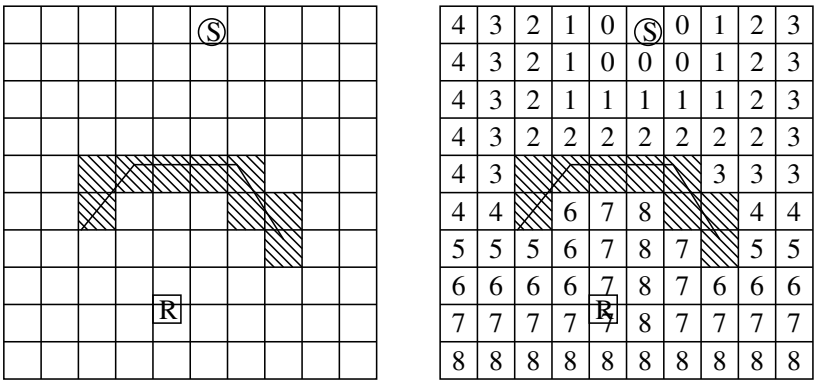


FIG. 22 – Discrétisation et calcul par vagues (8-voisinage)

7. Discussion

Les conflits d'actions concurrentes des agents sur l'environnement (plus particulièrement sur les objets qui les entourent) sont fréquentes et difficilement traitables. De plus, il est utile de pouvoir intégrer des influences globales dans notre système.

Imaginons deux agents, situés chacun d'un côté d'une porte battante, qui poussent celle-ci simultanément. Dans le cas où chaque agent agit directement sur la porte, *i. e.* modifie directement son environnement, il y a un conflit entre les deux actions. Tout d'abord, il y a la concurrence dans l'utilisation de la *ressource porte*. Mais ce conflit peut se gérer par une exclusion mutuelle sur la ressource. Par contre, chaque agent poussera la porte à son tour, croyant avoir réussi son action ; ce qui, au mieux, ne sera vrai que pour un seul d'entre eux.

De plus, dans le cas d'une action continue (par exemple celui de deux engrenages qui en entraînent un troisième, passif), il n'y a pas d'équilibre mais une oscillation permanente ; ce qui peut poser d'énormes problèmes, notamment le fait que l'agent ne s'apercevra pas de l'inutilité de son action mais, surtout, cela ne reflète absolument pas le comportement d'un monde *réaliste*[4][5].

Ici, l'agent n'opère plus d'action directe sur son environnement, il l'influence, il y applique une *force* (il *pousse* la porte) ; c'est alors l'environnement qui fait la somme des influences qui sont appliquées à chaque objet et qui détermine ainsi son nouvel état[1] (ou en tout cas le changement d'état).

Les agents savent plus facilement si leur action a réussi dans la mesure qu'ils attendaient. Ainsi, dans notre exemple de la porte, si nos agents appliquent des poussées d'égales intensités sur la porte, les deux forces vont s'annuler et chacun de nos agents s'apercevra que sa poussée était sans effet, il pourra donc conclure

4. Bien sûr, cela n'est un problème que dans un espace simulant un monde *réaliste,* qui suit des lois et des contraintes de cohérence.
5. Pour preuve : remplacez la porte par un œuf, il ne casserait jamais !

que sa poussée était trop faible dans ce cas là et donc que soit quelqu'un ou quelque chose bloque la porte, soit la porte est plus lourde que prévu.

De plus, transmettre des influences au lieu d'appliquer des actions permet d'intégrer facilement des forces de frottement, de l'inertie ou des influences globales : bruit dans les communications, champs de force (gravité, magnétisme, etc.).

Une autre conséquence encore plus importante est que ce motif fait clairement apparaître une distinction entre l'espace et l'environnement. En effet, l'environnement est uniquement composé d'objets situés alors que l'*espace* est une topologie de l'espace des coordonnées de ces objets situés.

En outre, le fait que le terme d'influence suggère un *stimulus* externe sans en avoir la connotation « réactif » permet d'englober sous un terme unique, et donc sous un mécanisme unique, l'ensemble des stimulations extérieures possibles auxquelles les agents peuvent être soumis, des forces physiques (accélération, attraction et répulsion) aux actes de langage, en passant par les différentes perceptions.

8. Motifs associés

Le motif *Marques* (page 93) peut profiter de ce motif pour la diffusion des marques dans l'environnement et pour leur perception (une *marque* est une source d'attraction/répulsion).

En ce qui concerne la mise en œuvre, les différents motifs objets suivants sont d'une aide indubitable :

Discrétisation (page 143) peut aider à régler les éventuels problèmes dus à la discrétisation de l'espace lors de l'application de la technique du gradient à vagues.

Entité-Physique (page 147) peut aussi intervenir : l'agent ne doit pas traiter les influences physiques qu'il subit (*e. g.*, s'il est au dessus d'un précipice, il doit tomber, même s'il n'a pas encore regardé en bas : seuls Will Coyote ou Grosminet peuvent ne pas tomber dans ces cas là), le corps de l'agent doit donc être considéré comme un objet physique.

Command [Gamma *et al.* 94, p. 233] qui encapsule les requêtes (Influence et Modification) dans un objet, permettant ainsi le paramétrage des clients avec différentes requêtes, la mise en tampon ou la mémorisation des requêtes et le support des opérations réversibles.

Composite [Gamma *et al.* 94, p. 86] qui permet de gérer un composé comme un composant (Influence et Modification).

⋆⋆⋆

Motifs architecturaux

Ces motifs introduisent des propositions pour l'architecture des agents. Une architecture d'agent décrit les flux d'information et de commandes qui circulent entre différents modules pour permettre à l'agent de choisir à chaque instant la meilleure action à faire compte tenu de la tâche qu'il doit accomplir et de la situation dans laquelle il se trouve.

Nous présentons ici les bases principales de cette série de motifs. Il pourrait être possible d'écrire un motif par architecture d'agent, mais, étant donné le nombre important — et toujours croissant — d'architectures et de modèles internes d'agents, il nous a semblé judicieux de présenter des motifs pour des familles d'architectures. Nous laissons donc à chaque concepteur la charge de décrire sa propre architecture — si elle a été appliquée avec succès à plusieurs reprises et si elle apporte des concepts différents de celles déjà présentées ou s'il s'agit d'une mise en œuvre particulièrement efficace.

Nous proposons quatre motifs de ce type :

Architecture BDI : ce type d'architecture est la mise en œuvre du modèle BDI *(Belief Desire Intention)*.

Architecture verticale : architecture hybride (l'agent est à la fois réactif et délibératif) dont les modules sont agencés en séquence et dans laquelle chaque module est associé à sa propre base de connaissances.

Architecture horizontale : autre architecture hybride mais dont les modules sont agencés en parallèle, suivant un modèle subsomptif.

Architecture récursive : architecture au contrôle décentralisé, les différents modules sont des agents, ce qui fait de l'agent un système multi-agent.

Motif

Architecture BDI

1. Synopsis

Ce type d'architecture permet de mettre en œuvre le modèle BDI *(Belief, Desire, Intention).*

2. Forces

– Les agents que l'on utilise fonctionneront sur le modèle BDI.

– Le comportement des agents est bien plus cognitif que réactif.

– La rationalité de l'agent est plus importante que sa rapidité.

– Les agents doivent avoir de nombreuses connaissances et doivent être capables de les manipuler.

– Les connaissances manipulées sont relativement abstraites.

– Les agents poursuivent plusieurs buts parallèles.

– Les agents ont des plans pour réaliser leurs buts.

3. Usages

Bien sûr, le premier exemple d'architecture BDI a été proposé par les créateurs du modèle BDI, Michael Georgeff, A. Lansky & Anand Rao. Cette architecture, utilisée dans le système PRS[1], [Lansky & Georgeff 87, Rao & Georgeff 91, Singh *et al.* 99], repose sur la structure présentée par la figure 23 page suivante. Quatre modules contiennent chacun respectivement les croyances *(Beliefs),* les désirs, les intentions et les plans (préconstruits) de l'agent. Ces connaissances sont stockées sous forme de règles et de faits. Le fonctionnement de ces différents modules (activation des règles, modification des bases) est géré par un module central, l'interprète.

Dans [d'Inverno *et al.* 97], Mark d'Inverno, David Kinny, Michael Luck & Michael Wooldridge proposent une spécification formelle de l'architecture

1. *Practical Reasoning System.*

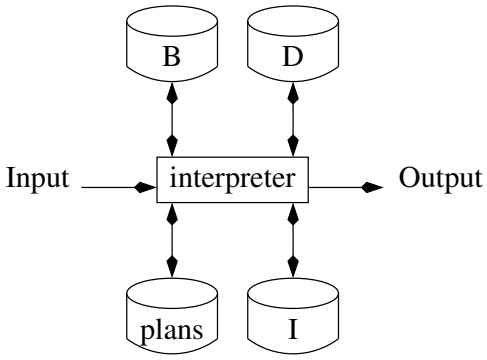


FIG. 23 – L'architecture BDI de PRS

de dMARS[2]. dMARS est une mise en œuvre clairement spécifiée de PRS. En effet, bien que largement utilisé, PRS manquait de clarté quant à son fonctionnement interne. La structure de dMARS est donc la même que celle de PRS.

Jack Intelligent Agents™, [HODGSON *et al.* 99, HOWDEN *et al.* 01], utilise la même architecture que PRS pour ses agents BDI. Jack se pose en successeur de PRS et dMARS et utilise donc la même structure.

Dans Zeus, la plate-forme multi-agent de *British Telecom,* l'architecture proposée pour la structure interne des agents est une architecture BDI, héritière de celle de dMARS. Zeus apporte une modification à dMARS en différenciant l'interface au monde concernant les perceptions et les actions de l'interface au monde concernant les messages (cf. figure 24). Elle sépare aussi l'interprète en différents modules : gestion des messages, planification, exécution et, le principal, le moteur de coordination qui contrôle le tout.

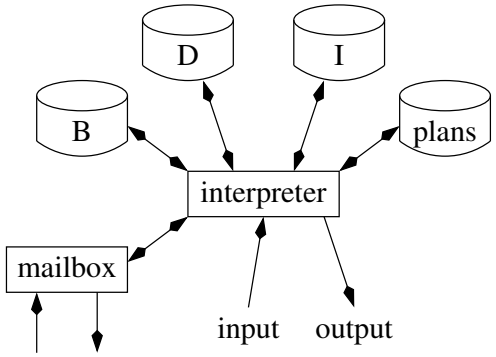


FIG. 24 – L'architecture BDI de Zeus

4. Solution

L'architecture proposée par ce motif est l'utilisation d'un module principal, l'interprète, qui fera le lien entre le modèle interne de l'agent et l'environnement de l'agent. Ce module sera connecté à quatre bases de connaissances.

La première de ces bases de connaissances contient les croyances de l'agent. Ces croyances sont des faits sur le monde et les règles de transformation de ces

2. *Distributed MultiAgent Reasoning System.*

faits dans le monde. C'est l'interprète qui intégrera les perceptions de l'agent dans ce module de croyances.

La deuxième de ces bases contient les désirs de l'agent. Ces désirs représentent les buts à plus ou moins long terme de l'agent. Ces désirs doivent être cohérents (*i. e.*, ne pas s'opposer les uns aux autres). Le module interprète manipulera ces désirs, notamment pour créer les intentions de l'agent.

Ces intentions forment la troisième base de l'agent. Une intention est une action (ou un ensemble d'actions) à réaliser. L'interprète définit cette intention à partir des désirs et des plans préconstruits que l'agent possède. Une intention est une instanciation d'un plan ou d'une partie d'un plan en fonction de l'état du monde et des ressources disponibles.

Enfin, la dernière base de l'agent contient une bibliothèque de plans préétablis qui lui permettront de choisir les actions à effectuer et donc les intentions à instancier.

L'architecture BDI générique résultante est donc celle qui est présentée sur la figure 25.

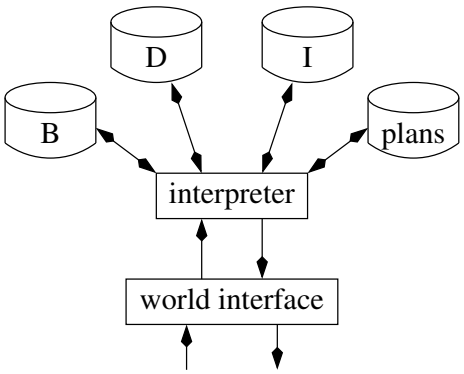


Fig. 25 – Architecture BDI générique

5. Mise en œuvre

Dans l'article [d'Inverno *et al.* 97], est proposée une spécification très claire, en *Object-Z*, de la mise en application de l'architecture BDI.

M. Mulder, Jan Treur & Michael Fischer, [Mulder *et al.* 97], ont mis en œuvre deux versions de l'interprète BDI, l'une en *Concurrent MetateM* et l'autre grâce à desire (cf. le chapitre II page 33).

Les croyances de l'agent sont représentables sous la forme d'une base de faits et de règles de type Prolog (*i. e.*, des faits et des prédicats logiques). La plus grande difficulté est qu'il est souvent nécessaire d'utiliser une logique temporelle et modale, incluant divers degrés de croyance.

Les désirs de l'agent sont représentables sous la forme de buts, préfixés de prédicats modaux tels que « vouloir que » ou « chercher à savoir si ».

Les plans sont des ensembles ordonnés d'actions. Ils font aussi entrer en jeu des conditions d'application et des conditions de terminaison.

Les intentions de l'agent sont des tâches que l'agent doit réaliser.

L'interprète choisit les plans à instancier et les croyances à modifier en fonction de ses buts, des perceptions qu'il reçoit et des croyances de l'agent.

6. Discussion

L'utilisation de cette architecture BDI a le principal avantage de mettre en adéquation l'architecture interne de l'agent et son modèle conceptuel, si ce dernier est calqué sur le modèle BDI ou un de ses dérivés (comme [Parsons & Giorgini 99, Vercouter *et al.* 00, Ma & Shi 00]).

Le modèle BDI a de nombreux avantages. C'est un modèle de délibération très utilisé et qui a inspiré de nombreux travaux. Son but est de modéliser le comportement décisionnel d'un agent rationnel, il n'est pas applicable à des agents devant avoir des réactions rapides.

Le principal problème de cette architecture est qu'elle fait porter tout le poids de la décision sur le module interprète qui doit gérer les différentes bases et leurs relations. La réalisation de ce module est donc le point le plus délicat de ce type d'architecture.

D'autre part, le formalisme apporté par l'utilisation d'une logique modale et temporelle permet d'accéder aux avantages de la formalisation (rigueur, vérifiabilité, automatisation, etc.) mais la manipulation d'un tel formalisme ne permet vraiment de gagner que si le problème abordé a une taille suffisante. En l'occurrence, les agents que l'on veut utiliser doivent avoir réellement besoin d'utiliser le modèle BDI, c'est-à-dire qu'ils doivent avoir des tâches complexes à réaliser et les décisions qu'ils ont à prendre doivent être conséquentes.

7. Motifs associés

Les autres types d'architecture interne d'un agent peuvent pallier certains des problèmes inhérents à cette architecture : *Architecture verticale* (page 117), *Architecture horizontale* (page 125) ou *Architecture récursive* (page 131).

⋆⋆⋆

Motif

Architecture verticale

1. Synopsis

Ce type d'architecture modulaire propose d'étager les modules suivant des niveaux de connaissance, chaque module sera chargé d'opérer à un niveau de connaissance précis.

2. Forces

– Les agents doivent avoir un comportement temps réel (*i. e.*, ils doivent répondre en un temps limité et connu).

– La situation requiert une réaction rapide de la part des agents.

– Toutes les situations ne peuvent être abordées par un comportement uniquement réactif : l'agent doit avoir plusieurs niveaux de cognition.

– Les agents agissent suivant des plans qu'ils doivent construire ou reconnaître en temps réel.

3. Usages

Dans son mémoire de thèse, [Firby 89], Robert James Firby développe une architecture verticale en trois niveaux utilisant les RAP *(Reactive Action Packages).* Un RAP est un ensemble de méthodes alternatives qui permettent d'accomplir une même tâche. L'architecture se décompose en trois couches (cf. figure 26 page suivante) : le planificateur, le système d'exécution et le contrôleur. Le contrôleur envoie des percepts au système d'exécution qui lui répond par des actions primitives. Des plans abstraits, c'est-à-dire un ensemble de tâches à effectuer représentées par des RAP — donc pour lesquelles la méthode de réalisation n'est pas encore désignée —, sont transmis du planificateur au système d'exécution. C'est ce dernier qui essaiera les différentes méthodes que les

RAP proposent : à l'instant de l'exécution, le contexte actuel est comparé au contexte type qui est associé à chaque méthode.

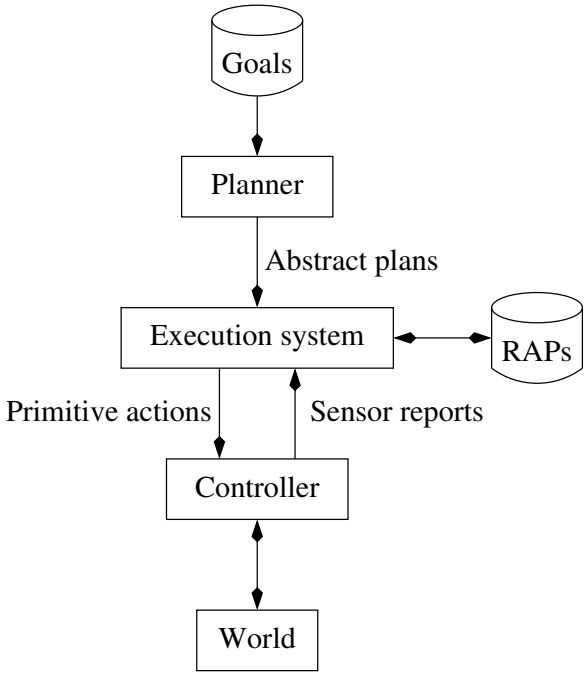


FIG. 26 – L'architecture verticale de FIRBY

Erann GAT a développé atlantis, [GAT 92], sur la même architecture que FIRBY. Le système d'exécution a été renommé en séquenceur et le planificateur en système délibératif *(deliberator)*. Le rôle du séquenceur est donc clarifié et ouvert (l'on peut plus facilement abstraire sa fonction et donc la modifier), de même pour le module délibératif dont l'action planificatrice est passée en second plan derrière son rôle de module délibératif : la planification est alors vue comme une forme particulière de délibération.

En 1992, Jonathan H. CONNELL décrit l'architecture verticale à trois couches SSS *(servo, subsumption, symbolic)*, [CONNELL 92]. La couche *servo* s'occupe des capteurs et des effecteurs, en manipulant des données dont les valeurs possibles sont continues (l'espace et le temps sont continus). La couche *subsumption* s'occupe des comportements par reconnaissance de situation : l'espace est discrétisé et le module essaie de reconnaître des motifs situation–action. Chacun de ces motifs est associé à un comportement réactif pour lequel chaque situation entraîne une modification des effecteurs (donc du niveau inférieur). Le troisième module, *symbolic,* manipule un espace-temps discret et tente de reconnaître des évènements discrets qui lui permettront de modifier le comportement de la couche inférieure (en inhibant ou en renforçant des comportements, avec des règles comme « après E faire A » ou « faire A jusqu'à E »). Cet exemple est intéressant car il intègre une discrétisation progressive de l'espace et du temps à mesure que l'on monte dans le niveau de cognition. Le réactif demande le continu (rapidité) et le cognitif le discret (abstraction).

En 1995, Robert James FIRBY reprend son architecture à base de RAP dans animate, [FIRBY *et al.* 95], sans y apporter de grandes modifications.

Dans [BONASSO *et al.* 95, BONASSO *et al.* 97], R. Peter BONASSO, Robert

J. Firby, Erann Gat, David Kortenkamp, David P. Miller et Marc G. Slack ont développé ${}_3T$, une architecture fondée sur atlantis et sur celle de Firby. La différence majeure est l'exécution concurrente des trois niveaux et leur asynchronicité, alors que dans les architectures précédentes, le séquenceur (ou système d'exécution) détenait le contrôle de l'exécution. Un autre apport de ${}_3T$ est le refonte de la couche réactive (contrôleur) en un gestionnaire de compétences *(Skill Manager)*. Ce gestionnaire gère aussi des événements : ce sont des drapeaux qui sont modifiés par les capteurs et par les compétences (*e. g.* « obstacle à droite » ou « impossible de rouler »).

Jörg P. Müller, [Müller & Pischel 93, Müller 96], a développé une architecture verticale appelée interrap et fondée sur le modèle cognitif de ratman, [Bürckert & Müller 91]. Ratman propose de découper les connaissances de l'agent en niveaux de connaissances, chaque niveau étant composé de méta-connaissances sur le niveau directement inférieur — le préfixe « méta » signifie ici qu'elles portent sur les connaissances du niveau inférieur. L'on peut rapprocher cette approche de la théorie des niveaux d'apprentissage de G. Bateson, [Bateson 84]. Chaque niveau d'interrap est donc associé avec une base de connaissances (cf. fig. 27).

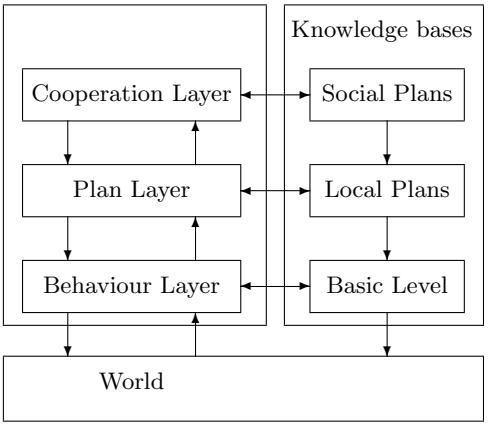


Fig. 27 – L'architecture verticale interrap couplée au modèle ratman

Christoph G. Jung propose — d'abord dans [Jung & Fischer 98], puis dans son mémoire de thèse : [Jung 99] — de paralléliser les modules d'interrap pour permettre une cognition plus rapide. L'architecture modifiée prend le nom d'interrap-r.

4. Solution

L'architecture générique qui ressort des différents exemples suit le schéma de la figure 28 page suivante : les modules sont placés en niveaux, le flux des perceptions arrive dans l'interface au monde et remonte les niveaux ; le flux des actions descend les niveaux jusqu'aux effecteurs, dans l'interface au monde. À chaque niveau est associée une base de connaissances ; en général, chacune comporte des méta-connaissances sur celle du niveau directement inférieur.

Les exemples précédemment présentés proposent, en général, un découpage en trois niveaux : réactif pour le premier module, contrôle pour le deuxième et

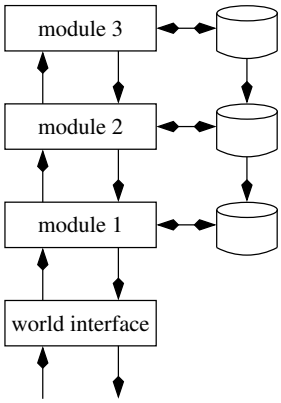


FIG. 28 – Architecture verticale

planification pour le troisième module. Il est envisageable de n'avoir que deux modules (cas limite inférieur) ou bien d'augmenter leur nombre.

Ces trois niveaux sont accompagnés d'un pseudo-niveau — *pseudo* car il n'est pas relié à des connaissances. Il est parfois confondu dans le module 1, celui qui gère le niveau réactif. Nous l'appellerons niveau zéro. Il fait l'interface avec le monde *(world interface)*. Il contient les capteurs et les effecteurs, en relation directe avec le monde.

Le premier module gère le niveau réactif. Il correspond au *Basic Behaviour Level* d'interrap-r, qui gère la base de connaissances sur les comportements *(Behaviour Layer)*. C'est le plus sollicité, il s'occupe des comportements réflexes de l'agent.

Le deuxième module gère le niveau local. C'est le *Local Planification Level* d'interrap-r, sa base de connaissances est un ensemble de plans *(Plan Layer)*. Il s'occupe d'appliquer un plan répondant à la situation actuelle et à la situation désirée à courte ou moyenne échéance. Il peut aussi décider d'appliquer ou de construire des plans simples.

Le troisième module gère le niveau social. C'est le *Social Planification Level* d'interrap-r, il fait entrer en scène des connaissances sur les comportements sociaux *(Cooperation Layer)*. C'est à ce niveau que les plans sont construits ou en tout cas décidés. Les plans qu'il construit, ou applique, ont une vision plus lointaine dans le futur, ils peuvent aussi faire cas des comportements prévisibles des autres agents.

En ce qui concerne l'architecture — la façon dont sont agencés les modules —, le principe fondateur est la séquentialité des modules. C'est le module de niveau inférieur qui appelle le module de niveau directement supérieur, en général lorsqu'il ne peut résoudre lui-même le problème que lui pose la situation en cours. La séquence de mise en application des modules peut être totale : tous les modules sont appelés, ou non : dès qu'un module trouve l'action ou le plan adéquat, la séquence est interrompue.

De même, la séquence peut être « à une passe » ou « à deux passes ». Pour la première, présentée figure 29 page ci-contre, le module qui arrête la séquence (*i. e.*, celui qui a trouvé l'action ou le plan à appliquer) est celui qui demande l'application de l'action suivante. Pour la seconde, figure 28, l'action ou le plan redescend les modules un par un jusqu'aux effecteurs. La version « à une passe » peut être rapprochée de l'architecture en tube du motif *Pipe*, où des filtres suc-

cessifs modifient les données d'entrée pour en faire les données résultat. La version « à deux passes » se rapproche quant à elle de l'architecture OSI[1] qui découpe le traitement de l'information en couches, chacune d'un niveau logique particulier (les sept couches : physique, données, réseau, transport, session, présentation, application). Dans la phase de montée du modèle OSI, chaque couche ne traite que les informations qui la concernent et fait remonter les autres données aux couches supérieures si aucune action n'est à effectuer à ce niveau (sinon la phase de descente est enclenchée[2]). Dans la phase de descente, les données provenant de la couche supérieure sont emballées et transmises à la couche inférieure. Les couches inférieures sont un passage obligé, les couches supérieures peuvent être évitées.

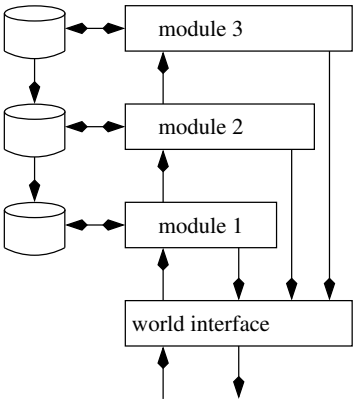


FIG. 29 – Architecture verticale à une passe

Les dernières modifications apportées à ces architectures introduisent un parallélisme, un fonctionnement concurrent des niveaux supérieurs, pour une plus grande rapidité de ces niveaux et donc un comportement plus réfléchi de la part de l'agent (nous verrons dans le champ *Discussion* que les agents verticaux ont tendance à être trop réactifs).

5. Mise en œuvre

Les travaux cités en exemples fournissent chacun une mise en œuvre d'une architecture verticale. Nous avons donc trois modèles de mise en œuvre : celui de FIRBY et d'atlantis, celui de ${}_3T$ et celui d'interrap et d'interrap-r.

La mise en œuvre d'atlantis repose sur le séquenceur (ou système d'exécution, cf. fig. 26 page 118) et sur les RAP.

${}_3T$ propose un modèle plus complexe, sa mise en œuvre repose sur un modèle événementiel : les capteurs et les effecteurs manipulent des drapeaux pour que le séquenceur puisse décider du contexte d'application des méthodes. Ce contexte se décompose en deux sous-contextes : le contexte externe reposant sur l'environnement de l'agent et le contexte interne relevant des possibilités des « compétences » (*i. e.* les actions possibles des effecteurs). Le choix de la

1. *Open Systems Interconnection* (norme ISO 9646-1).
2. En général, cela arrive lorsqu'une erreur se produit. Cela peut aussi arriver lorsque la couche possède toutes les données nécessaires à la réponse.

méthode à appliquer pour réaliser l'action en cours (décidée par le plan en cours, modifié par le plannificateur) se fait parmi celles proposées par le RAP suivant ce contexte.

Les architectures interrap et interrap-r demandent une conception avancée des bases de connaissances associées aux modules.

6. Discussion

Les architectures verticales ont été conçues pour avoir un fonctionnement hybride, alliant la rapidité d'un comportement réactif à la clairvoyance d'une planification à moyen terme ou à long terme. Donc, d'un point de vue théorique, l'idée semble bonne.

D'autre part, le découpage de la cognition en niveaux de connaissance permet, comme tout découpage, de diviser le travail pour le concepteur et pour l'agent.

Toutefois, plusieurs problèmes surgissent. Le premier problème qui se pose est la difficulté de déterminer les connaissances que chaque niveau doit manipuler. D'une part, plusieurs tâches sont réalisées par chaque module et donc dans chaque niveau se confondent les connaissances liées aux tâches du module correspondant. Et d'autre part, les connaissances liées à une même tâche sont disséminées dans les différentes bases.

De plus, chaque niveau est un niveau méta du précédent. Cela implique une abstraction de plus en plus grande et donc de plus en plus délicate à manipuler.

De la même façon, chaque module doit donner ses connaissances et ses résultats au module supérieur. Ceci implique que le flux entre deux modules est particulier aux modules. Le flux arrivant aux capteurs est un flux de percepts. Celui qui arrive au module réactif représente la situation en termes de percepts. Celui qui arrive au module local représente la situation en termes de *stimuli* et de réactions. Celui qui arrive au module social représente la situation en termes de situations successives et de plans.

Dans l'autre sens (*i. e.*, dans la redescente du flux), il s'agit de plans, puis d'actions, et, enfin, d'ordres aux effecteurs.

Chaque module supplémentaire demande deux nouveaux protocoles, une nouvelle base de connaissances contenant un nouveau niveau de connaissance, niveau méta du précédent.

Pour pallier ce problème posé par le modèle ratman, dans [Fischer *et al.* 95], Klaus Fischer, Jörg P. Müller & Markus Pischel ont proposé une technique d'unification du contrôle et de l'architecture de chaque module dans interrap. Une structure unifiée des modules permet de les concevoir plus facilement — sur le même modèle. De la même façon, l'unification du contrôle permet de ne pas avoir à repenser les flux passant d'un module à un autre.

Le dernier problème posé par ce type d'architecture vient de ses propriétés temps réel. En effet, la séquentialité des modules et le fait que les modules de haut niveau sont forcément plus lents — car ils manipulent des informations de niveau plus élevé, donc de plus grosse granularité, surtout temporelle[3] —, font que, dans la plupart des cas concrets, seul le niveau réactif est apte à répondre

3. Il faut plus de temps pour les collecter, par agrégation des informations de niveau plus faible.

dans les temps ; les niveaux plus élevés soit n'ont pas le temps de répondre, soit répondent de façon inadéquate, la situation ayant évolué entre temps.

Les dernières moutures des architectures verticales ont parallélisé l'exécution des modules, au moins pour les niveaux supérieurs : l'architecture interrap-r, [Jung & Fischer 98, Jung 99], ou l'architecture ${}_3T$, [Bonasso *et al.* 95, Bonasso *et al.* 97]. Il faut bien avouer qu'interrap-r ne règle pas tous les problèmes d'interrap, notamment la communication entre les modules et les problèmes posés par le parallélisme (cf. le motif suivant *Architecture Horizontale*). Par contre ${}_3T$ semble une véritable amélioration bien que la parallélisation soit limitée par le rôle prépondérant du séquenceur : c'est toujours lui qui contrôle l'agent, la parallélisation revient en fait à rendre asynchrone la communication entre les modules mais pas à les rendre réellement autonomes.

7. Motifs associés

Les autres types d'architecture interne d'un agent peuvent pallier certains des problèmes inhérents à cette architecture : *Architecture horizontale* (page 125), *Architecture* BDI (page 113) ou *Architecture récursive* (page 131).

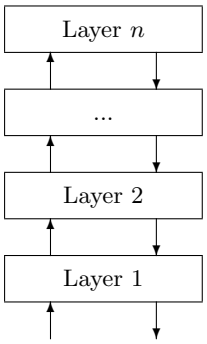


Fig. 30 – Motif *Layers*

Ce motif peut être rapproché du motif *Layers*, plus générique, représenté figure 30, présenté dans [Buschmann *et al.* 95]. L'architecture OSI en est l'exemple typique. Ici, chaque module correspond à une couche *(layer)*.

La version « à une passe » peut être rapprochée de l'architecture en tube du motif *Pipe*.

⋆⋆⋆

Motif

Architecture horizontale

1. Synopsis

Ce type d'architecture modulaire propose de découper le comportement de l'agent en tâches et de consacrer un module à chacune.

2. Forces

– Les agents doivent avoir un comportement temps réel (*i. e.*, ils doivent répondre en un temps limité et connu).

– La situation requiert une réaction rapide de la part des agents.

– Toutes les situations ne peuvent être abordées par un comportement uniquement réactif : l'agent doit avoir plusieurs niveaux de cognition.

– Le comportement de l'agent consiste à effectuer plusieurs tâches en parallèle.

– L'agent doit être résistant à l'échec d'une tâche.

3. Usages

En 1985, Rodney A. Brooks introduisit l'idée de décomposer le système de contrôle d'un robot mobile, non plus en modules fonctionnels sur l'architecture en tube (le motif *Pipe*, ou, plus particulièrement pour les architectures agent, la version à une passe du motif *Architecture verticale* qui précède celui-ci), mais suivant des comportements associés chacun à une tâche particulière, [Brooks 86]. [1]

1. Attention, Brooks qualifie d'horizontales les architectures verticales du motif *Architecture verticale* (qui suit la littérature sur les architectures hybrides), tout simplement du fait qu'il les dessine horizontalement, ne reconnaissant que les architectures à une passe, donc en forme de tube. Il qualifie d'architectures de subsomption ce que l'on appelle ici les architectures horizontales.

L'architecture proposée fonctionne grâce à la subsomption (fig. 31) :

– chaque tâche est remplie par un module ;

– tous les modules reçoivent les informations des capteurs en même temps ;

– les tâches sont ordonnées suivant un niveau de capacité ;

– les tâches de niveau supérieur peuvent prendre la place des niveaux inférieur (subsomption) ;

– les tâches sont relativement simples : « éviter les obstacles » (niveau zéro), « explorer » (niveau deux) ou « identifier les objets » (niveau cinq).

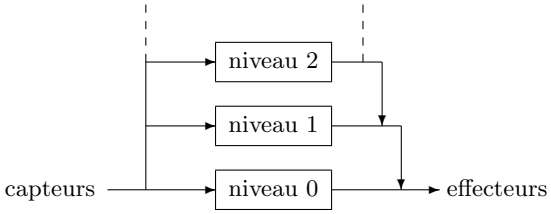


FIG. 31 – Architecture horizontale de BROOKS

La figure 32 (tirée de [FERGUSON 91, fig. 1 p. 253]) représente l'architecture *TouringMachines*, proposée par Innes A. FERGUSON dans sa thèse, [FERGUSON 92]. Les modules sont aussi parallèles, mais, cette fois-ci, ils ne sont pas dédiés à des tâches simples (ou uniques) mais à des tâches groupées par niveau de connaissances (un peu comme dans le modèle ratman et dans l'architecture interrap, cf. le motif précédent, *Architecture verticale*).

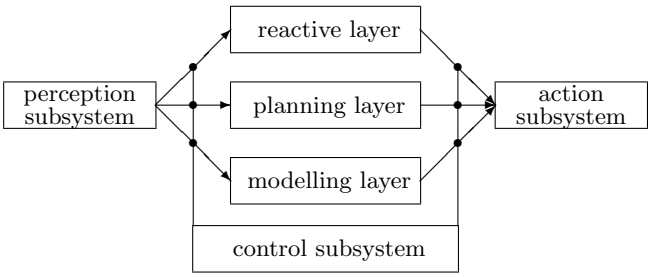


FIG. 32 – Architecture horizontale de *TouringMachines*

Aaron SLOMAN & Riccardo POLI utilisent une architecture horizontale pour le contrôle d'un robot mobile, [SLOMAN & POLI 95]. Cette architecture ressemble à celle de FERGUSON : elle regroupe les différentes tâches suivant leur niveau cognitif : réflexes, automatismes (innés et acquis), délibérations et introspections. Une frontière est déterminée entre les tâches réactives (réflexes et automatismes) et les autres tâches : l'*attention* de l'agent n'est nécessaire que pour les tâches de haut niveau. Cette architecture est implantée dans la bibliothèque de classes sim_agent.

4. Solution

L'architecture générique qui ressort des différents exemples suit le schéma de la figure 33 : les modules fonctionnent en parallèle, le flux des perceptions arrive dans tous les niveaux ; le flux des actions part de tous les niveaux et arrive jusqu'aux effecteurs. Un module de contrôle est nécessaire pour gérer, en entrée, la distribution des percepts et, en sortie, le choix des actions (en particulier par subsomption des modules de haut niveau sur les modules de bas niveau).

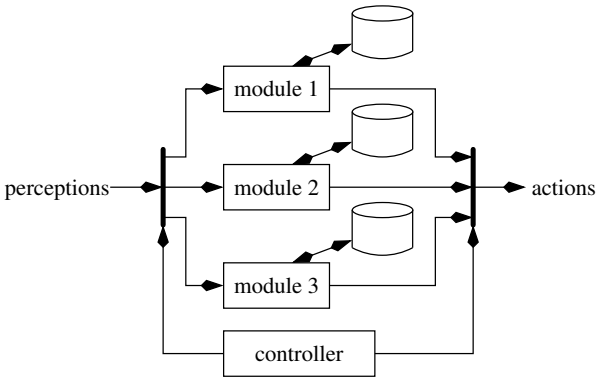


Fig. 33 – Architecture horizontale

Chaque module est associé à une tâche spécifique. L'on peut traiter des différentes tâches liées au comportement de l'agent (*e. g.* une tâche par rôle), ou bien regrouper les tâches de l'agent suivant des niveaux de connaissance (un peu comme dans le motif précédent : *Architecture verticale*) ; le découpage peut alors se faire en trois niveaux : réactif, planificateur et modélisateur.

Le niveau réactif *(Reactive Layer)* s'occupe des comportements réflexes de l'agent.

Le niveau planificateur *(Planning Layer)* s'occupe de construire, ou d'appliquer, un plan répondant à la situation actuelle et à la situation désirée à courte ou moyenne échéance.

Le niveau modélisateur *(Modelling Layer)* maintient le modèle du monde de l'agent, le modèle que l'agent se fait des autres agents ; de plus, en cas de conflit, il modifie les plans engendrés par le module plannificateur.

Un quatrième module est nécessaire dans cette architecture. Il s'agit du module de contrôle qui empêche les autres modules d'entrer en conflit lors de l'envoi de leurs ordres au module d'action (les effecteurs). Il doit savoir juger quel module a raison si les actions que les modules proposent entrent en conflit.

En ce qui concerne l'agencement des modules dans ce type d'architecture, le principe fondateur est le parallélisme des modules. Tous les modules travaillent en même temps sur les informations issues des capteurs.

5. Mise en œuvre

Les exemples cités plus haut décrivent chacun une implantation d'une architecture horizontale.

Les points principaux sont les suivants :

– les percepts sont donnés à tous les modules ;

– chaque module est autonome et travaille sur les percepts pour fournir une action à effectuer ;

– les actions fournies par les modules sont examinées par le module de contrôle pour :

1. savoir quelle(s) action(s) effectuer ;
2. éviter les conflits.

Le module de contrôle est le module le plus délicat à concevoir. La solution la plus simple est la subsomption : le module modélisateur a le pas sur les autres modules et le module plannificateur a lui-même le pas sur le module réactif. Cela pose l'hypothèse que les modules de haut niveau ont plus souvent raison que ceux de plus bas niveau.

Cette hypothèse n'est pas toujours vérifiée, notamment du fait que plus le niveau est haut, plus le calcul est long et les informations manipulées importantes : les modules de haut niveau peuvent ne pas avoir traité toutes les informations et, donc, peuvent proposer des actions d'un plan établi plus tôt et non encore mis à jour, donc potentiellement obsolète.

D'autre part, si l'on n'utilise pas de technique spéciale ou d'algorithme *anytime* — grâce auxquels les modules peuvent donner une réponse rapide qu'ils affinent au fur et à mesure —, c'est toujours le module réactif, le plus rapide, qui traite ses informations le premier et qui, de ce fait, donne le premier une réponse. L'action proposée par le module réactif sera donc exécutée puisque les autres modules n'auront encore rien proposé ; à moins qu'ils n'aient donné des indications au module de contrôle pour bloquer cette action, ce qui pose la question sur la nature de ces indications et le protocole utilisé pour les communiquer au module de contrôle.

6. Discussion

Ce type d'architecture a été conçu pour permettre un fonctionnement hybride de l'agent : à la fois réactif, pour la rapidité, et cognitif, pour l'adéquation du comportement à l'évolution de la situation.

La deuxième idée fondatrice est d'associer chaque module à une tâche particulière ou à un ensemble de tâches.

La troisième idée à la base des architectures horizontales est le parallélisme des modules. Celui-ci permet d'assurer une résistance aux échecs d'une tâche : si un module échoue, les autres peuvent continuer d'effectuer leur tâche et donc permettre à l'agent de poursuivre ses buts dans une certaine mesure.

D'un autre côté, le parallélisme des modules entraîne une charge de travail accrue : chaque module possède son fil d'exécution et demande donc plus de ressources matérielles.

De plus, comme l'on en a parlé dans le champ *Mise en œuvre,* la conception du module de contrôle et de son fonctionnement n'est pas aisée et demande une étude poussée des besoins et du fonctionnement désiré.

Un dernier problème est posé par cette architecture : les modules ne profitent pas des résultats des autres modules, or, de la façon dont sont en général découpés les modules (*i. e.*, par niveaux de connaissance), les modules de niveau

supérieur pourraient profiter des résultats des modules de niveau inférieur.

7. Motifs associés

Les autres types d'architecture interne d'un agent peuvent pallier certains des problèmes inhérents à cette architecture : *Architecture verticale* (page 117), *Architecture* BDI (page 113) ou *Architecture récursive* (page 131).

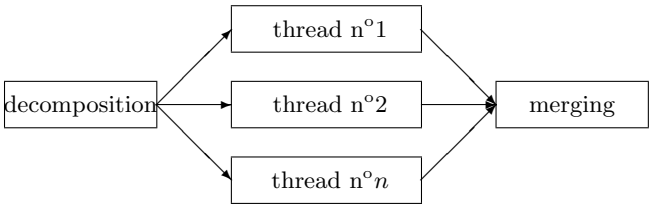


FIG. 34 – Motif *Master-slaves*

Ce motif peut être rapproché du motif *Master-slaves*, plus générique, présenté figure 34, décrit dans [MASSINGILL *et al.* 99]. Dans ce motif, une tâche est décomposée en plusieurs sous-tâches. Chaque sous-tâche est résolue dans son propre fil d'exécution. Les résultats doivent ensuite être fusionnés pour former le résultat de la tâche principale.

⋆⋆⋆

Motif

Architecture récursive

1. Synopsis

Ce type d'architecture modulaire propose d'agentifier les modules qui composent l'agent.

2. Forces

– La composante systémique du paradigme agent est primordiale.
– Différentes granularités dans les composantes du système sont nécessaires :

– Le système est composé de sous-systèmes, eux-mêmes composés de sous-systèmes, etc.
– Des agents forment des groupes qui doivent, eux aussi, être vus comme des agents.
– Un agent est composé de différents modules indépendants qui interagissent pour composer le comportement de l'agent.
– Les modules internes de l'agent sont relativement complexes.
– Le contrôle des modules internes d'un agent gagne à être décentralisé.

3. Usages

Dans toute la suite de ce motif, nous appellerons le système principal le macro-système et le système qui compose un agent sera appelé le micro-système.

Le modèle LRO3, développé par Serge Stinckwich lors de la première partie de sa thèse, [Stinckwich 94], propose une architecture d'objets concurrents récursive. Ces objets concurrents servent à mettre en œuvre des agents. LRO3 est mise en œuvre de deux façons distinctes : par le biais de processus

et par le biais d'acteurs. La récursivité de LRO3 est figée et la décomposition des agents malaisée.

Michel Occello & Yves Demazeau proposent, dans [Occello & Demazeau 97], une architecture d'agent récursive. Cette architecture, astro, permet aux agents de se décomposer (*i. e.*, leurs tâches et leurs connaissances) en d'autres agents récursifs lorsque le besoin survient. Ce besoin est caractérisé par une fonction de complexité (dont la valeur dépend de paramètres « définis par le concepteur » *(sic)*). Suivant sa fonction de complexité et suivant s'il a ou non atteint son but, un agent peut se décomposer ou se recomposer.

Kelly Fernandes & Michel Occello développent, dans [Fernandes & Occello 00], un SMA récursif qui leur permet d'organiser les micro-agents, en les regroupant dans des macro-agent. Cette organisation est effectuée suivant les contraintes données par d'autres macro-agents, fortement cognitifs, qui cherchent à atteindre des buts et qui sont en communication entre eux et avec l'utilisateur. Les micro-agents sont des agents purement réactifs. Les agents récursifs servent à représenter des groupes d'agents (récursifs ou non). Ils servent donc à simplifier la gestion des micro-agents. C'est l'architecture d'agent de [Occello & Demazeau 97] qui est utilisée pour les agents récursifs.

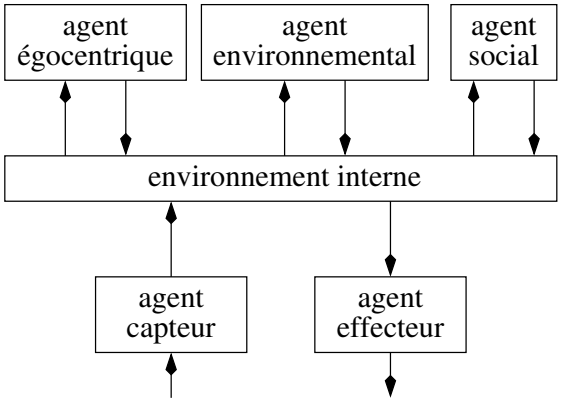


Fig. 35 – L'architecture apa

L'architecture apa[1], développée par François Girault lors de sa thèse, [Girault 02], est une architecture récursive qui a la particularité de n'utiliser que des agents situés (cf. fig. 35). En effet, les micro-agents sont situés dans un environnement interne au macro-agent et toutes les communications (en fait des influences, cf. le motif *Influences*) passent par cet environnement interne. Les capteurs et les effecteurs sont vus comme des micro-agents très élémentaires, ce qui permet de les manipuler comme des agents et, donc, de maintenir une cohérence dans les concepts et dans la gestion des communications. Apa a été tout d'abord développée sur le substrat de la Robocup, donc avec comme objectif le support de SMA à la fois réactifs et cognitifs. Par la suite, apa a été appliquée à l'analyse syntaxique, sous la mouture tapas[2], [Lebarbé & Girault 01]. Le modèle apa, et donc l'architecture apa, utilise aussi très fortement le motif *Marques*.

1. Anticipation par Perception Augmentée.
2. Traitement et Analyse par Perception Augmentée en Syntaxe.

4. Solution

La solution proposée par ce motif est de faire de l'agent un système multi-agent dans lequel les modules internes de l'agent seront eux-mêmes des agents, autonomes et téléologiques.

Il s'agit d'appliquer le principe de la systémique : un système est composé de sous-systèmes, eux-mêmes composés de sous-systèmes, etc.

L'architecture qui résulte de ce principe varie suivant la structure que l'on choisit pour l'architecture d'un système multi-agent. De plus, l'architecture du micro-système n'est pas forcément identique à celle du macro-système.

En effet, il existe différentes façons d'organiser un SMA. L'on peut avoir un SMA d'agents situés dans un espace et qui voient leurs interactions fortement contraintes par l'environnement. L'on peut avoir un SMA d'agents communiquant *via* un médium de communication partagé ne faisant intervenir aucune position, aucune métrique entre les agents (*i. e.*, dans lequel tous les agents sont égaux et « situés » au même « endroit ») — ce genre de médium s'apparente à un bus de communication. L'on peut encore imaginer n'importe quelle structure intermédiaire entre ces deux situations.

L'architecture générique que l'on peut toutefois faire émerger de toutes ces situations possibles ressemble à celle qui est représentée sur la figure 36. Les agents internes communiquent *via* l'environnement du micro-système. Cet environnement peut être constitué d'un espace et de contraintes de communications sur la situation des micro-agents. Cet environnement peut aussi se limiter à n'être qu'un simple bus de communication entre les micro-agents.

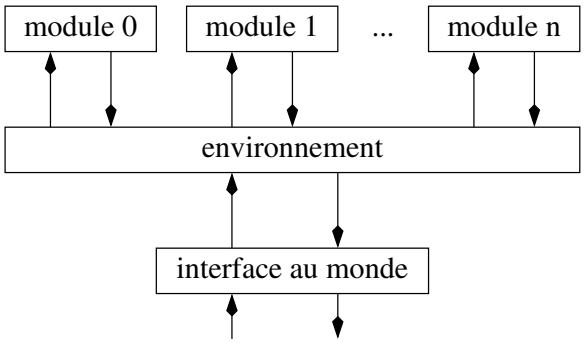


Fig. 36 – Architecture récursive générique

5. Mise en œuvre

La mise en œuvre de ce motif demande à l'évidence la présence d'agents minimaux, c'est-à-dire des agents qui ne sont pas eux-mêmes composés de micro-agents. L'on peut imaginer deux types d'agents minimaux : les *transducteurs* et les agents non récursifs.

Le premier type d'agent minimal, les transducteurs, correspond à de simples filtres. Ce ne sont pas véritablement des agents mais l'on peut les considérer comme des agents dégénérés pour traiter les agents minimaux comme des agents à part entière.

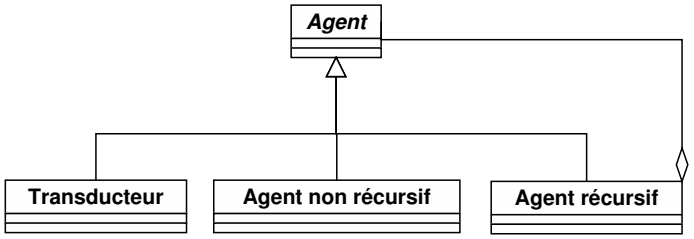


FIG. 37 – Les différents types d'agent

Le second type d'agent minimal englobe tous les types d'agents possibles. L'on peut utiliser tout modèle et toute architecture d'agent que l'on souhaite pour les réaliser. Pour donner un exemple, nous utiliserons une architecture très simple dans laquelle le module décisionnaire manipule des données qu'il place dans la mémoire de l'agent, ce qui actionne les effecteurs de celui-ci (cf. fig. 38).

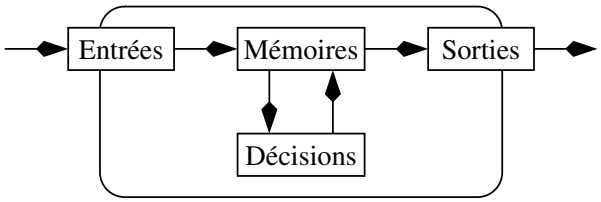


FIG. 38 – Architecture d'un agent non récursif

En partant d'un agent non-récursif, il suffit de remplacer le module décisionnaire de l'agent par un ou plusieurs agents (récursifs ou non) pour en faire un agent récursif (cf. fig. 39). Le module « mémoires » prend alors son intérêt puisqu'il devient le bus de communication utilisé par les différents micro-agents décisionnaires. L'on peut alors remarquer que les modules d'entrée et de sortie sont en fait des transducteurs qui, l'un, place les perceptions en mémoire et, l'autre, réagit aux actions demandées par les autres modules.

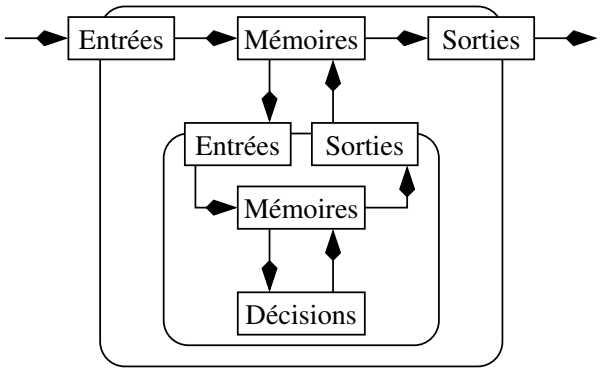


FIG. 39 – Architecture d'un agent récursif

6. Discussion

Le premier avantage apporté par l'application de ce motif est qu'il met en avant la composante systémique du paradigme agent. Cette composante est d'ailleurs l'une des plus importante des SMA car elle permet la récursivité agent–SMA (cf. l'introduction de ce mémoire, page 5). Ce motif facilite donc l'intégration de la réflexivité dans le modèle de l'agent et dans son architecture même.

Cette réflexivité permet aussi aux agents de simuler le comportement des autres agents pour extrapoler leurs réactions et donc mieux anticiper, mieux « comprendre » et prendre de meilleures décisions.

Dans un autre ordre d'idée, ce type d'architecture suit la lignée des systèmes multi-experts : un agent est composé d'experts autonomes qui coopèrent pour décider du comportement de l'agent.

En ce qui concerne la mise en œuvre, ce motif facilite la gestion du contrôle des modules internes de l'agent, en évitant les écueils des autres modèles d'architecture (manque de réactivité, de cognition, blocages et incohérences). En fait, ce motif propose de ne pas gérer ce contrôle puisqu'il propose d'utiliser un système multi-agent qui, par définition, décentralise le contrôle.

Par contre, en décentralisant le contrôle des modules internes, la récursivité ne fait que déplacer le problème du contrôle puisque celui-ci devra être traité *via* l'organisation de micro-système. Or, l'organisation d'un SMA et la collaboration et la coopération des agents au sein du SMA n'est pas forcément évidente.

De plus, la récursivité pose le problème des agents atomiques ou minimaux, ceux qui ne sont pas récursifs. Où doit s'arrêter la récursivité, comment concevoir ces agents minimaux et leurs interactions. Mais ce dernier point n'est pas si délicat : les agents minimaux, ainsi que leurs interactions, se doivent d'être simples.

7. Motifs associés

De toute évidence, le motif *Composite*, [Gamma *et al.* 94, p. 163] (cf. ce mémoire, page 21), sert à réaliser ce motif dans une programmation orientée objet. Les agents récursifs sont en effet des composites d'agents (cf. fig. 37 page précédente), les agents non récursifs étant les atomes du motif *Composite*.

Les autres motifs architecturaux de ce thésaurus — c'est-à-dire : *Architecture verticale* (page 117), *Architecture horizontale* (page 125) et *Architecture* BDI (page 113) —, peuvent servir pour définir l'architecture interne des micro-agents. Ils représentent aussi des alternatives à ce motif.

Le motif *Marques* (page 93) peut être utilisé pour la communication entre les micro-agents (comme c'est le cas dans apa).

Le motif *Schémas d'organisation* (page 79) peut bénificier de ce motif pour représenter des groupes d'agents prenant un rôle dans une organisation : le groupe est alors réifié par un agent récursif contenant les membres du groupe.

Tous les motifs de ce thésaurus peuvent en fait servir à réaliser ce motif. Celui-ci faisant intervenir des SMA dans sa réalisation, tous les motifs orientés agent sont susceptibles de lui être appliqués.

⋆⋆⋆

AntiMotifs

Ces motifs sont plus faciles à expliquer sous la forme d'AntiMotifs (cf. section 3.3.5 page 25). C'est-à-dire qu'ils expliquent certains symptômes d'un dysfonctionnement dû à une mauvaise technique de conception, de développement ou de mise en œuvre. Ils explicitent donc ces symptômes et donnent une solution de reconception–ré-ingénierie du système touché.

Iniquité : partage inéquitable des ressources, quelles soient logicielles ou bien matérielles, mémorielles ou computationnelles.

Discrétisation : perte partielle d'information, notamment sur des données de type numérique.

Entité Physique : les actions physiques sur l'agent sont retardées ou « oubliées ».

Motif

Iniquité

1. Synopsis

Lorsque le parallélisme est simulé trop simplement, les agents ne sont pas égaux en droit en ce qui concerne l'utilisation des ressources, notamment des ressources calculatoires.

2. Symptômes

Des conflits liés à la concurrence d'action apparaissent. Par exemple lorsque deux agents veulent utiliser la même ressource au même moment, au même pas d'exécution, deux cas de figure se présentent : soit le premier agent dans la liste utilise la ressource et l'autre agent ne pourra plus l'utiliser, et, dans ce cas, c'est toujours le même agent qui aura la priorité ; soit les deux agents utilisent la ressource et celle-ci aura été utilisée deux fois !

Nous avons donc deux principaux types de symptômes :

- iniquité de partage des ressources ;
- incohérence d'utilisation des ressources.

3. DysSolution

Ces symptômes surviennent souvent lorsque le parallélisme et les actions concurrentes sont simulées par un algorithme simple d'exécution séquentielle.

Les agents étant autonomes et distincts, ils doivent pouvoir exercer des actions concurremment les uns des autres. Si le nombre d'agents est trop important, ou si les ressources processeur sont trop faibles, chaque agent ne peut avoir son propre fil d'exécution *(thread).* Le parallélisme doit donc être simulé par un algorithme séquentiel.

Les comportements des agents sont découpés en actions élémentaires ou en séquences temporelles (*quanta* comportementaux) et, à chaque pas d'exécution, l'ensemble des actions élémentaires qui doivent être exécutées à ce pas

sont effectivement réalisées. La solution la plus simple est alors de les réaliser séquentiellement, sans autre ordre que celui des agents dans la liste de tous les agents utilisant ce fil d'exécution (on peut en effet utiliser plusieurs fils d'exécution tout en en ayant strictement moins que nécessaire pour que chaque agent ait le sien propre).

L'on peut aussi utiliser un ordre aléatoire sur les agents ou les actions. Dans ce cas, le système est exposé à une interférence entre le générateur aléatoire de l'agenda d'exécution et les différents générateurs aléatoires de la simulation. En effet, la conception de générateurs aléatoires suffisamment aléatoires, rapides et non corrélés est un problème connu.

4. Solution

Nous avons deux possibilités pour simuler le parallélisme sur une machine séquentielle. Soit l'on utilise une méthode asynchrone, où les agents ont accès au processeur chacun à leur tour. Soit l'on utilise une méthode synchrone, où les actions des agents qui interviennent entre un instant t et l'instant $t + 1$ sont mémorisées pour être appliquées toutes ensembles sur l'état du monde à l'instant t pour donner l'état du monde à l'instant $t + 1$. Que ce soit l'un ou l'autre terme de cette alternative qui est choisi, les conflits entre actions des agents sont toujours possibles.

Partant de là, plusieurs solutions s'offrent pour éliminer les symptômes de cet antimotif.

La première de ces solutions, la plus complexe, est d'utiliser un mécanisme de règlement automatique des conflits entre les actions des agents. C'est une solution très difficile car elle demande un modèle équitable de règlement de conflit. Il faut donc reconnaître les conflits lorsqu'ils se présentent, avoir un moyen de les régler, sans simplement les retarder, et s'assurer que ce règlement est équitable. De plus, cette solution a le désavantage de réintroduire un contrôleur central.

Une fausse solution est de prévenir les conflits en instaurant un système de réservation sur les ressources, voire même un système de négociation entre les agents. Ceci n'est pas envisageable pour tous les types de ressource. De plus, la détection d'une situation de conflit avant même son apparition est difficile.

Un algorithme de partage équitable du temps d'activité sur le processeur est une des meilleures solutions a envisager.

Une dernière solution — qui peut être appliquée en même temps que les précédentes — est l'utilisation du motif *Influences* (cf. page 105) qui permet de simplifier la gestion des conflits d'actions en séparant les actions de leurs effets.

5. Mise en œuvre

En ce qui concerne la mise en œuvre d'un algorithme d'exécution équitable qui peut permettre d'éviter les symptômes de cet antimotif, la programmation système, et notamment la programmation de systèmes d'exploitation multiprocessus ou *multithread*, est la meilleure piste à suivre.

Certaines plates-formes multi-agents, surtout les plates-formes orientées vers la simulation ou qui permettent ce genre d'application, intègrent les solutions proposées ici en proposant des agenda d'exécution plus ou moins équitables (MadKit, oRis, Cormas, etc.).

6. Discussion

Le soin apporté à l'équité de l'agenda d'exécution est un point important de tout système multi-agent, et en particulier lorsqu'il s'agit d'une simulation. L'utilisation d'un algorithme équitable permet de mieux s'approcher de la réalité.

Le principal problème est que, comme souvent, il faut faire un compromis entre l'adéquation du SMA avec le but visé et les moyens que l'on peut apporter à la réalisation de cette adéquation. Il faut par exemple faire attention à ne pas réintégrer un contrôleur global en incluant un mécanisme de gestion des conflits ; l'on perdrait alors l'un des intérêts des SMA qu'est la distribution du contrôle.

7. Motifs associés

Comme nous l'avons signalé dans le champ *Solution,* l'utilisation du motif *Influences* (cf. page 105) permet de gérer les conflits d'actions concurrentes.

L'antimotif *Discrétisation* peut collaborer avec celui-ci pour le choix de la durée du pas d'exécution.

⋆⋆⋆

Motif

Discrétisation

1. Synopsis

Comment choisir le bon niveau de discrétisation des données, quand est-ce nécessaire ?

2. Symptômes

- Une partie des états du système représentant des solutions est inaccessible.
- Les agents, le système complet, ont un comportement inapproprié.
- Des phénomènes percolatoires apparaissent.

3. DysSolution

La cause de ces problèmes vient en partie du fait que l'ordinateur est une machine discrète et que les phénomènes complexes que l'on veut représenter ont, eux, certaines caractéristiques de l'infini et du continu. Comme la machine ne peut gérer une infinité de valeurs possibles, il faut discrétiser les valeurs. Trois formes de discrétisation se présentent : approcher, réduire ou symboliser.

Approcher consiste à réduire le nombre de valeurs possibles en utilisant un représentant (une valeur particulière) pour un sous-ensemble de valeurs possibles. L'application exemplaire de ce principe est le codage des nombres réels par des nombres à virgule flottante. La précision ε de ces nombres fait qu'un nombre x représente en fait toute la population de nombres réels de $[x; x + \varepsilon[$ (ou $]x - \varepsilon; x]$ si $x < 0$).

Réduire consiste à décider que seul le fait qu'une valeur appartienne à un intervalle donné nous intéresse. La valeur en elle-même n'est pas significative, l'on raisonne sur des classes de valeurs. Un exemple classique est de n'utiliser que le nombre d'années pour coder un âge, ou bien de faire des catégories plus vastes : 18–24 ans, 25–35 ans, etc. Réduire revient donc à approcher avec une précision très faible (un ε élevé).

Symboliser est la réduction à outrance. Au lieu de conserver les données sous forme numérique, l'on utilise des symboles. Par exemple, l'on utilisera des symboles comme « enfant », « adolescent », « adulte » en lieu et place de nombres. Cela permet aussi une certaine souplesse sur la signification des symboles. Conceptuellement, symboliser et réduire sont deux mécanismes totalement différents. La symbolisation s'appuie en effet sur des différences significatives : un adulte est majeur.

Dans chacune des trois solutions, il y a une perte d'information évidente.

En réalité, la cause fondamentale de cet antimotif est l'habitude de la discrétisation. Lorsque l'on modélise un système complexe, l'on sait (trop) bien que l'on va avoir à faire des simplifications et donc que l'on va utiliser la discrétisation.

Le concepteur est souvent conscient de la nécessité de bien choisir le niveau de discrétisation, de ne pas trop discrétiser tout en simplifiant au mieux. Le problème vient qu'à mesure que l'on progresse dans la conception, l'on a tendance à oublier que l'on a fait une discrétisation et que l'on n'a alors pas toute l'information.

Pour ancrer notre propos, examinons le cas de l'espace dans lequel les agents évoluent. Dans bon nombre d'applications, cet espace est réduit à un échiquier, un ensemble de cases, souvent carrées, parfois hexagonales. Cette discrétisation de l'espace est souvent largement suffisante pour représenter le milieu spatial du système multi-agent mais elle implique de nombreux problèmes.

Parmi ceux-ci, il y a le problème du choix de la taille des cellules : un agent par cellule, plusieurs cellules pour un agent, plusieurs agents par cellule, etc. Il s'agit en fait de bien choisir le niveau de discrétisation adéquat au problème à gérer.

Un autre problème est celui de la gestion des déplacements : par exemple, dans le cas de cellules carrées, s'agit-il de 4-voisinage ou de 8-voisinage (*i. e.*, les déplacements en diagonale sont-ils autorisés). Cette simple question peut poser de gros problèmes : dans le cas du 8-voisinage, un déplacement en diagonale permet de faire en trois étapes un déplacement qui en aurait pris six en 4-voisinage ; alors se pose aussi le problème de la dépense d'énergie correspondant à chaque type de déplacement.

Un troisième problème lié est celui des obstacles : un passage suffisamment large mais mal discrétisé peut devenir infranchissable.

En ce qui concerne le temps — qui est une composante importante des SMA, surtout lorsqu'il s'agit de simulations —, le choix de sa discrétisation est primordial. En effet, pour une simulation, l'on choisit souvent d'utiliser une exécution pas à pas (cf. l'antimotif *Iniquité*). La longueur d'un pas d'exécution est un choix important : s'il est trop long, des évènements peuvent être ratés, s'il est trop court, c'est le temps global de la simulation (celui perçu par l'utilisateur) qui sera trop long.

De plus, un pas trop court demande des actions primitives (les actions élémentaires demandées par l'agent) trop élémentaires et des décisions très fréquentes de la part de l'agent (et donc des décisions très rapides du point de vue du temps suggestif de l'agent). Le modèle de l'agent dépend donc fortement de ce pas d'exécution.

4. Solution

La première leçon de cet antimotif est bien entendu de faire très attention à la discrétisation des données lors de la conception mais surtout lors de la mise en œuvre, car c'est lors de la mise en œuvre que le choix de la représentation des données est réalisé.

Une autre leçon de cet antimotif est de bien documenter son système, et surtout de spécifier les parties du systèmes où la discrétisation, et donc la perte d'information, a lieu. Il faut aussi explicitement indiquer les raisons des choix appliqués.

Pour ne pas avoir de problème, il faut bien choisir le grain de discrétisation utilisé. Ceci est hélas une évidence, de même que c'en est une de dire que ce choix dépend fortement de l'application en cours de conception. Toutefois, nous pouvons donner quelques indices pour aider à ce choix.

Le premier conseil est de porte une grande attention à ce qui fait information (*i. e.*, ce qui fait différence) pour le problème considéré et les pour les acteurs impliqués.

Pour réaliser une bonne discrétisation, il faut avant tout essayer d'imaginer toutes les situations que le mauvais choix de la granularité pourrait modifier (comme par exemple rendre un passage inaccessible s'il s'agit de discrétiser l'espace). De toutes ces situations, il faut être conscient que certaines sont inévitables et qu'il faut donc faire un compromis lors du choix de la granularité.

D'autre part, l'utilisation de types de données tels que les nombres à virgule flottante permet d'une part de s'approcher le plus possible de la « réalité », et d'autre part de conserver la notion de nombre réel dans le code lui-même (comme partie de la documentation du système).

Dans un autre ordre d'idée, une autre solution proposée par cet antimotif est de ne pas discrétiser à tous les niveaux. Le modèle interne de l'agent, par exemple, n'a pas forcément besoin d'être discrétisé : il peut très bien continuer de « penser » continu même si, à un autre niveau les données sont discrètes (comme par exemple dans le calcul des perceptions ou dans l'application des actions).

5. Discussion

Après l'application des solutions proposées par cet antimotif — solutions que l'on pourrait résumer par : le choix de la bonne granularité, une bonne documentation et éviter la discrétisation à tout crin —, les symptômes doivent, sinon disparaître, à tout le moins trouver une explication et donc une récupération des erreurs pourra être intégrée au système.

Le fait de documenter clairement les endroits de système où les discrétisations sont effectuées et la documentation des liens entre ces discrétisations et les modèles utilisés permettent de modifier facilement la granularité utilisée.

De la même façon, avoir deux modèles (au moins) pour les données de l'agent — le premier étant le modèle interne continu sur lequel les décisions sont prises et le second l'implantation discrétisée de ce modèle —, permet de modifier l'un quasiment indépendamment de l'autre.

Par contre, l'utilisation de plusieurs niveaux repousse le problème du choix de la bonne granularité. Cela le repousse aux niveaux de la mise en œuvre. Cela

ne le résout pas, bien que cela le confine à certains niveaux et évite ainsi sa dissémination dans tout le système.

6. Motifs associés

L'antimotif *Iniquité* peut entrer en collaboration avec cet antimotif lorsqu'il s'agit de discrétiser le temps du système (cf. page 139).

Le motif *Influences* peut utiliser cet antimotif lors de l'utilisation du mécanisme de gradient à vagues (cf. page 105).

⋆⋆⋆

Entité physique

Synomyme : *Entité Rationnelle.*

1. Synopsis

Lorsque l'agent gère au niveau délibératif les actions physiques qui lui sont appliquées, des incohérences surviennent.

2. Symptômes

– Les agents ne réagissent pas correctement aux actions auxquelles ils sont soumis.

– Les actions physiques sur les agents subissent un retard de prise en compte ou sont « oubliées ».

3. DysSolution

La principale cause de ces symptômes est qu'il n'y pas de distinction nette de la partie physique (*i. e.*, celle qui est soumise à l'environnement et aux forces physiques) de la partie proactive et décisionnaire de l'agent.

Ce sont les agents qui gèrent les actions physiques qui leur sont appliquées. L'agent possède, dans son comportement, les mécanismes de traitement de toutes les actions auquel il est soumis, y compris celles qui sont inhérentes à des lois externes (les lois de la physique par exemple) et auxquelles il ne peut que se soumettre.

Nous nous placerons ici dans le cas de l'utilisation du motif *Influences* (cf. page 105), dont le vocabulaire nous permet de mieux cerner les concepts en jeu. Le terme le plus important est celui d'*influence,* qui englobe, sous un même vocable, toutes les perturbations auxquelles un agent peut être soumis : forces, messages, perceptions et autres *stimuli.* Cet antimotif reste bien entendu valable si le motif *Influences* n'est pas appliqué.

Si les deux types d'influences qu'un agent peut recevoir — celles qu'il peut traiter et celles auxquelles il doit obéir, (nous les appelons Stimulus et Force sur le diagramme du motif *Influences*, reproduit ici fig. 40) — sont identiques, alors leur traitement est effectué par le même mécanisme.

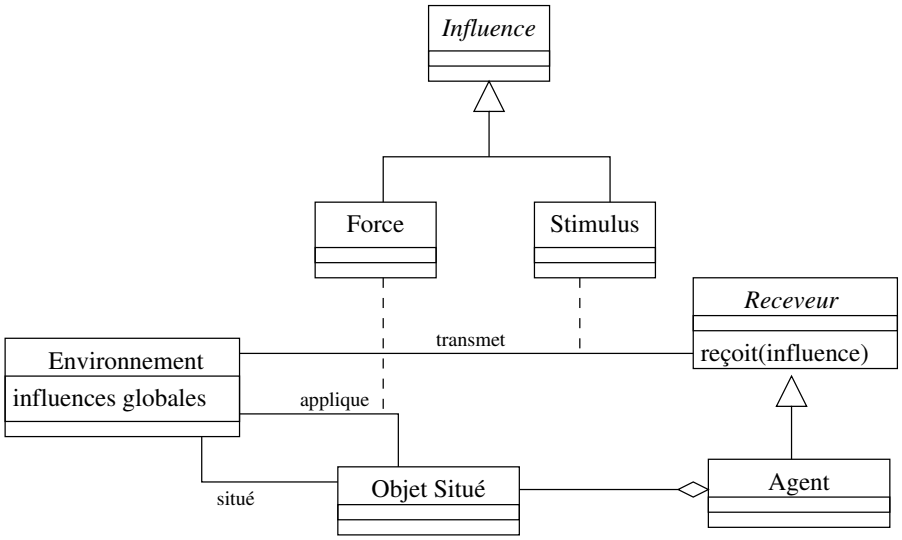


FIG. 40 – Mise en œuvre du motif *Influences*

Le mécanisme de gestion des perturbations de l'agent est un mécanisme décisionnel. En effet, la plus grande différence entre un objet et un agent est que ce dernier peut dire « non ». Il peut donc décider de la réponse en fonction de la perturbation, de son type ou de son origine.

Si ce mécanisme est aussi utilisé pour gérer les actions effectuées sur le corps de l'agent (comme par exemple la force de gravité), alors ce dernier peut tout simplement refuser de l'appliquer. Or, tout ce que l'agent peut faire pour s'opposer à une telle perturbation est de réagir par une perturbation contraire. Il peut réaliser une contre-action pour annuler les effets de l'action. Il ne peut pas annuler l'action physique par sa seule pensée.

4. Solution

La solution que nous proposons pour éviter les symptômes de cet antimotif est la séparation de la gestion des influences que l'agent peut consciemment traiter et celles qu'il doit subir.

En séparant les deux types d'influences, on doit aussi différencier leur traitement par l'agent. Et la solution la plus simple est de séparer l'objet de l'environnement représentant la partie physique de l'agent de l'agent lui-même (comme c'est le cas sur la figure).

L'*entité physique* est donc un objet situé dans l'environnement, il y subit les influences globales ou inhérentes à sa qualité d'objet situé. L'agent n'intervient pas dans le traitement de ces influences.

5. Mise en œuvre

L'une des conséquences de l'application de la dyssolution est que le code de traitement des modifications inéluctables de l'agent (forces de frottement, fatigue, etc.) est réparti au travers du code de l'agent, mélangé aux traitements des perceptions et des messages et à la prise de décision. La solution que nous proposons demande donc de séparer ces mécanismes dans le code : même si les traitements sont similaires, il faut les découpler (en utilisant par exemple l'héritage).

6. Discussion

L'application de cet antimotif permet de différencier la partie physique de l'agent, purement objet, de sa partie décisionnaire, qui contient toute l'application du paradigme agent. Cette différenciation s'applique aussi au système : les contingences physiques de l'environnement sont séparées du SMA, bien que les liens entre ces parties subsistent (notamment le fait que les agents sont situés dans l'environnement et que leurs interactions avec celui-ci influencent son comportement).

Cette différenciation demande aussi plus de travail puisqu'il est alors nécessaire de gérer différemment les « forces » des autres influences que l'agent reçoit. Cet antimotif demande donc de séparer ce qui avait été rapproché par le motif *Influences*.

En fait, le motif *Influences* ne demande pas que toutes les perturbations soient gérées par le même mécanisme. Il propose seulement, qu'à un niveau d'abstraction élevé, toutes les perturbations puissent être considérées de la même façon.

7. Motifs associés

Le motif *Délégation*, [Grand 98, p. 53], peut être considéré comme le motif père de celui-ci. Il est utilisé lorsque l'héritage est impossible ou poserait des problèmes (ici, l'héritage interviendrait entre l'agent et l'objet situé : un agent est un objet situé qui peut réagir et décider).

Ce motif facilite la mise en œuvre du motif *Influences*. C'est d'ailleurs la raison pour laquelle nous avons utilisé le vocabulaire de ce dernier.

⋆⋆⋆

Chapitre V

Une application

An ounce of application is worth a ton of abstraction.

— Anonyme

First shalt thou take out the Holy Pin,
then shalt thou count to three, no more, no less.
Three shalt be the number thou shalt count,
and the number of the counting shalt be three.
Four shalt thou not count, neither count thou two,
excepting that thou then proceed to three. Five is right out.
Once the number three, being the third number, be reached, then
lobbest thou thy Holy Hand Grenade of Antioch towards thou foe,
who being naughty in my sight, shall snuff it.

— MONTY PYTHON, Holy Grail (1974)

Pour appuyer notre thèse, nous développerons ici un exemple concret de l'utilisation des motifs orientés agent que nous venons de décrire. En partant d'un problème simple (en tout cas dans ses extensions envisagées), nous utiliserons nos motifs — ainsi d'ailleurs que des motifs objets, particulièrement lors de la mise en œuvre — pour analyser, concevoir et mettre en œuvre un système multi-agent qui aidera à comprendre et à résoudre ce problème.

Avant d'entrer dans cette application particulière, voyons comment, d'une manière générale, l'on sélectionne et applique des motifs de conception.

1. La consommation de motifs

L'application d'un motif peut prendre deux formes différentes. La première considère que l'on connaît le motif à appliquer. La question que l'on se pose

alors est : Suis-je dans la situation, le contexte, décrit par le motif ? La seconde est plus large : l'on ne connaît pas encore le motif à appliquer. La question est alors double : Quelle est la situation ? Quel est le contexte ? et À quel motif correspond cette situation, ce contexte ? De plus, après la question de la sélection du motif, une autre question se pose : Comment appliquer un motif ?

Nous allons donc aborder le problème de la consommation des motifs dans ces deux aspects : sélection et application.

1.1. Comment sélectionner un motif ?

Pour bien sélectionner un motif de conception, il faut tout d'abord bien comprendre comment un motif peut résoudre un problème, notamment le fait qu'un motif ne propose pas une solution « clef en main » mais des lignes directrices qui permettent de forger sa propre solution, de la même façon qu'un motif ne s'adresse pas à un problème particulier mais à un contexte problématique, à un ensemble de contraintes.

Une fois ces notions bien comprises, il faut avoir une connaissance globale des motifs, savoir dans les grandes lignes quels sont les motifs et les contraintes qu'ils proposent de résoudre. Pour cela, il faut lire le *Synopsis* et les *Forces* de tous les motifs.

L'on perd beaucoup de l'intérêt des motifs si l'on ne les considère qu'individuellement. Il faut donc étudier les relations entre les motifs d'un même catalogue. Ils ont en effet été écrits ensemble et sont destinés à être utilisés ensemble ; certains motifs se complètent ou s'utilisent, d'autres proposent un groupe de solutions alternatives à un ensemble de contraintes similaires. C'est le champ *Motifs associés* qui est ici le plus utile.

Les connaissances nécessaires que nous venons d'énumérer sont des connaissances globales sur le fonctionnement des motifs et sur les principes du catalogue utilisé. Il s'agit donc de connaissances à acquérir avant tout développement. Ensuite, lorsque l'on est en phase de développement, il faut savoir choisir le motif qui pourrait résoudre les contraintes posées. Pour cela, et une fois les contraintes particulières dégagées du problème, il faut déterminer le groupe de motifs qui pourrait s'appliquer et donc étudier les motifs se trouvant dans la même section, et qui ont des buts similaires.

Un autre indice de l'applicabilité d'un motif est de déceler, dans le système que l'on conçoit, les points qui pourraient être la cause d'une reconception du système. Il s'agit des contraintes globales, des besoins, que l'on pense susceptibles de varier dans le cycle de vie du logiciel. Si l'on prévoit plus tôt les parties du logiciel susceptibles d'évoluer — et, par la même occasion, les parties stables —, l'on peut appliquer plus sûrement les techniques qui s'offrent à leur conception et choisir celles qui permettront (ou non) une adaption aux variations des besoins.

Lorsque l'on développe un logiciel, il faut aussi savoir quand ne pas utiliser un motif. En effet, l'application d'un motif peut entraîner de lourdes conséquences : des choix irréversibles (à moins d'une reconception du logiciel) ou simplement une flexibilité trop importante, un nombre d'indirections trop élevé, en somme, une abstraction trop élevée pour un cas concret simple.

1.2. Comment appliquer un motif ?

Une fois le motif sélectionné, il faut savoir l'appliquer. Comme nous l'avons déjà relevé dans la section 4 page 28 du chapitre I, un motif est à la fois instancié, paramétré et adapté, c'est-à-dire utilisé.

C'est pourquoi il est nécessaire de bien lire le motif, entièrement, avant son utilisation. La partie *Discussion,* qui relève les conséquences, utiles ou néfastes, du motif est primordiale.

Ensuite, le champ *Solution* est bien entendu le plus important car il expose les concepts, les principes et les techniques participant à la résolution des contraintes que le motif se propose de lever.

Une fois la solution bien maîtrisée, le champ *Mise en œuvre* est là pour aider à son application dans le développement du logiciel. Ce champ propose des techniques d'application et, souvent, des alternatives, en exposant les contraintes posées ou résolues par chacune.

Un point important de l'utilisation d'un motif est de bien documenter le logiciel, par des commentaires dans le code, les diagrammes construits, la documentation écrite. Cela permet de faire ressortir l'utilisation du motif dans le but d'aider le travail d'équipe et de repérer les points où le motif a été utilisé, pour une future reconception.

Une autre façon de documenter l'utilisation du motif est de faire apparaître, dans le nom des classes, des objets, des algorithmes ou de toute structure ajoutée au logiciel par le biais de l'utilisation du motif, le nom générique utilisé dans le motif pour représenter cette structure.

Une dernière forme d'application des motifs est l'utilisation d'outils d'aide à la conception et au développement (les fameux *CASE tools*) qui intègrent les motifs dans le processus de conception et dans les différents diagrammes construits lors de ce processus. Certains travaux proposent des techniques pour que cette intégration soit réellement un gain pour les concepteurs et les développeurs, [Pavel & Winter 96, Rapicault 99, Sunyé 99]. D'autres travaux proposent une application automatique des motifs, [Eden *et al.* 97b].

—★—

Maintenant que l'on sait choisir le bon motif et que l'on sait comment l'utiliser, l'on peut pratiquer et utiliser les motifs pour le développement de ses logiciels. Nous présentons donc maintenant une application de nos motifs pour le développement d'un SMA.

2. Le sujet de l'expérience

Nous allons simuler un problème-jouet connu, celui de créer un monde virtuel dans lequel des robots poussent des palets à des places définies. Le monde dans lequel évoluent les robots est un espace cartésien à deux dimensions contenant :

- des robots ;
- des palets mobiles ;
- des « places » fixes dans lesquelles doivent être poussés les palets ;
- des prises dans lesquelles les robots viennent se recharger en énergie.

Les robots se déplacent dans ce monde et, en entrant en collision avec eux, ils poussent les autres objets du monde (palets et autres robots). Les places et les prises sont fixes et ne sont pas des obstacles (les palets et les robots peuvent être positionnés dessus).

Dans le modèle le plus simple, tout palet peut être poussé sur toute place. Ce modèle peut être compliqué en fixant, pour chaque palet ou pour chaque type de palet, une place particulière ou un type de place particulier.

3. L'analyse

Nous pouvons séparer le problème en différents aspects :

- des objets (robots, palets, places, prises) sont situés dans un espace ;
- certains de ces objets (robots et palets) suivent certaines règles :
  - ils ne peuvent se chevaucher,
  - ils subissent des accélérations ;
- les agents-robots font plusieurs types d'action :
  - ils se déplacent dans l'espace ;
  - ils poussent des palets vers des places ;
  - ils repèrent les objets, évitent les obstacles ;
  - ils se rechargent auprès de prises.

3.1. Les perceptions et les actions

Pour la transmission des perceptions et des actions, nous suivons le motif *Influences*.

Le motif *Marques* nous propose que les différents objets émettent des marques de reconnaissance et de positionnement. Donc, pour la perception, nous suivons le motif *Marques* :

- les objets physiques émettent des marques pour être repérables ;
- ils émettent aussi un signal de répulsion pour pouvoir être évitables.

3.2. Le comportement de l'agent

Dans cette version simple, l'agent a un comportement réactif simple : s'il est épuisé, il va se recharger à la prise la plus proche, sinon, il cherche et pousse le palet le plus proche vers la place la plus proche. L'agent a donc deux grands comportements : chercher une prise pour se recharger et chercher et pousser un palet vers sa place.

Pour trouver la prise la plus proche, il lui suffit de remonter le gradient d'attraction des marques émises par les prises.

Pour chercher et pousser un palet vers la place la plus proche, la première solution envisagée est d'utiliser deux gradients. Le robot suivra le gradient des marques des palets pour en trouver un, puis suivra celui des places pour pousser le palet vers la place la plus proche.

Les perceptions Il y aura trois gradients d'attraction associés aux marques :

- celui des palets ;

– celui des places ;
– celui des prises.

Le problème posé par cette solution simple est qu'il faut que la marque d'un palet cesse d'être active lorsque celui-ci se trouve sur une place et réciproquement pour la marque de la place. Si ce n'est pas le cas, les robots continueront d'être attirés par des places déjà occupées et par des palets déjà placés. De plus, ceux-ci seront très proches puisque le robot viendra de pousser le palet sur sa place, un comportement trop simpliste de l'agent lui fera alors repousser le palet qu'il vient de placer.

La question est alors de savoir qui désactivera ces marques, et, surtout, comment se rendre compte que le palet est sur la place.

Si l'on décide de placer ce mécanisme dans le palet ou dans la place, l'on est, d'une part, obligé de le placer dans les deux. En effet, pourquoi la place dirigerait-elle le palet plutôt que l'inverse ? D'autre part, pour que ce soit le palet et la place qui repèrent la situation, il leur faudrait une capacité de perception. Ce qui demanderait des capteurs et un mécanisme de vérification à chaque pas. C'est-à-dire que les places et/ou les palets seraient quasiment des agents : ils auraient une boucle perception–délibération–action simple.

Pour éviter cette situation, nous intégrons la recherche de couplage dans un objet plus global : l'environnement. Certes, cela demande que l'environnement fasse la vérification à chaque pas et, surtout, qu'il modifie la marque des places et des palets en conséquence, mais c'est la solution la plus directe (l'environnement a déjà un rôle global par la gestion des perceptions et des actions).

Nous ajoutons un dernier gradient (facultatif), de répulsion cette fois, qui permettra aux agents d'éviter les collisions non désirées (en particulier avec les autres robots).

Poussée Comment un agent-robot peut-il pousser un palet ?

Il suffit que le robot (R_0 sur la figure 41) se place au contact du palet (C), derrière celui-ci sur la direction qui le mène au but (D). Ensuite il peut le pousser en entrant en collision volontaire avec lui.

Il faut aussi faire attention au cas où le robot se trouve entre le palet et la place. En effet, dans ce cas, le robot doit se placer derrière le palet mais en le contournant, sinon, s'il percute le palet, il l'emmènera dans une mauvaise direction. Il utilisera donc son capteur d'évitement d'obstacles.

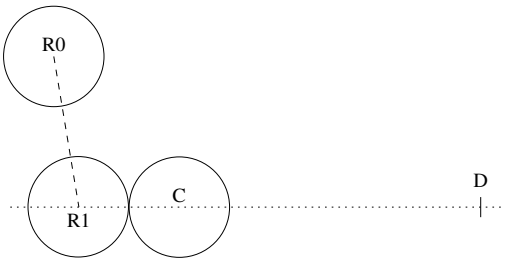


Fig. 41 – Un agent pousse un palet

Pour réaliser ce comportement de poussée, il faudra que l'agent puisse repé-

rer les objets de façon précise (notamment pour connaître la position des palets).

Les capteurs Nous avons donc besoin de trois types de capteurs :

– un pour les gradients d'attraction (le robot aura un capteur pour chacun des trois gradients de ce type) ;

– un pour le gradient de répulsion d'évitement d'obstacles ;

– un capteur positionnel (à courte portée) pour localiser les objets avec précision.

3.3. L'espace

Collisions La gestion des collisions est assez complexe. En effet, une collision modifie les trajectoires des objets et peut donc entraîner d'autres collisions.

Il faut gérer ce problème de manière événementielle : il faut traiter les collisions au moment où elles ont lieu. S'il n'y a pas de collision, les objets continuent sur leur trajectoire.

Nous allons donc fonctionner par étapes :

1. calculer toutes les collisions possibles à partir des trajectoires courantes ;
2. déterminer la date de la première de ces collisions ;
3. avancer le temps du système jusqu'à cette date (*i. e.*, déplacer tous les objets sur leur trajectoire jusqu'à cette date) ;
4. réaliser la première collision (*i. e.*, calculer la nouvelle trajectoire des objets en cause) ;
5. recommencer.

En ce qui concerne les objets, les trajectoires qui résultent d'une collision sont fonction de la trajectoire des objets avant la collision.

Soient o_i et o_j les objets qui entrent en collision, $\vec{v_i}$ la vitesse (*i. e.*, trajectoire) de o_i et $\vec{v_j}$ celle de o_j, $\vec{x_i}$ la position de o_i et $\vec{x_j}$ celle de o_j, m_i la masse de o_i et m_j la masse de o_j, r_i le rayon de o_i et r_j celui de o_j.

Pour simplifier les équations, on posera $\vec{p} = \vec{o_j} - \vec{o_i}$, $\vec{v} = \vec{v_j} - \vec{v_i}$ et $d = r_i + r_j$.

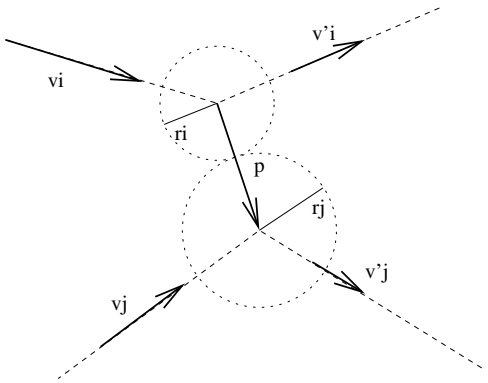


FIG. 42 – Collision entre deux objets

Pour que les deux objets se percutent, il faut que $\vec{p}.\vec{v} < 0$ (première condition initiale).

Le choc aura lieu lorsque les deux objets seront à une distance égale à la somme des rayons. Ceci nous donne une équation du second degré (équation 1), qui n'a de solution que si son discriminant réduit est positif (équation 2, seconde condition initiale).

$$(\vec{p} + \vec{v}.t)^2 = d^2 \tag{1}$$

$$\Delta' = (\vec{p}.\vec{v})^2 - \vec{v}^2.(\vec{p}^2 - d^2) \geq 0 \tag{2}$$

Si les deux conditions initiales sont remplies, le choc a lieu et l'équation 1 a deux solutions positives. La solution minimale est en fait la seule (la seconde correspondrait à un second choc, si les objets ne subissaient pas le premier qui leur fait changer de trajectoire).

$$t_{choc} = \frac{-(\vec{p}.\vec{v}) - \sqrt{\Delta'}}{\vec{v}^2} \tag{3}$$

Nous connaissons donc la date du choc (équation 3) et donc la position des objets à ce moment (puisque l'on connaît leur vitesse et leur position initiale). Il reste à déterminer la trajectoire des objets à partir du choc. Pour simplifier, on considérera le choc comme parfaitement élastique.

En nous plaçant maintenant au moment du choc, et grâce aux principes de la conservation de la quantité de mouvement, de la conservation de l'énergie cinétique totale et de la conservation du moment cinétique total, nous obtenons les trois relations suivantes ($\vec{g_i}$ et $\vec{g_j}$ sont la position respective de o_i et o_j par rapport au centre de gravité de o_i et o_j) :

$$\begin{aligned} m_i\,\vec{v_i}' + m_j\,\vec{v_j}' &= m_i\,\vec{v_i} + m_j\,\vec{v_j} \\ m_i\,{v_i}'^2 + m_j\,{v_j}'^2 &= m_i\,{v_i}^2 + m_j\,{v_j}^2 \\ m_i\,\vec{g_i} \wedge \vec{v_i} + m_j\,\vec{g_j} \wedge \vec{v_j} &= m_i\,\vec{g_i} \wedge \vec{v_i}' + m_j\,\vec{g_j} \wedge \vec{v_j}' \end{aligned}$$

D'où il découle que :

$$\begin{aligned} \vec{v_i}' &= \vec{v_i} + 2\frac{m_j}{m_i + m_j}\left(\frac{\vec{v}.\vec{p}}{d}\right)\frac{\vec{p}}{d} \\ \vec{v_j}' &= \vec{v_j} - 2\frac{m_i}{m_i + m_j}\left(\frac{\vec{v}.\vec{p}}{d}\right)\frac{\vec{p}}{d} \end{aligned}$$

En ce qui concerne les rencontres avec les bords de l'espace, un objet qui percute un bord glissera sur le bord (*i. e.*, sa composante verticale, respectivement horizontale, sera annulée s'il percute un bord horizontal, resp. vertical).

4. La conception

Nous allons simuler le monde et les robots par une simulation à temps discret : l'exécution se fera pas à pas. À chaque pas d'exécution, les influences sont calculées, les perceptions envoyées aux capteurs et les agents décident alors de leurs actions pour ce pas. Les actions seront des influences.

Cette manière de simuler nous demande donc de faire attention aux deux antimotifs *Discrétisation* et *Iniquité*.

Pour simplifier les calculs pour la gestion des objets dans l'espace (collision, déplacement, etc.), nous supposerons tous les objets comme ayant une enveloppe

circulaire dans le plan, et dont la masse est proportionnelle au cube du rayon (*i. e.*, ce sont des cylindres de hauteur proportionnelle au rayon).

En suivant l'application de l'antimotif *Discrétisation*, nous conserverons des coordonnées réelles (ou plutôt à virgule flottante).

En ce qui concerne *Iniquité*, notre application-jouet peut très bien supporter les interférences dues à l'iniquité d'exécution. De plus, l'utilisation du motif *Influences* permet de régler une partie des conflits possibles.

Sur la figure 43 sont représentées les différentes classes du système : une simulation est un ensemble d'agents et d'objets situés dans un environnement.

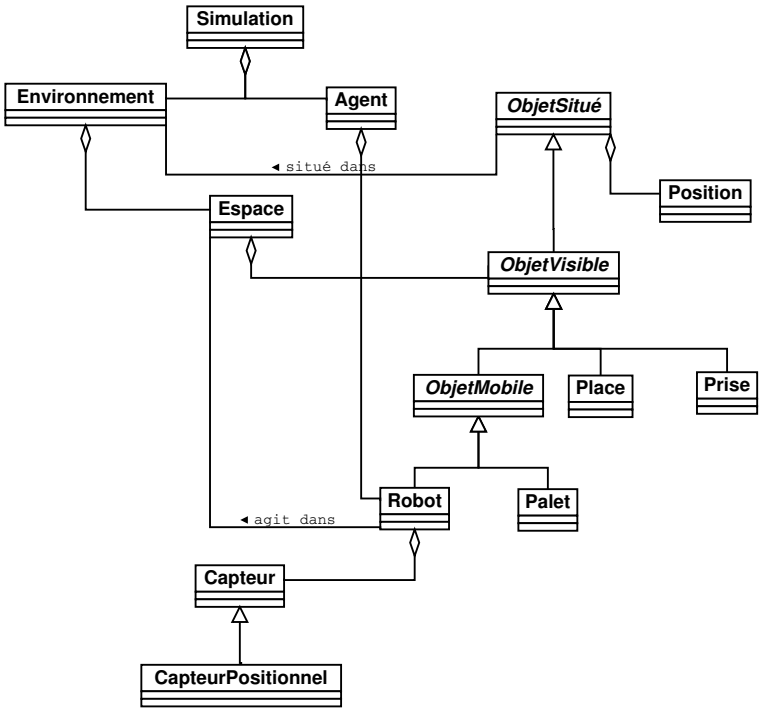


FIG. 43 – Les classes du système

Nous avons appliqué l'antimotif *Entité-Physique* en séparant le Robot de l'Agent qui le gère. De cette façon, le robot subira les forces physiques (déplacement, collision, attraction, répulsion) sans intervention de l'agent (si ce n'est la demande de déplacement lorsqu'il donnera des ordres à son robot).

4.1. Application du motif *Marques*

Le fait qu'une marque ait une position en fait un objet situé, par contre, elle n'est pas visible, bien que l'on puisse la percevoir. Nous avons donc séparé les ObjetsSitués des ObjetsVisibles. De la même façon, les places, ainsi que les prises, ne sont pour nous que des emplacements dans l'espace, elles n'on pas de réelle substance — on peut y placer d'autres objets (ce qui est d'ailleurs le but des agents) — et ne subissent donc pas de force physique (collision, déplacement). Nous avons donc aussi séparé les ObjetsVisibles des ObjetsMobiles.

Les différents types d'objet ont chacun leur marque.

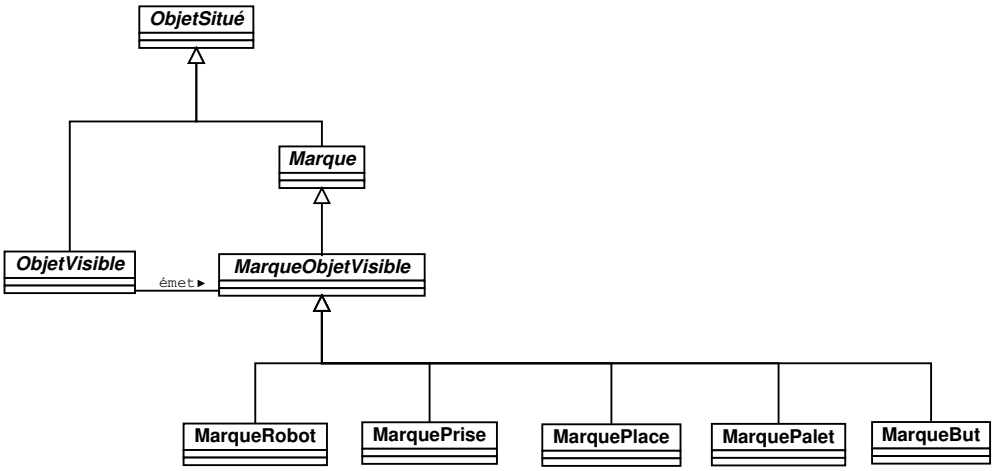


FIG. 44 – Les différentes marques

4.2. Application du motif *Influences*

Il y a deux grands types de Receveurs d'influences : les mécanismes de gestion des influences que sont les Influenceurs et l'Environnement, et les objets qui subissent les influences (les ObjetsMobiles qui subissent des forces physiques et les Capteurs qui sont sensibles à un certain type de signal, généré par certaines influences).

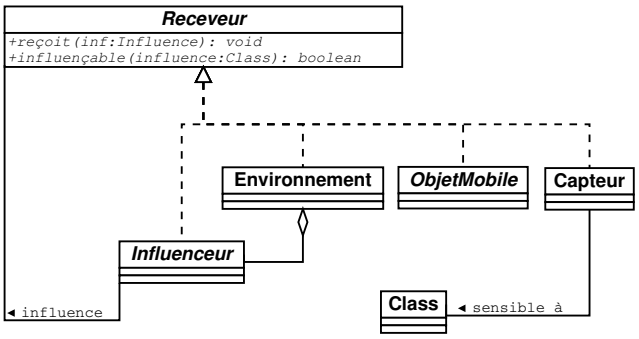


FIG. 45 – La réception des influences

Les Influenceurs sont les objets qui gèrent les influences. Pour cela, ils permettent à des Receveurs de s'inscrire, sur le modèle du motif *Observer*, [GAMMA *et al.* 94], (l'inscription est automatique dans notre application).

Nous avons quatre types d'influenceurs :

– les GradientsMarques qui reçoivent des CréationsMarque et mémorisent les marques correspondantes et qui envoient des InfluencesMarque aux capteurs des robots pour qu'ils se guident ;

– les Localisateurs qui permettent aux capteurs des robots de localiser précisément les objets visibles ;

– le Collideur[1] qui empêche les objets de se chevaucher en gérant les collisions ;

1. De l'anglais *collider*, lui même du latin *collidere* « entrer en collision ».

– et, enfin, le GradientRépulseur qui engendre une Répulsion pour permettre aux robots d'éviter les collisions (facultatif).

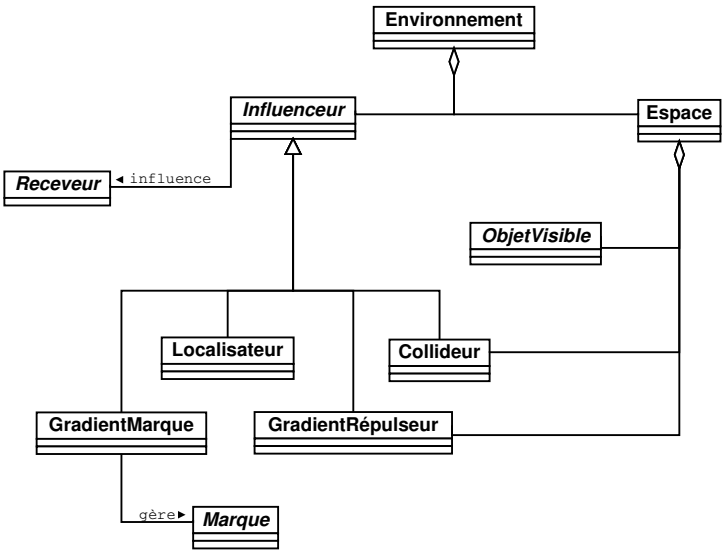


FIG. 46 – La gestion des influences

D'après notre analyse, nous avons besoin de plusieurs types d'influences :

– les influences dues à la présence des marques : InfluenceMarque (ce sont des perceptions) ;

– les influences servant à créer des marques : CréationMarque (cette influence sert à prévenir le GradientMarque correspondant) ;

– les perceptions de localisation : InfluenceObjetVisible ;

– les forces physiques : Accélération ;

– les forces de répulsion : Répulsion (pour l'évitement d'obstacle, contrairement aux Accélérations, les Répulsions peuvent être ignorées) ;

– les influences créées lors de l'apparition ou la disparition d'un objet dans l'espace : ApparitionObjet et DisparitionObjet (ces influences sont générées par la création d'ObjetsVisibles, pour « prévenir » l'espace et les différents mécanismes de gestion des collisions et de l'évitement des obstacles).

Certaines de ces influences agissent sur un objet particulier : les InfluencesDirigées. Elles poussent cet objet dans une Direction précise. Le terme « pousser » pouvant être pris dans le sens de force inéluctable (Accélération) ou de tendance perçue (Répulsion).

Les InfluencesMarques sont les influences envoyées aux capteurs du robot pour qu'il voie le gradient des Marques, en indiquant la direction d'attraction (et la pente) de ce gradient.

Les autres influences, dites *situantes,* servent à indiquer la présence d'un objet, soit aux capteurs du robot, soit aux différents mécanismes de gestion des influences (les Influenceurs).

Les objets qui peuvent recevoir des influences sont appelés des Receveurs. Ils sont capables de recevoir et donc de gérer un certain type d'influence (ce type est vérifiable par la fonction influençable). Ils reçoivent les influences par

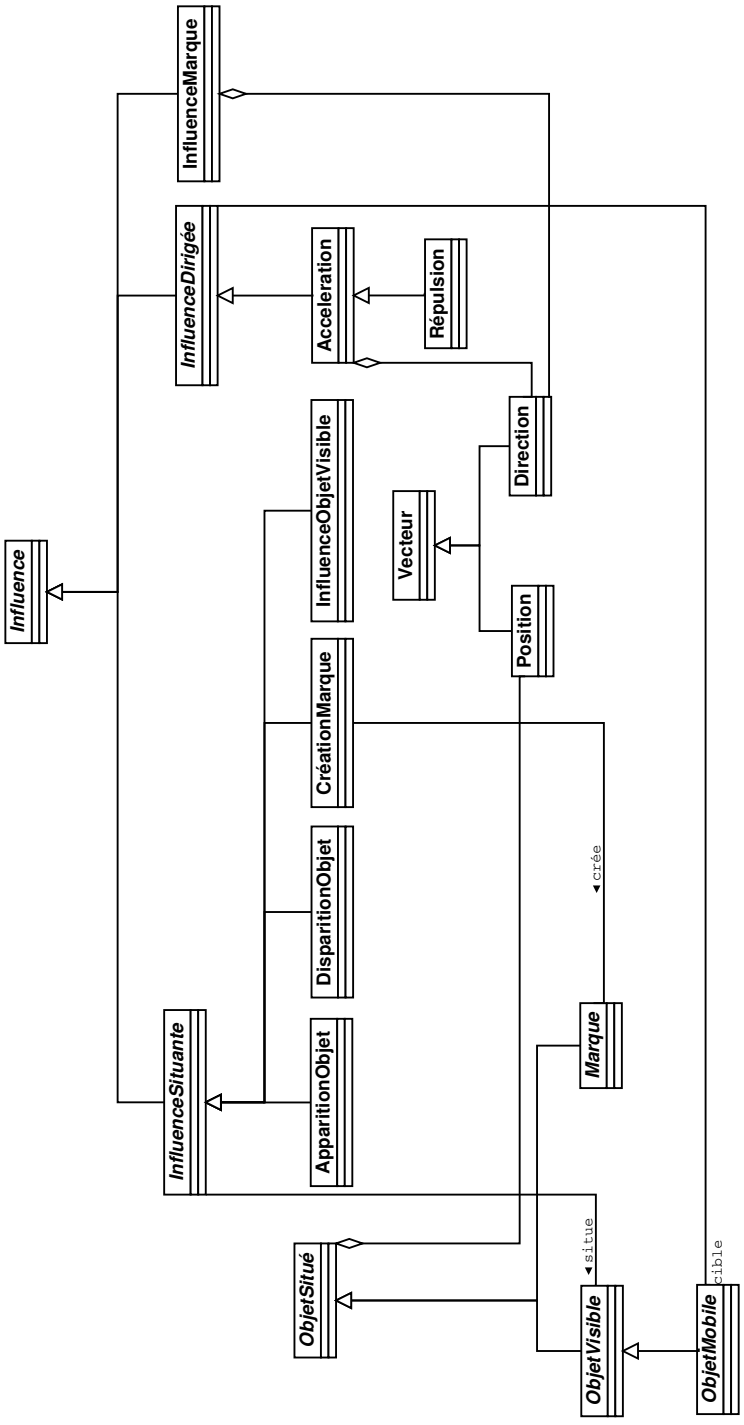
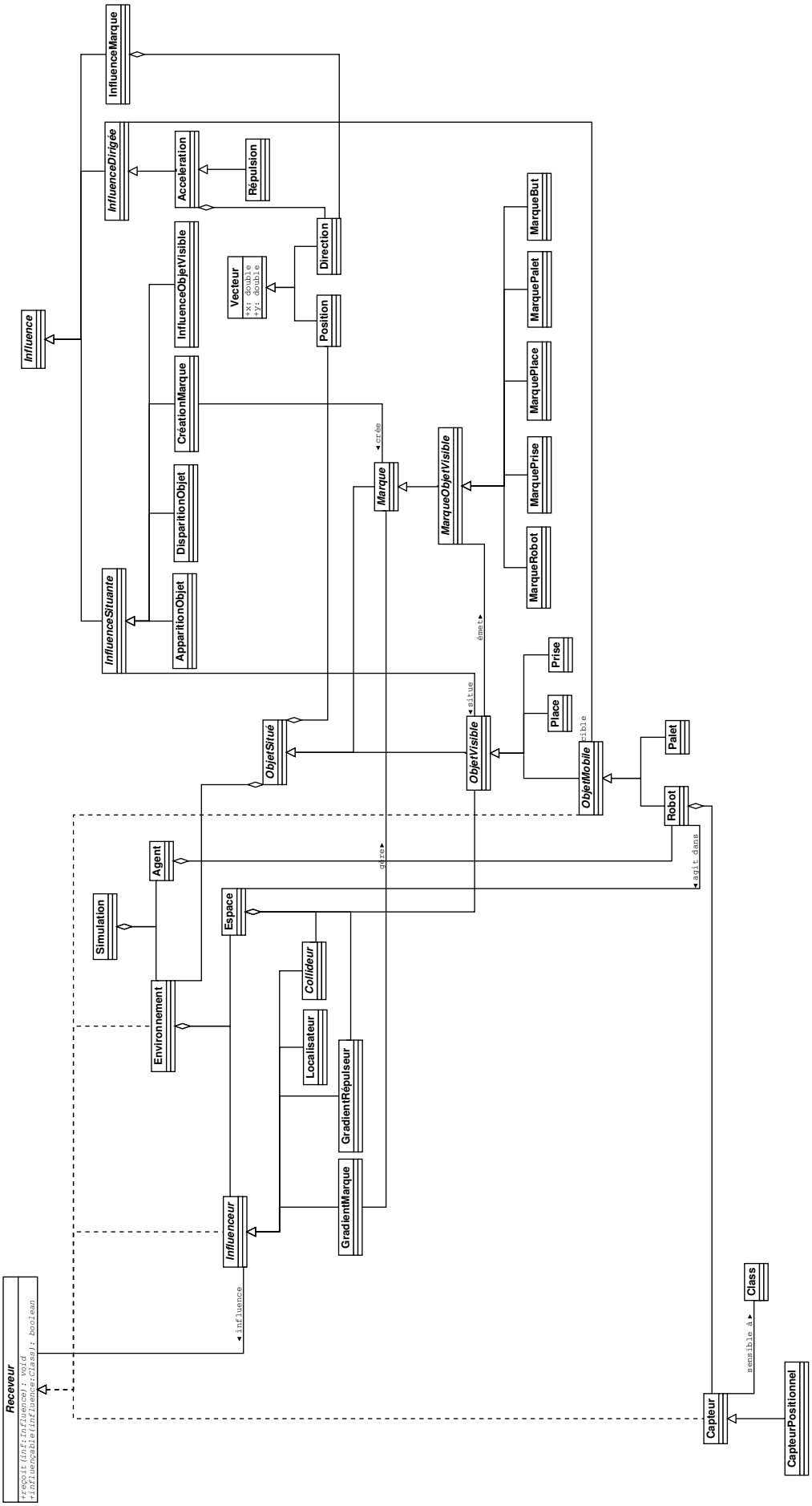


FIG. 47 – Les différentes influences

leur méthode reçoit. Les influences reçues sont stockées pour être combinées et appliquées au prochain pas d'exécution.

4.3. Récapitulatif

Sur la page ci-contre figure un diagramme de classes récapitulatif.


InfluenceMarque
InfluenceDirigée
Acceleration
Répulsion
Direction
Vecteur
+x: double
+y: double
Position
Influence
InfluenceObjetVisible
CréationMarque
DisparitionObjet
ApparitionObjet
InfluenceSituante
crée
Marque
MarqueObjetVisible
MarqueBut
MarquePalet
MarquePlace
MarquePrise
MarqueRobot
émet
Prise
Place
ObjetSitué
situe
ObjetVisible
visible
ObjetMobile
Palet
Robot
gère
agit dans
Agent
Simulation
Espace
Collideur
Environnement
Localisateur
GradientRépulseur
GradientMarque
Influenceur
influence
Receveur
+reçoit(inf:Influence):void
+influençable(influence:Class):boolean
Class
sensible à
Capteur
CapteurPositionnel

5. Mise en œuvre

La programmation s'est faite en Java, en quelques jours. Elle a été plutôt directe, voire quasi-automatique.

Une petite interface graphique nous permet de visualiser l'espace et les objets visibles ainsi que les différents gradients (représentés par des échantillons des forces d'attraction/répulsion qu'ils génèrent à différents points de l'espace).

On peut voir un cliché de cette interface sur la figure 48. Les cases à cocher de la partie droite permettent d'activer et de désactiver l'affichage des gradients (ici, c'est le gradient d'attraction des places qui est représenté par les petits vecteurs).

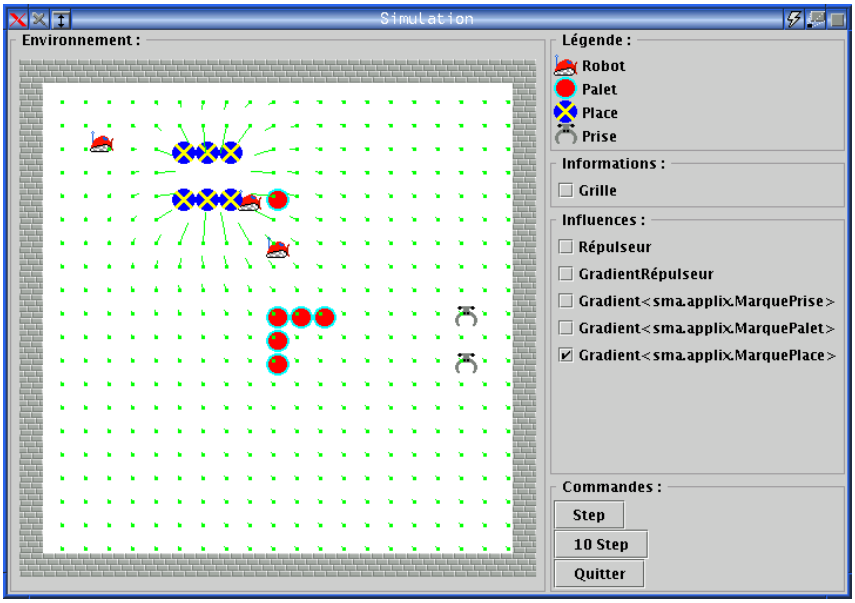


FIG. 48 – L'interface graphique de l'application-jouet

En cliquant sur un objet (palet, place, prise ou robot), on obtient des informations sur l'objet à l'instant du clic (*e. g.*, pour un robot : position, taille, masse, énergie restante, capteurs, cf. figure 49).

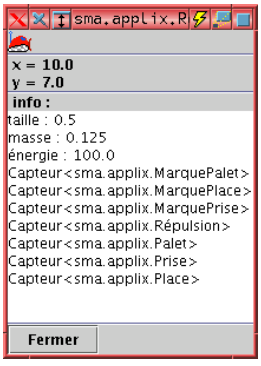


FIG. 49 – Fenêtre d'information sur un objet visible

La simulation est configurée par le biais d'un fichier de configuration XML (voir la DTD et l'exemple utilisé figure 48 page précédente en annexe). Nous avons choisi d'utiliser les possibilités de réflexivité et de chargement dynamique des classes de Java pour la construction de la simulation. Les éléments de configuration indiquent au programme le nom de la classe des objets à créer. Nous pouvons ainsi facilement ajouter des types de palet ou de marque en ne fournissant que le code de ces classes, sans modifier d'autres classes de l'application qu'éventuellement celle de l'agent qui doit pouvoir différencier son comportement suivant les types de palets. Les capteurs utilisent aussi ce mécanisme et la seule classe Capteur permet de créer des capteurs différenciés. Il suffit donc de pourvoir les robots des bons capteurs.

Cela nous permet aussi de modifier le comportement de l'agent en ne lui fournissant que certains capteurs. Par exemple, si l'on ne déclare pas de capteur d'évitement d'obstacle (*i. e.*, un capteur sensible aux Répulsions), les agents se percuteront bien plus souvent. De la même façon, l'on peut spécialiser les agents en spécialisant leurs capteurs (les palets et les places qu'ils ne perçoivent pas seront ignorés).

6. Conclusions

6.1. Sur le choix de l'application

Cette application a été choisie pour ses aspects situé et réactif (l'on peut y intégrer des agents délibératifs mais des agents purement réactifs suffisent en première approximation). De ce fait, l'on pourrait penser qu'elle a été choisie pour que nos motifs soient applicables. Et il est évident qu'elle le fut : pour tester l'applicabilité de motifs qui concernent la conception d'interfaces graphiques, l'on choisit une application graphique avec forces interactions. Donc, le fait de choisir une simulation avec des agents situés pour tester parmi des motifs qui concernent la conception de SMA ceux dont l'un des sujets est la simulation avec des agents situés est logique et n'est pas un biais de ce test.

6.2. Sur le développement et le futur du logiciel

La rapidité de développement de cette application-jouet est bien évidemment due en partie à la simplicité du problème. Toutefois, l'utilisation des différents motifs (surtout *Influences* et *Marques*) et la prise en compte des antimotifs (l'intégration de *Discrétisation* et *Entité-Physique* et le fait de passer outre *Iniquité*) a indiscutablement simplifié et guidé le travail.

Si l'on veut augmenter la complexité du problème, la modification du logiciel sera facilitée par l'utilisation des motifs.

En effet, s'il s'agit de diversifier le nombre de types de palet, chacun étant associé à un type de place particulier, il suffira de sous-classer la classe Palet et la classe MarquePalet, et d'ajouter, dans le fichier de configuration (et dans lui seul), les gradients associés aux marques et la déclaration des capteurs associés à ces gradients (pour que les agents puissent différencier les palets).

S'il s'agit de disposer les places de telle façon que l'ordre de placement des palets prenne une grande importance (pour éviter les blocages par exemple), il suffira de modifer la classe Agent pour y intégrer un des motifs architecturaux

que nous proposons. D'ailleurs, une planification étant nécessaire, une architecture de type vertical trouverait sûrement très bien sa place.

L'on peut aussi vouloir organiser les agents en différenciant leurs capacités ou leurs buts. Nous créerions ainsi un terreau pour les métamotifs *Schémas d'organisation* et *Protocoles* et pour leurs motifs fils.

—★—

Ainsi pensons-nous avoir démontré, grâce à cette petite application, que les motifs, et les motifs orientés agents en particulier, permettent un développement plus rapide et plus souple des SMA.

De plus, cet exemple permet de démontrer que nos motifs sont facilement applicables et aident à la documentation du système multi-agent développé.

Conclusions & perspectives

A conclusion is the place where you got tired of thinking.

— Anonyme

Il est maintenant temps, après avoir décrit nos motifs orientés agent et après avoir exposé leur utilisation dans le développement d'un SMA, de tirer un bilan de nos travaux. Ce bilan se décompose en deux parties : d'une part, sur le travail réalisé et, d'autre part, sur l'expérience et les connaissances acquises lors de cette réalisation.

Nous commencerons par la seconde partie, l'expérience acquise et les conclusions que l'on peut en tirer, pour revenir sur la première, nos réalisations, accompagnées des perspectives qu'elles laissent envisager.

1. Écrire des motifs

La plus grande part de nos réalisations a consisté en l'écriture de motifs de conception orientés agent. Or, produire des motifs, c'est-à-dire décrire des solutions partielles et éprouvées à des ensembles de contraintes, n'est pas une œuvre très facile.

1.1. *Caveat emptor* (précautions d'emploi)

Certes, découvrir un motif n'est pas très compliqué, il suffit de se rendre compte de la co-récurrence d'un problème et de sa solution dans au moins trois systèmes. Mais cela ne suffit pas pour décrire ce motif.

Un des problèmes posés par les motifs est l'effet de mode inhérent à toutes les nouvelles technologies. L'intérêt des motifs réside dans le fait qu'ils forment un lexique, mais l'effet de mode induit une surabondance de motifs, dont une partie se compose de spécialisations de motifs déjà publiés.

À moins de bien organiser les motifs, ce qui n'est pas souvent le cas, l'on en arrive à noyer leur utilité dans un vocabulaire par trop important. Mais,

comme c'est le cas dans les langues naturelles, les doublons ou les mots peu utilisés tendent à disparaître et le langage commun demeure efficace. De même, les jargons, sous-vocabulaires spécialisés, peuvent subsister dans des domaines bien définis.

Une autre précaution à prendre avec les motifs concerne le modèle qu'ils décrivent. Celui-ci doit avoir été utilisé plusieurs fois : c'est le principe même des motifs que de participer d'un processus d'accumulation d'expérience. Ce serait inutile de faire un motif d'un modèle ou d'une architecture qui n'aurait été éprouvé qu'une seule fois. La pluralité des exemples permet l'abstraction, moins il y a d'exemples et moins la « structure qui [les] relie » est visible. C'est la « règle des trois » de l'écriture des motifs : il faut au moins trois exemples.

1.2. Comment (d)écrire un motif ?

Premièrement, l'on ne décrit pas un motif unique. Il est obligatoirement associé à d'autres motifs, dans un catalogue-thésaurus. D'autre part, la description d'un motif n'est jamais définitive, c'est un travail d'écriture et de réécriture, intégrant les réactions de la communauté. Ceci clairement posé et assimilé, comment décrit-on donc des motifs ?

La première chose à faire est de bien répertorier les exemples de chaque motif. Cela permet de mieux cerner le motif et de s'assurer de la « règle des trois » qui demande un minimum de trois exemples d'utilisation du motif. Il est à noter que l'on ne cherche pas *tous* les exemples d'instances de ce motif, seulement des instances significatives, exemplaires (!), qui couvrent le champ d'utilisation du motif.

Ensuite, il faut choisir le format général des motifs. Il ne faut pas hésiter à choisir ses propres champs, avec lesquels l'on sera plus à l'aise. Il n'est en effet pas facile de se couler dans le format d'autres auteurs, principalement parce que leurs motifs ne portent pas sur le même domaine ou ne supposent pas le même contexte général. Il est connu que le format du Groupe de quatre, par exemple, est un des plus difficiles. Le format peut d'ailleurs varier légèrement au cours de l'écriture.

Une fois les motifs ébauchés, l'on peut les classer, suivant leur domaine d'application, suivant les problèmes qu'ils traitent ou suivant la phase de développement dans laquelle ils s'exercent. Cette catégorisation demande parfois une légère modification des champs choisis. Elle permet aussi de faire ressortir les liens entre les motifs que l'on décrit.

La phase d'écriture est la plus difficile. Il faut bien analyser la solution pour pouvoir l'expliquer clairement. Il en est de même pour tous les autres champs du motif (forces, conséquences, etc.). Il faut aussi garder à l'esprit que le motif que l'on écrit est un guide pour le concepteur, il faut donc qu'il soit clair et qu'il réponde à toutes les attentes du lecteur, qu'il comporte tous les points nécessaires à un bon motif (contexte, forces, problèmes ou contraintes traités, solution ou techniques proposées, conséquences entraînées par son utilisation, et, surtout, des exemples).

En fait, nous venons de décrire notre expérience de l'écriture de motifs, mais il existe aussi des « motifs d'écriture de motifs », proposés par des auteurs spécialistes en description de motifs, [Coplien & Woolf 97, Meszaros & Doble 98].

Il existe plusieurs travaux pour faciliter la découverte de motifs. Par exemple, Jagdish Bansiya propose un outil de découverte automatique de l'utilisation de motifs dans un logiciel, [Bansiya 98]. Ce genre d'outils peut permettre d'une part d'aider à documenter un logiciel *a posteriori,* et, d'autre part, à vérifier la pertinence d'un motif. Cet outil est toutefois limité aux motifs de conception objet pour le moment, et nous ne pensons pas qu'une telle technique puisse être employée pour des motifs de niveau d'abstraction plus élevé comme le sont les motifs orientés agent, ou même, plus simplement, les motifs comportementaux.

D'un autre côté, Stéphane Manhes, [Manhes 98], décrit des motifs métiers — c'est-à-dire des motifs relatifs à un ensemble d'applications particulier — et un moyen d'extraction automatisé de tels motifs depuis des logiciels existants. Il s'agit là encore de motifs orientés objet. Le principe est de repérer des motifs dans un ensemble de logiciels (*i. e.* des structures d'associations répétées dans les différents codes et diagrammes).

1.3. Une formalisation

Pour finir, revenons sur une question importante que l'on peut se poser vis à vis de l'aspect formel des motifs.

En effet, l'écriture d'un motif se fait en grande partie en langue naturelle, à laquelle sont adjoints des schémas. Peut-on donc vraiment parler de formalisation ?

Clairement, nous répondons par l'affirmative (oui !). Les champs proposés permettent surtout de définir les points importants à aborder, leur ordre et donc les interactions qu'ils entretiennent.

Est-ce que l'utilisation de la langue naturelle rend cette formalisation trop souple ? Cela dépend en fait du degré d'avancement du motif. Au début de sa rédaction, les champs guident l'auteur dans sa description. Ensuite, les retours de la communauté permettent d'améliorer cette description.

Donc, les motifs sont une capitalisation de l'expertise, réalisée par une formalisation de triplets contexte, problème, solution visant à décrire la solution proposée par une pratique éprouvée résolvant certaines contraintes récurrentes dans un contexte précis.

2. Bilan et perspectives

Les différents motifs orientés agent que nous avons décrits dans ce mémoire apportent différents avantages à la conception de systèmes multi-agents. D'une part, en tant que motifs, ils apportent au développement de SMA tous les avantages que les motifs peuvent apporter au développement de logiciels (cf. le chapitre *Avantages,* page 61). D'autre part, en tant que motifs orienté agents, ils contribuent à la fois au domaine du développement de SMA et à la technique même des motifs.

Ces motifs agent permettent de mieux comprendre le paradigme agent : la collecte d'informations et le rapprochement des différents travaux qui servent d'exemples aux motifs apportent tant à la théorisation et à l'abstraction qu'à l'applicabilité et à l'application de modèles, d'architectures ou de méthodes qui semblaient bien connus, sans toutefois l'être réellement. De plus, l'adéquation des motifs comme aide à l'apprentissage du paradigme agent, comme les motifs

de [Gamma *et al.* 94] le sont pour le paradigme objet, est une contribution importante à ce paradigme.

Si nous reprenons les caractéristiques principales du paradigme agent que nous avons avancées dans l'introduction de ce mémoire (plus exactement section 2.3 page 6), nous pouvons, à chacune d'entre elles, faire correspondre un ou plusieurs de nos motifs :

L'aspect systémique est représenté par les motifs *Architecture récursive* (récursivité de la notion de système) et *Schémas d'organisation* (organisation d'un système, coopération et compétition).

Les motifs *Marques*, *Influences* et *Entité Physique* traitent de l'environnement dans les SMA et de la communication entre les agents et l'environnement.

Les motifs *Marques*, *Influences* et *Protocoles* concernent l'aspect communicationnel des SMA et notamment de la négociation.

En ce qui concerne l'autonomie, les antimotifs *Iniquité* et *Entité Physique* explicitent les problèmes que peut poser son intégration logicielle.

Enfin, le problème du contrôle dans les SMA et dans les agents eux-mêmes est traité par les motifs architecturaux *Architecture verticale*, *Architecture horizontale*, *Architecture* BDI et *Architecture récursive* et par les motifs qui traitent de la communication dans les SMA : *Influences* pour l'aspect physique, *Marques* pour l'aspect logique, *Protocoles* pour l'aspect dialogique et *Schémas d'organisation* pour la modélisation.

La technique des motifs, telle que nous l'avons appliquée dans ce mémoire en décrivant nos motifs orientés agent, se montre tout autant applicable à des niveaux d'abstraction plus élevés que la publicité des motifs tendait à le laisser supposer. C'est d'ailleurs cette utilisation des motifs à des niveaux d'abstraction élevés qui différencie principalement nos motifs des motifs agents des autres équipes (cf. chapitre IV page 67, où nous avions qualifié ces travaux de motifs plus orientés objet qu'orientés agent).

—★—

Les motifs que nous avons décrits ici ne sont pas tous aboutis. En effet, un motif n'est jamais vraiment achevé : dans le sens où il s'agit d'une forme déclarative de l'expertise et que l'expertise s'affine et se modifie avec l'expérience.

Ainsi, parmi nos motifs, ceux qui ont été publiés et qui ont fait l'objet de discussion, [Sauvage 01a, Sauvage 01b, Sauvage 01c, Sauvage 02b], sont-ils les plus aboutis (notamment les motifs *Marques* et *Influences*). De même, les motifs que nous avons eu le plus d'occasion de rencontrer dans notre expérience personnelle — parfois même avant de connaître l'existence des motifs — sont-ils eux aussi parmi les plus achevés. De ce fait, nos motifs ont encore besoin d'être affinés et confrontés à l'usage de la communauté et donc validés.

D'autre part, une meilleure intégration de nos motifs dans le thésaurus que nous présentons est nécessaire, d'abord les uns avec les autres, puis avec les autres motifs orientés agent (dont nous avons parlé au chapitre IV) et, enfin, avec des motifs permettant de faciliter leur mise en œuvre (tels les motifs de conception orientés objet).

Parallèlement, il existe sûrement d'autres motifs orientés agents que nous n'avons pas décrits ici. Ne seraient-ce que les motifs organisationnels relatifs au métamotif *Schémas d'organisation* ou les motifs comportementaux qui composent

les différents rôles qu'un agent peut avoir dans les divers protocoles qu'un SMA peut utiliser (cf. le métamotif *Protocoles*). Un thésaurus sur les motifs orientés agent se doit d'être le plus complet possible si l'on veut pouvoir s'en servir comme aide de conception pour tout type de SMA.

—★—

Nous terminerons sur les conclusions de [Gamma *et al.* 94], que nous reprendrons à notre compte :

> *"It's possible to argue that this book hasn't accomplished much. After all, it doesn't present any algorithms or programming techniques that haven't been used before. It doesn't give a rigorous method for designing systems, nor does it develop a new theory of design — it just documents existing designs. You could conclude that it makes a reasonable tutorial, perhaps, but it certainly can't offer much to an experienced object-oriented designer.*
>
> *We hope you think differently. Cataloging design patterns is important. It gives us standard names and definitions for the techniques we use. If we don't study design patterns in software, we won't be able to improve them, and it'll be harder to come up with new ones.*
>
> *This book is only a start. It contains some of the most common design patterns that expert object-oriented designers use, and yet people hear and learn about them solely by word of mouth or by studdying existing systems. Early drafts of the book prompted other people to write down the design patterns they use, and it should prompt even more in its current form. We hope this will mark the start of a movement to document the expertise of software practitioners."*

⋆⋆⋆

Annexe A

DTD de configuration

```
<!-- DTD "Configuration simulation" -->

<!ELEMENT simulation   (espace,
                        influenceur*,
                        objet*,
                        agent*) >
<!ELEMENT espace       (localisateur*) >
<!ELEMENT localisateur  EMPTY >
<!ELEMENT influenceur   EMPTY >
<!ELEMENT objet         EMPTY >
<!ELEMENT agent        (robot) >
<!ELEMENT robot        (capteur*) >
<!ELEMENT capteur       EMPTY >

<!ATTLIST espace
        taille    CDATA #REQUIRED >
<!ATTLIST localisateur
        objet     CDATA #REQUIRED >
<!ATTLIST influenceur
        type      CDATA #REQUIRED
        objet     CDATA #REQUIRED
        annul     CDATA "Rien"
        rayon     CDATA ".05" >
<!ATTLIST objet
        type      CDATA #REQUIRED
        x         CDATA #REQUIRED
        y         CDATA #REQUIRED
        taille    CDATA ".5" >
<!ATTLIST agent
        evitement CDATA "1.0" >
<!ATTLIST robot
        x         CDATA #REQUIRED
```

```
        y         CDATA #REQUIRED
        vitesse   CDATA "1.0"
        taille    CDATA ".5" >
<!ATTLIST capteur
        type      CDATA #REQUIRED
        objet     CDATA #REQUIRED
        rayon     CDATA "0" >
```

Annexe B

Exemple de configuration XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE simulation SYSTEM "sma-config.dtd">

<simulation>
  <espace taille="20">
    <localisateur objet="Place" />
    <localisateur objet="Prise" />
    <localisateur objet="Palet" />
  </espace>
  <objet type="Place" x="6" y="3"/>
  <objet type="Place" x="7" y="3"/>
  <objet type="Place" x="8" y="3"/>
  <objet type="Place" x="6" y="5"/>
  <objet type="Place" x="7" y="5"/>
  <objet type="Place" x="8" y="5"/>
  <objet type="Prise" x="18" y="10"/>
  <objet type="Prise" x="18" y="12"/>
  <objet type="Palet" x="10" y="5"/>
  <objet type="Palet" x="10" y="10"/>
  <objet type="Palet" x="11" y="10"/>
  <objet type="Palet" x="12" y="10"/>
  <objet type="Palet" x="10" y="11"/>
  <objet type="Palet" x="10" y="12"/>
  <!-- influenceur type="Marque" objet="But" / -->
  <influenceur type="Marque" objet="Prise" annul="Robot" />
  <influenceur type="Marque" objet="Palet"  annul="Place" />
  <influenceur type="Marque" objet="Place" annul="Palet" />
  <agent>
    <robot x="8.8" y="5" vitesse=".5">
```

```
    <!-- capteur type="Marque" objet="But" / -->
    <capteur type="Marque" objet="Palet" />
    <capteur type="Marque" objet="Place" />
    <capteur type="Marque" objet="Prise" />
    <capteur type="Répulsion" objet="" />
    <capteur type="Positionnel" objet="Palet" rayon="2" />
    <capteur type="Positionnel" objet="Prise" rayon="2" />
    <capteur type="Positionnel" objet="Place" rayon="2.5" />
  </robot>
</agent>
<agent>
  <robot x="10" y="7" vitesse=".5">
    <!-- capteur type="Marque" objet="But" / -->
    <capteur type="Marque" objet="Palet" />
    <capteur type="Marque" objet="Place" />
    <capteur type="Marque" objet="Prise" />
    <capteur type="Répulsion" objet="" />
    <capteur type="Positionnel" objet="Palet" rayon="1" />
    <capteur type="Positionnel" objet="Prise" rayon="2" />
    <capteur type="Positionnel" objet="Place" rayon="1.5" />
  </robot>
</agent>
<agent>
  <robot x="2.5" y="2.5" vitesse=".5">
    <capteur type="Marque" objet="But" />
    <capteur type="Marque" objet="Palet" />
    <capteur type="Marque" objet="Prise" />
    <capteur type="Marque" objet="Place" />
    <capteur type="Répulsion" objet="" />
    <capteur type="Positionnel" objet="Palet" rayon="1.5" />
    <capteur type="Positionnel" objet="Prise" rayon="2" />
    <capteur type="Positionnel" objet="Place" rayon="1.5" />
  </robot>
</agent>
<!-- agent>
  <robot x="5" y="2.5" vitesse=".5">
    <capteur type="Marque" objet="But" />
    <capteur type="Marque" objet="Palet" />
    <capteur type="Marque" objet="Prise" />
    <capteur type="Marque" objet="Place" />
    <capteur type="Répulsion" objet="" />
    <capteur type="Positionnel" objet="Palet" rayon="1.5" />
    <capteur type="Positionnel" objet="Prise" rayon="2" />
    <capteur type="Positionnel" objet="Place" rayon="1.5" />
  </robot>
</agent>
<agent>
  <robot x="16.5" y="16.5" vitesse=".75">
    <capteur type="Marque" objet="But" />
    <capteur type="Marque" objet="Palet" />
    <capteur type="Marque" objet="Prise" />
```

```
      <capteur type="Marque" objet="Place" />
      <capteur type="Répulsion" objet="" />
      <capteur type="Positionnel" objet="Palet" rayon="1.5" />
      <capteur type="Positionnel" objet="Prise" rayon="2" />
      <capteur type="Positionnel" objet="Place" rayon="1.5" />
    </robot>
  </agent>
  <agent>
    <robot x="1.5" y="16.5" vitesse=".75">
      <capteur type="Marque" objet="But" />
      <capteur type="Marque" objet="Palet" />
      <capteur type="Marque" objet="Prise" />
      <capteur type="Marque" objet="Place" />
      <capteur type="Répulsion" objet="" />
      <capteur type="Positionnel" objet="Palet" rayon="1.5" />
      <capteur type="Positionnel" objet="Prise" rayon="2" />
      <capteur type="Positionnel" objet="Place" rayon="1.5" />
    </robot>
  </agent -->
</simulation>
```

Bibliographie

Alice said,
"I always use an en-dash instead of a hyphen
when specifying page numbers like '480–491' in a bibliography."
— Donald E. KNUTH, The TEXbook (1986)

The devil can cite Scripture for his purpose.
— William SHAKESPEARE, The merchant of Venice (I–3)

Nous avons distingué les ouvrages cités dans ce mémoire (références) des textes qui nous ont guidé dans nos travaux mais qui n'ont pas eu l'occasion d'être cités. Étant donnée leur importance pour nous, nous avons jugé utile d'en citer quelques uns — ceux qui se rapportent le plus à nos recherches —, ne serait-ce que pour donner un contexte plus important aux lecteurs.

Pour faciliter la recherche depuis le texte du présent mémoire, les références sont indiquées par une étoile (⋆) dans la marge gauche.

Nous avons essayé d'être le plus complet possible en ce qui concerne les informations dont nous disposons sur les différentes références bibliographiques utilisées. Ces informations sont plus que suffisantes pour que le lecteur puisse retrouver les articles et ouvrages cités. Nous pensons que les informations complémentaires peuvent faciliter la recherche du lecteur et en même temps lui permettre de situer plus facilement le contexte de l'article ou de l'ouvrage cité.

Enfin, comme le fait justement remarquer l'épigraphe tirée du *Marchand de Venise,* nous espérons que les citations que nous faisons ne dénaturent pas l'esprit des auteurs.

⋆ [**Aarsten *et al.* 96**]
A. Aarsten, D. Brugali & G. Menga, « Patterns for Cooperation », *in* [Plop 96].

⋆ [**Adami *et al.* 98**]
C. Adami *et al.* (dir.), *Artificial Life VI — proceedings of the Sixth International Conference on Artificial Life*, UCLA, Californie (USA), 27–29 juin 1998, A Bradford book,MIT Press, Cambridge, Massachusetts (USA).

⋆ [**Agha & Jamali 99**]
G. A. Agha & N. Jamali, « Concurrent Programming for DAI », *in* [Weiss 99], chap. 12, pages 505–537.

[**Aknine *et al.* 98**]
S. Aknine, S. Pinson & M. Zacklad, « Un système d'agents récursifs pour l'aide au travail coopératif », *in* [Barthès *et al.* 98].

⋆ [**Amiguet 03**]
M. Amiguet, *Moca : un modèle componentiel dynamique pour les systèmes multi-agents organisationnels*, Thèse de doctorat (informatique), université de Neuchâtel (Suisse), 2003.

⋆ [**Aridor & Lange 98**]
Y. Aridor & D. B. Lange, « Agent Design Patterns: Elements of Agent Application Design », *in* P. Sycara & M. J. Wooldridge (dir.), *Agents'98 — Proceedings of the Second International Conference on Autonomous Agents*. ACM Press, mai 1998.

⋆ [**Ash 65**]
R. B. Ash, *Information theory*, Dover Publications, New York (USA), 1965.

[**Baeijs *et al.* 96**]
C. Baeijs, Y. Demazeau & S. Pesty, « Les organisations dans les SMA », *in* [Müller & Quinqueton 96].

⋆ [**Ballet 00**]
P. Ballet, *Intérêts mutuels des SMA et de l'immunologie — Application à l'immunologie, l'hématologie et au traitement d'images*, Thèse de doctorat (informatique), université de Bretagne occidentale, 28 janvier 2000.

⋆ [**Bansiya 98**]
J. Bansiya, « Automating Design-Pattern Identification », *Dr. Dobb's Journal*, pages 20–28, juin 1998.

⋆ [**Barthès *et al.* 98**]
J.-P. Barthès, V. Chevrier & C. Brassac (dir.), *Systèmes multi-agents, de l'interaction à la socialité, JFIADSMA'98*, Pont-à-Mousson, 18–20 novembre 1998, Hermès, Paris.

⋆ [**Batard 96**]
É. Batard, « L'agent comme signe », *in* [Müller & Quinqueton 96].

⋆ [**Bateson 84**]
G. Bateson, *La nature et la pensée*, La couleur des idées, Seuil, Paris, avril 1984.

⋆ [**Bäumer *et al.* 99**]
C. Bäumer, M. Breugst, S. Choy & T. Magedanz, « Grasshopper — A Universal Agent Platform Based on OMG masif and fipa Standards », Rapport technique, IKV++ GmbH, 1999.

[**Bellot *et al.***]
P. Bellot, J.-P. Cottin & J.-F. Monin, « Développement et validation de logiciels. Méthodes formelles », Techniques de l'Ingénieur, n° H 2 550.

⋆ [**Bellot & Matiachoff**]
P. Bellot & C. Matiachoff, « Applications distribuées en Java — Java/RMI et IDL/corba », Techniques de l'Ingénieur, n° H 2 760.

[**Bellot & Robinet**]
P. Bellot & B. Robinet, « Conception de logiciels et portabilité », Techniques de l'Ingénieur, n° H 3 288.

⋆ [**Boissier *et al.* 99**]
O. Boissier, Z. Guessoum & M. Occello, « Dossier plates-formes SMA — Plates-formes de développement de systèmes multi-agents », *Bulletin de l'afia*, n° 39, 10–25, octobre 1999.

[**Bonabeau *et al.* 99**]
É. Bonabeau, M. Dorigo & G. Théraulaz, « L'Intelligence en essaim », *in* [Gleizes & Marcenac 99], pages 25–38.

⋆ [**Bonabeau & Théraulaz 94**]
É. Bonabeau & G. Théraulaz (dir.), *Intelligence collective*, Hermès, Paris, 1994.

⋆ [**Bonasso *et al.* 97**]
R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. Miller & M. Slack, « Experiences with an Architecture for Intelligent, Reactive Agents », *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, n° 1, 1997.

⋆ [**Bonasso *et al.* 95**]
R. P. Bonasso, D. Kortenkamp, D. Miller & M. Slack, « Experiment with an Architecture for Intelligent, Reactive Agents », *in* M. J. Wooldridge, J. P. Müller & M. Tambe (dir.), *Intelligent Agents II — Workshop on Agent Theories, Architectures, and Languages, atal'95*, vol. 1037 de *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Heidelberg, 1995.

⋆ [**Booch *et al.* 01**]
G. Booch, I. Jacobson & J. Rumbaugh, *Unified Modeling Language Specification*, septembre 2001.

⋆ [**Booth & Stewart 93**]
M. Booth & J. Stewart, « Un modèle de l'émergence de la communication », *in* [JFIADSMA 93], pages 9–18.

[**Boukachour *et al.* 98**]
H. Boukachour, A. Cardon, S. Durand & F. Lesage, « Conception d'un SMA adaptatif : application à la gestion de crise », Rapport technique 1998.029, lip6, université Paris VI, 1998.

⋆ [**Bousquet *et al.* 98**]
F. Bousquet, I. Bakam & C. Le Page, « Environnement de simulation cormas — COmmon-pool Resources and Multi-Agent Systems », *in Smaget : SMA & gestion de l'environnement et du territoire*, Clermont-Ferrand, 5 octobre 1998. {Session de tutoriels}

[**Brazier *et al.* 98a**]
F. M. T. Brazier, F. Cornelissen, R. Gustavsson, C. M. Jonker, O. Lindeberg, B. Polak & J. Treur, « Compositional Design and Verification of a MAS for One-to-Many Negotiation », *in International Conference on Multi-Agent Systems, icmas'98*, 1998.

⋆ [**Brazier *et al.* 95a**]
F. M. T. Brazier, B. M. Dunin-Kęplicz, N. R. Jennings & J. Treur, « Formal Specification of Multi-Agent Systems: a Real-World Case », *in* V. Lesser (dir.),

First International Conference on Multi-Agent Systems, icmas'95, pages 25–32, San-Francisco, Californie (USA), juin 1995, MIT Press, Cambridge, Massachusetts (USA).

[**Brazier *et al.* 96a**]
F. M. T. Brazier, B. M. Dunin-Kęplicz, N. R. Jennings & J. Treur, « Modelling Distributed Industrial Processes in a Multi-Agent Framework », *in* G. O'Hare & S. Kirn (dir.), *Towards the Intelligent Organisation — The Coordination Perpective*, Springer-Verlag, Heidelberg, 1996.

[**Brazier *et al.* 97a**]
F. M. T. Brazier, B. M. Dunin-Kęplicz, N. R. Jennings & J. Treur, « Desire: Modelling Multi-Agent Systems in a Compositional Formal Framework », *International Journal of Cooperative Information Systems*, M. Huhns & M. P. Singh (dir.), vol. 6, n° 1, 67–94, janvier 1997. {Numéro spécial Formal Methods in Cooperative Information Systems: Multiagent Systems}

[**Brazier *et al.* 98b**]
F. M. T. Brazier, C. M. Jonker, F. J. Jüngen & J. Treur, « Distributed Scheduling to Support a Call Centre: a Co-operative Multi-Agent Approach », *Applied Artificial Intelligence Journal*, 1998.

[**Brazier *et al.* 96b**]
F. M. T. Brazier, C. M. Jonker & J. Treur, « Formalization of a cooperation model based on joint intentions », *in* J. P. Müller, M. J. Wooldridge & N. R. Jennings (dir.), *Intelligent Agents III — Proc. of the Third International Workshop on Agent Theories, Architectures and Languages,* atal'96, vol. 1193 de *Lecture Notes in Artificial Intelligence*, pages 141–155. Springer-Verlag, Heidelberg, 1996.

[**Brazier *et al.* 96c**]
F. M. T. Brazier, C. M. Jonker & J. Treur, « Modelling Project Coordination in a Multi-Agent Framework », *in* 5^{th} *Workshop on Enabling Technology for Collaborative Enterprises,* wetice'96, pages 148–155. IEEE Computer Society Press, 1996.

⋆ [**Brazier *et al.* 98c**]
F. M. T. Brazier, C. M. Jonker & J. Treur, « Principles of Compositional Multi-Agent System Development », *in* J. Cuena (dir.), *Ifip'98 Conference IT&knows'98*. Chapman & Hall, 1998.

[**Brazier *et al.* 96d**]
F. M. T. Brazier, J. Treur & N. J. E. Wijngaards, « The Acquisition of a Shared Task Model », *in* N. Shadbolt, K. O'Hara & G. Shreiber (dir.), *Advances in Knowledge Acquisition — Proceedings* 9^{th} *European Knowledge Acquisition Workshop,* ekaw'96, vol. 1076 de *Lecture Notes in Artificial Intelligence*, Nottingham (Royaume Uni), mai 1996, Springer-Verlag, Heidelberg.

[**Brazier *et al.* 95b**]
F. M. T. Brazier, J. Treur, N. J. E. Wijngaards & M. Willems, « Formal Specification of Hierarchically (De)Composed Tasks », *in* B. R. Gaines & M. Musen (dir.), 9^{th} *Banff Knowledge Acquisition for Knowledge Based Systems Workshop*, 1995.

[**Brazier *et al.* 96e**]
F. M. T. Brazier, P. A. T. van Eck & J. Treur, « Modelling Cooperative Behaviour for Resource Access in a Compositional Multi-Agent Framework », *in* J. L. Fiadeiro & P.-Y. Schobbens (dir.), 2^{nd} *Workshop of the ModelAge Project*, pages 27–40, Sesimbra (Portugal), 15–17 janvier 1996.

[**Brazier *et al.* 97b**]
F. M. T. Brazier, P. A. T. van Eck & J. Treur, « Modelling a Society of Simple Agents: from Conceptual Specification to Experimentation », *in* R. Conte,

R. Hegselmann & P. Terna (dir.), *Simulating Social Phenomena*, vol. 456 de *Lecture Notes in Economics and Mathematical Systems*, pages 103–109. Springer-Verlag, Heidelberg, 1997.

[**Brazier *et al.* 97c**]
F. M. T. Brazier, P. A. T. van Eck & J. Treur, « Modelling Competitive Cooperation of Agents in a Compositional Multi-Agent Framework », *in* E. Plaza & R. Benjamins (dir.), *Knowledge Acquisition, Modeling and Management*, vol. 1319 de *Lecture Notes in Artificial Intelligence*, pages 317–322. Springer-Verlag, Heidelberg, 1997.

[**Brazier *et al.* 96f**]
F. M. T. Brazier, P. A. T. van Eck, J. Treur, R. Albrecht & H. Herre, « Design of a Modelling Framework for MAS », *in* R. Albrecht & H. Herre (dir.), *Trends in Theoretical Informatics. Bande 89 in Schriftenreihe der Österreichischen Computer Gesellschaft*, pages 173–191, Wien, München, 1996, R. Oldenbourg Verlag.

[**Brazier *et al.* 94**]
F. M. T. Brazier, P. H. G. van Langen, J. Treur, N. J. E. Wijngaards & M. Willems, « Modelling a Design Task in desire: the VT example », rapport technique IR-3777, Faculteit der Wiskunde en Informatica, VU, décembre 1994.

⋆ [**Brillouin 59**]
L. Brillouin, *La science et la théorie de l'information*, Jacques Gabay, 1959.

[**Briot 88**]
J.-P. Briot, « From Object to Actors: Study of a Limited Symbiosis in Smalltalk-80 », rapport de recherche 88-58, RXF (Rank Xerox France)–Laboratoire d'Informatique Théorique et de Programmation, université Pierre et Marie Curie, septembre 1988.

⋆ [**Briot 89a**]
J.-P. Briot, « Actalk : une Plateforme de Modélisation de Langages d'Acteurs en Smalltalk-80 », *in 7^e Congrès afcet Reconnaissance des Formes et Intelligence Artificielle, RFIA'89*, vol. 1, pages 147–161, Paris, novembre–décembre 1989.

[**Briot 89b**]
J.-P. Briot, « Classifying and Designing Actor Languages in the Smalltalk-80 Environment », *in* S. Cook (dir.), *European Conference on Object-Oriented Programming, ecoop'89*, British Computer Society Workshop, pages 109–129, Nottingham (Royaume Uni), juillet 1989, Cambridge University Press, Royaume Uni.

[**Briot 89c**]
J.-P. Briot, *Des Objets aux Acteurs, 1982–1989 : 7 ans de réflexion*, Habilitation à diriger des recherches, Laboratoire d'Informatique Théorique et de Programmation, université Pierre et Marie Curie, 10 juillet 1989.

[**Briot 90**]
J.-P. Briot, « OOCP = OOP + C », *in 3^rd Conference on the Technology of Object-Oriented Languages and Systems, tools Pacific'90*, pages 417–421, Sydney (Australie), novembre 1990.

[**Briot 91**]
J.-P. Briot, « Training in New Programming Technologies: an Experience », *in European Conference on Education and Computer Science in europa'92*, Nuove Frontiere dell'Insegnamento, pages 117–126, Salerne (Italie), novembre 1991, Elea Press, Italie. {Conférence invitée}

[**Briot 93a**]
J.-P. Briot, « 5^e génération d'ordinateurs ou recherche du 3^e type ? », *France Japon Eco*, n° 57, 32–35, hiver 1993. {Article invité dans le dossier spécial « la recherche industrielle »}

[**Briot 93b**]
J.-P. Briot, « Object-Oriented Concurrent Programming: Introducing a New Programming Methodology », *in* J. Dassow (dir.), *7th International Meeting of Young Computer Scientists, IMYCS'92*, Topics in Computer Science, Smolenice (Slovaquie), 1993, Gordon & Breach. {Conférence invitée}

[**Briot 94a**]
J.-P. Briot, « Modélisation et Classification de Langages de Programmation Concurrente à Objets : l'Expérience Actalk », *in Langages et Modèles à Objets, LMO'94*, pages 153–165, Grenoble, octobre 1994.

[**Briot 94b**]
J.-P. Briot, « Object-Oriented Design of a Generic Scheduler », *in* A. Yonezawa, S. Matsuoka & W. Kato (dir.), *Object-Oriented Computing II, JSSST wooc'93*, n° 6 *in* Lecture Notes/Software Science, pages 73–81. Kindai Kagaku Sha, Tokyo, Japon, 1994.

[**Briot 96**]
J.-P. Briot, « Experience in Classification and Reuse of Synchronization Schemes », *in* K. Futatsugi & S. Matsuoka (dir.), *Object Technologies for Advanced Software, isotas'96*, vol. 1049 de *Lecture Notes in Computer Science*, pages 227–249, Kanazawa (Japon), mars 1996, Springer-Verlag, Heidelberg.

[**Briot 99**]
J.-P. Briot, « Programmation d'applications concurrentes et réparties par objets, réflexion, interaction et agents », *Technique et Science Informatiques, TSI*, vol. 19, n° 1-2-3, 107–112, janvier–mars 1999. {Article invité *in*Numéro spécial : an 2000}

[**Briot 00**]
J.-P. Briot, « Actalk: A Framework for Object-Oriented Concurrent Programming — Design and Experience », *in* J.-P. Bahsoun, T. Baba, J.-P. Briot & A. Yonezawa (dir.), *Object-Oriented Parallel and Distributed Programming*, pages 209–231. Hermès, Paris, 2000.

⋆ [**Briot & Cointe 86**]
J.-P. Briot & P. Cointe, « The OBJVLISP Model: Definition of a Uniform, Reflexive and Extensive Object Oriented Language », *in* B. du Boulay, D. Hogg & L. Steels (dir.), *European Conference on Artificial Intelligence, ecai'86*, Advances in Artificial Intelligence-II, pages 225–232, Brighton (Royaume Uni), juillet 1986, North-Holland.

[**Briot & Cointe 87**]
J.-P. Briot & P. Cointe, « A Uniform Model for Object-Oriented Languages Using Class Abstraction », *in* J. McDermott (dir.), *10th International Joint Conference on Artificial Intelligence, ijcai'87*, vol. 1, pages 40–43, Milan (Italie), août 1987.

[**Briot & Cointe 89a**]
J.-P. Briot & P. Cointe, « ClassTalk : une Transposition des Métaclasses d'ObjVlisp à Smalltalk-80 », *in 7e Congrès afcet Reconnaissance des Formes et Intelligence Artificielle, RFIA'89*, pages 127–146, Paris, novembre–décembre 1989.

[**Briot & Cointe 89b**]
J.-P. Briot & P. Cointe, « Programming with ObjVlisp Metaclass in Smalltalk-80 », *Sigplan Notices*, N. Meyrowitz (dir.), vol. 24, n° 10, 419–432, octobre 1989. {Numéro spécial, Conference on Object-Oriented Programming Systems, Languages and Applications, oopsla'89}

[**Briot & de Ratuld 89**]
J.-P. Briot & J. de Ratuld, « Design of a Distributed Implementation of ABCL/1 », *Sigplan Notices*, vol. 24, n° 4, 15–17, avril 1989. {Numéro spécial, Conference on Object-Oriented Programming Systems, Languages and Applications, oopsla'88}

⋆ [**Briot & Gasser 98**]
J.-P. Briot & L. Gasser, « (Jean-Pierre Briot interviews Les Gasser on) Agent and Concurrent Objects », *IEEE Concurrency*, D. Kafura & J.-P. Briot (dir.), vol. 6, n° 4, 74–81, octobre–décembre 1998. {Séries spéciales sur Acteurs et Agents}

[**Briot & Guerraoui 96a**]
J.-P. Briot & R. Guerraoui, « A Classification of Various Approaches for Object-Based Parallel and Distributed Programming », rapport technique 96–01, University of Tokyo, Department of Information Science, Tokyo (Japon), 17 janvier 1996.

[**Briot & Guerraoui 96b**]
J.-P. Briot & R. Guerraoui, « Objets pour la programmation parallèle et répartie : intérêts, évolutions et tendances », *Technique et Science Informatiques, TSI*, A. Napoli & J.-F. Perrot (dir.), vol. 15, n° 6, 765–800, juin 1996. {Numéro spécial, Systèmes à objets : tendances actuelles et évolution}

[**Briot & Guerraoui 96c**]
J.-P. Briot & R. Guerraoui, « Smalltalk for Concurrent Distributed Programming », *Informatik/Informatique, Swiss Informaticians Society, Switzerland*, R. Guerraoui (dir.), n° 1, 16–19, février 1996. {Numéro spécial, Smalltalk}

[**Briot & Guerraoui 97**]
J.-P. Briot & R. Guerraoui, « Smalltalk : du mono-processeur et mono-utilisateur à la programmation concurrente et répartie », *L'Objet*, F. Pachet & H. Mili (dir.), vol. 3, n° 4, 379–391, décembre 1997. {Numéro spécial, Smalltalk}

[**Briot & Guerraoui 99**]
J.-P. Briot & R. Guerraoui, « A Classification of Various Approaches for Object-Based Parallel and Distributed Programming », *in* J. A. Padget (dir.), *Collaboration between Human and Artificial Societies*, vol. 1624 de *Lecture Notes in Artificial Intelligence*, pages 3–29. Springer-Verlag, Heidelberg, 1999. {Conférence invitée}

[**Briot *et al.* 98**]
J.-P. Briot, R. Guerraoui & K.-P. Löhr, « Concurrency and Distribution in Object-Oriented Programming », *ACM Computing Surveys*, vol. 30, n° 3, 291–329, septembre 1998.

[**Briot & Lescaudron 90**]
J.-P. Briot & L. Lescaudron, « Building an Unified Programming Environment for Object-Oriented Concurrent Languages », *in Fifth International Symposium on Computer and Information Sciences iscis V*, vol. 2, pages 835–844, Nevsehir (Turquie), octobre–novembre 1990.

[**Briot & Lescaudron 92**]
J.-P. Briot & L. Lescaudron, « Design of a Generic and Reusable Scheduler for Smalltalk-80 », *in Ecoop'92 Workshop on Object-Based Concurrency and Reuse*, Utrecht (Hollande), juin 1992.

[**Briot & Yonezawa 87**]
J.-P. Briot & A. Yonezawa, « Inheritance and Synchronization in Concurrent OOP », *in* J. Bézivin, J.-M. Hullot, P. Cointe & H. Lieberman (dir.), *European Conference on Object-Oriented Programming, ecoop'87*, vol. 276 de *Lecture Notes in Computer Science*, pages 32–40. Springer-Verlag, Heidelberg, 1987.

⋆ [**Brooks 86**]
R. A. Brooks, « A Robust Layered Control System for a Mobile Robot », *IEEE Journal of Robotics and Automation*, vol. 2, n° 1, 14–23, avril 1986.

[**Brooks 91a**]
R. A. Brooks, « Intelligence without reason », rapport technique 1293, MIT AI Laboratory, avril 1991.

⋆ **[Brooks 91b]**
R. A. Brooks, « Intelligence Without Representation », *Artificial Intelligence Journal*, n° 47, 139–159, 1991.

⋆ **[Brueckner 00]**
S. A. Brueckner, « An Analytic Approach to Pheromone-Based Coordination », *in Fourth International Conference on MultiAgent Systems, icmas'2000*, pages 369–370, Boston, Massachusetts (USA), 10–12 juillet 2000.

⋆ **[Bürckert & Müller 91]**
H. J. Bürckert & J. Müller, « Ratman: Rational Agents Testbed for Multi Agent Networks », *in* Y. Demazeau & J.-P. Müller (dir.), *Decentralized AI2*, 1991.

⋆ **[Buschmann *et al.* 95]**
F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad & M. Stal, *Pattern-oriented software architecture — A system of patterns*, John Wiley & Sons, Chichester, 1995.

[Calderoni *et al.* 97]
S. Calderoni, R. Courdier, S. Leman & P. Marcenac, « Construction expérimentale d'un modèle multi-agent », *in* [Quinqueton *et al.* 97].

⋆ **[Canal 98]**
R. Canal, « Environnement et réaction en chaîne — Le cas des systèmes multi-agents situés », *in* [Barthès *et al.* 98], pages 235–250.

⋆ **[Cardon 98]**
A. Cardon, « Modélisation des systèmes adaptatifs par agents : vers une analyse-conception orientée agents », rapport technique 1998.011, lip6, université Paris VI, 1998.

[Cardozo *et al.* 93]
E. Cardozo, J. F. M. Fernandes, G. M. B. Oliveira, J. S. Sichman & Y. Demazeau, « Decentralized organizations for artificial intelligence: some implementation issues », *in COMDEX/SUCESU-SP, South America '93*, São Paulo (Brésil), août 1993.

⋆ **[Chevrier 93]**
V. Chevrier, « GTMAS : un outil pour la construction et l'évaluation des SMA », *in* [JFIADSMA 93], pages 45–56.

[Choukroun]
M. Choukroun, « Développement rapide d'applications », Techniques de l'Ingénieur, n° H 3 240.

⋆ **[Clements *et al.* 97]**
P. E. Clements, T. Papaioannou & J. Edwards, « Aglets: Enabling the Virtual Enterprise », *in Managing Enterprises — Stakeholders, Engineering, Logistics and Achievement, ME–SELA'97*, 1997.

⋆ **[Collinot & Drogoul 98a]**
A. Collinot & A. Drogoul, « Applying an Agent-Oriented Methodology to the Design of Artificial Organisations: a Case Study in Robotic Soccer », *Journal of Autonomous Agents and Multi-Agent Systems*, n° 1, 1998.

⋆ **[Collinot & Drogoul 98b]**
A. Collinot & A. Drogoul, « Using the Cassiopeia Method to Design a Soccer Robot Team », *Applied Artificial Intelligence Journal*, n° 12, 1998.

⋆ **[Collis & Ndumu 99]**
J. C. Collis & D. T. Ndumu, *The Role Modelling Guide*, août 1999.

⋆ **[Connell 92]**
J. H. Connell, « SSS: A Hybrid Architecture Applied to Robot Navigation », *in IEEE Conference on Robotics and Automation, ICRA92*, pages 2719–2724, 1992.

[**Cooper 98**]
J. W. Cooper, *The Design Patterns Java Companion*, Addison Wesley, Reading, Massachusetts (USA), octobre 1998.

[**Cooper 00**]
J. W. Cooper, *Java Design Patterns: A Tutorial*, Addison Wesley, Reading, Massachusetts (USA), janvier 2000.

⋆ [**Coplien *et al.* 96**]
J. Coplien, A. Robertson, G. Wonderly & M. Lindner, « Unix Shell Patterns », *in* [Plop 96].

⋆ [**Coplien & Woolf 97**]
J. O. Coplien & R. Woolf, « A Pattern Language for Writers' Workshops », *in* [EuroPLoP 97].

[**Dabija 93**]
V. G. Dabija, *Deciding Whether to Plan to React*, PhD thesis, Stanford University, Department of Computer Science, décembre 1993.

[**Darr & Birmingham 96**]
T. P. Darr & W. P. Birmingham, « An agent model for distributed part-selection », rapport technique CSE-TR-307-96, University of Michigan, Department of Electrical Engineering and Computer Science, 1996.

⋆ [**de Rosnay 95**]
J. de Rosnay, *L'Homme symbiotique*, Points Essais, Seuil, Paris, 1995.

⋆ [**Demazeau 95**]
Y. Demazeau, « From interactions to collective behaviour in agent-based systems », *in 1st European conference on cognitive science*, Saint-Malo, avril 1995.

⋆ [**Depke *et al.* 00**]
R. Depke, R. Heckel & J. M. Küster, « Agent-Oriented Modeling with Graph Transformation », *in First International Workshop on Agent-Oriented Software Engineering, aose'2000*, Limerick (Irlande), 10 juin 2000.

⋆ [**Deugo *et al.* 99a**]
D. Deugo, E. A. Kendall & M. Weiss, « Agent Patterns », `http://www.scs.carleton.ca/~deugo/Patterns/Agent/Presentations/AgentPatterns`, 21 novembre 1999.

⋆ [**Deugo *et al.* 99b**]
D. Deugo, F. Oppacher, J. Kuester & I. Von Otte, « Patterns as a Means for Intelligent Software Engineering », *in Proceedings of The International Conference on Artificial Intelligence, IC-AI'99*, pages 605–611. CSREA Press, 1999.

[**Deugo & Weiß 99**]
D. Deugo & M. Weiss, « A Case for Mobile Agent Patterns », *in Mac3, Mobile Agents in the Context of Competition and Cooperation, Autonomous Agents'99*, 1999.

⋆ [**Di Marzo Serugendo 00**]
G. Di Marzo Serugendo, « A Formal Development and Validation Methodology applied to Agent-Based Systems », *in First International Workshop on Agent-Oriented Software Engineering, aose'2000*, Limerick (Irlande), 10 juin 2000.

⋆ [**d'Inverno *et al.* 97**]
M. d'Inverno, D. Kinny, M. Luck & M. J. Wooldridge, « A Formal Specification of dMARS », *in Intelligent Agents IV — Workshop on Agent Theories, Architectures and Languages, atal'97*, Providence, Rhode Island (USA), juillet 1997.

[**Douglass 98**]
B. P. Douglass, *Real-Time UML: Developing Efficient Objects for Embedded Systems*, Object Technology, Addison Wesley, Reading, Massachusetts (USA), mars 1998.

⋆ [**Drogoul 93a**]
A. Drogoul, *De la simulation multi-agent à la résolution collective de problèmes. Une étude de l'émergence de structures d'organisation dans les SMA*, Thèse de doctorat (informatique), université Paris VI, 1993.

[**Drogoul 93b**]
A. Drogoul, « When Ants Play Chess (Or Can Strategies Emerge From Tactical Behaviors?) », *in Modelling Autonomous Agents in a Multi-Agent World, maamaw'93*, 1993.

⋆ [**Drogoul & Fresneau 98**]
A. Drogoul & D. Fresneau, « Métaphore du fourragement et modèle d'exploitation collective de l'espace sans communication ni interaction pour des colonies de robots autonomes mobiles », *in* [Barthès *et al.* 98], pages 99–114.

⋆ [**Drogoul & Zucker 98**]
A. Drogoul & J.-D. Zucker, « Methodological Issues for Designing MAS with Machine Learning Techniques: Capitalizing Experiences from the RoboCup Challenge », Rapport technique 1998.041, lip6, université Paris VI, 1998.

[**Dunin-Kęplicz & Nawarecki 02**]
B. M. Dunin-Kęplicz & E. Nawarecki (dir.), *From Theory to Practice in Multi-Agent Systems, Second International Workshop of Central and Eastern Europe on Multi-Agent Systems, ceemas2001, Cracow, Poland, September 26-29, 2001, Revised Papers*, vol. 2296 de *Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, 2002.

[**Dunin-Kęplicz & Treur 95a**]
B. M. Dunin-Kęplicz & J. Treur, « Compositional Formal Specification of Multi-Agent Systems », *in* M. J. Wooldridge & N. R. Jennings (dir.), *Intelligent Agents — ecai'94 Workshop on Agent Theories, Architectures and Languages, atal'94*, vol. 890 de *Lecture Notes in Artificial Intelligence*, pages 102–117. Springer-Verlag, Heidelberg, janvier 1995.

[**Dunin-Kęplicz & Treur 95b**]
B. M. Dunin-Kęplicz & J. Treur, « Modelling Reasoning and Acting Agents », *in* B. R. Gaines & M. Musen (dir.), *9th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, KAW'95*, pages 22–1–22–20. RDG Publications, Calgary, Canada, 1995.

⋆ [**Durand 96**]
B. Durand, *Simulation multi-agent et épidémiologie opérationnelle. Étude d'épizooties de fièvre aphteuse*, Thèse de doctorat (informatique), université de Caen, juin 1996.

[**Eden *et al.* 97a**]
A. H. Eden, A. Yehudai & J. Y. Gil, « Automatic Application of Design Patterns », *Journal of Object Oriented Programming*, mai 1997.

⋆ [**Eden *et al.* 97b**]
A. H. Eden, A. Yehudai & J. Y. Gil, « Precise Specification and Automatic Application of Design Patterns », *in 12th Annual IEEE International Conference of Automated Software Engineering, ase'97*, 1997.

⋆ [**El Fallah Seghrouchni & Magnin 01**]
A. El Fallah Seghrouchni & L. Magnin (dir.), *JFIADSMA'01*, Montréal (Canada), 12–14 novembre 2001, Hermès, Paris.

[**El Kaim 97**]
W. El Kaim, *Structuration, placement et exécution de composants logiciels dans les applications réparties ou parallèles : Mise en œuvre avec des applications construites selon le paradigme client-serveur sur des architectures matérielles hybrides*, Thèse de doctorat (informatique), université Paris VI, 1997.

⋆ [**Elammari & Lalonde 99**]
M. Elammari & W. Lalonde, « An Agent-Oriented Methodology: High-Level and Intermediate Models », *in Agent-Oriented Information Systems, AOIS1999*, Heidelberg (Allemagne), 14 & 15 juin 1999.

⋆ [**EuroPLoP 96**]
1st European Conference on Pattern Languages of Programming and Computing, Europlop'96, Irsee (Allemagne), 1996.

⋆ [**EuroPLoP 97**]
2nd European Conference on Pattern Languages of Programming and Computing, Europlop'97, Irsee (Allemagne), 1997.

⋆ [**EuroPLoP 98**]
3rd European Conference on Pattern Languages of Programming and Computing, Europlop'98, Irsee (Allemagne), 1998.

[**EuroPLoP 99**]
4th European Conference on Pattern Languages of Programming and Computing, Europlop'99, 1999.

⋆ [**EuroPLoP 00**]
5th European Conference on Pattern Languages of Programming and Computing, Europlop'00, 2000.

[**EuroPLoP 01**]
6th European Conference on Pattern Languages of Programming and Computing, Europlop'01, 2001.

⋆ [**Fenet 01**]
S. Fenet, *Vers un paradigme de programmation pour les applications distribuées basé sur le comportement des insectes sociaux : application à la sécurité des réseaux*, Thèse de doctorat, université Claude Bernard, Lyon I, décembre 2001.

⋆ [**Fenet & Hassas 98**]
S. Fenet & S. Hassas, « Une approche multi-agent de résolution de problème par interaction : cas de l'équilibrage dynamique multi-critères », *in* [Barthès *et al.* 98], pages 115–130.

[**Ferber 95**]
J. Ferber, *Les systèmes multi-agents : vers une intelligence collective*, InterÉdition, Paris, 1995.

[**Ferber & Briot 88**]
J. Ferber & J.-P. Briot, « Design of a Concurrent Language for Distributed Artificial Intelligence », *in International Conference on Fifth Generation Computer Systems, FGCS'88*, vol. 2, pages 755–762, Tokyo (Japon), novembre–décembre 1988.

⋆ [**Ferber & Müller 96**]
J. Ferber & J.-P. Müller, « Influences and Reaction: a Model of Situated Multiagent Systems », *in Second International Conference on Multi-Agent Systems, icmas'96*, Kyoto (Japon), décembre 1996.

⋆ [**Ferguson 91**]
I. A. Ferguson, « Toward an Architecture for Adptative, Rational, Mobile Agents »,

in E. Werner & Y. Demazeau (dir.), *Decentralized AI3 — proceedings of the 3rd European Workshop on Modelling Autonomous Agents in a Multi-Agent World, maamaw'91*, pages 249–261, Kaiserslautern (Allemagne), 5–7 août 1991, Elsevier, Paris.

⋆ **[Ferguson 92]**
I. A. Ferguson, *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*, PhD thesis, university of Cambridge, Royaume Uni, novembre 1992.

[Ferguson 94]
I. A. Ferguson, « Autonomous Agent Control: a Case for Integrating Models and Behaviours », *in AAAI Fall Symposium on Control of the Physical World by Intelligent Agents*, New Orleans, Louisiane (USA), novembre 1994.

[Ferguson 95]
I. A. Ferguson, « Intergrated Control and Coordinated Behavior: a Case for Agent Models », *in* M. J. Wooldridge & N. R. Jennings (dir.), *Intelligent Agents — ecai'94 Workshop on Agent Theories, Architectures and Languages, atal'94*, vol. 890 de *Lecture Notes in Artificial Intelligence*, pages 102–117. Springer-Verlag, Heidelberg, janvier 1995.

⋆ **[Fernandes & Occello 00]**
K. Fernandes & M. Occello, « Une approche multi-agents hybride pour la conception de systèmes complexes à raisonnement intégré », *in Rencontres Jeunes Chercheurs en Intelligence Artificielle, RJCIA'2000*, Lyon (France), 10–13 septembre 2000.

⋆ **[Firby 89]**
R. J. Firby, *Adaptive Execution in Complex Dynamic Worlds*, PhD thesis, Yale University, janvier 1989.

[Firby 92]
R. J. Firby, « Building Symbolic Primitives with Continuous Control Routines », *in Proceedings of the 1st International Conference on Artificial Intelligence Planning System*, pages 62–69, College Park, Maryland (USA), juin 1992.

[Firby 93]
R. J. Firby, « Interfacing the RAP System to Real-Time Control », note de travail du projet Animate Agent AAP-1, University of Chicago, juillet 1993.

⋆ **[Firby *et al.* 95]**
R. J. Firby, R. E. Kahn, P. N. Prokopowicz & M. J. Swain, « An Architecture for Vision and Action », *in Proceedings of the International Joint Conference on Artificial Intelligence, ijcai'95*, pages 72–79, août 1995.

⋆ **[Fischer *et al.* 95]**
K. Fischer, J. P. Müller & M. Pischel, « Unifying Control in a Layered Agent Architecture », *in* V. Lesser (dir.), *First International Conference on Multi-Agent Systems, icmas'95*, San-Francisco, Californie (USA), juin 1995, MIT Press, Cambridge, Massachusetts (USA).

⋆ **[Fischmeister & Lugmayr 99]**
S. Fischmeister & W. Lugmayr, « The Supervisor-Worker Pattern », *in* [Plop 99].

⋆ **[Fisher & Wooldridge 96]**
M. Fisher & M. J. Wooldridge, « On the Formal Specification and Verification of Multi-Agent Systems », *International Journal of Cooperative Information System*, 1996.

⋆ **[Foisel 98a]**
R. Foisel, « Construire des systèmes multi-agents à partir de schémas d'interactions », *in* [Barthès *et al.* 98], pages 295–308.

⋆ **[Foisel 98b]**
R. Foisel, *Modèle de réorganisation de SMA : Une approche descriptive et opérationnelle*, Thèse de doctorat (informatique), université Henri Poincaré, Nancy I, 13 novembre 1998.

⋆ **[Foisel *et al.* 97]**
R. Foisel, V. Chevrier & J.-P. Haton, « Un modèle pour la réorganisation de SMA », *in* [Quinqueton *et al.* 97].

⋆ **[Fowler & Scott 00]**
M. Fowler & K. Scott, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Object Technology, Addison Wesley, Reading, Massachusetts (USA), 2nde édition, avril 2000.

⋆ **[Gamma *et al.* 93]**
E. Gamma, R. Helm, R. E. Johnson & J. Vlissides, « Design Patterns: Abstraction and Reuse of Object-Oriented Design », *in Conférence ecoop'93*. Springer-Verlag, Heidelberg, 1993.

⋆ **[Gamma *et al.* 94]**
E. Gamma, R. Helm, R. E. Johnson & J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, Reading, Massachusetts (USA), 1994, xv–395 pages.

⋆ **[Gasser & Briot 92]**
L. Gasser & J.-P. Briot, « Object-Based Concurrent Programming and Distributed Artificial Intelligence », *in* N. M. Avouris & L. Gasser (dir.), *Distributed Artificial Intelligence: Theory and Praxis*, Eurocourses — Computer and Information Science, pages 81–107. Kluwer, 1992.

⋆ **[Gat 92]**
E. Gat, « Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots », *in AAAI'92*, 1992.

⋆ **[Geib & Merle]**
J.-M. Geib & P. Merle, « Corba : des concepts à la pratique », Techniques de l'Ingénieur, n° H 2 758.

[Ghanea-Hercock *et al.* 99]
R. Ghanea-Hercock, J. C. Collis & D. T. Ndumu, « Co-operating Mobile Agents for Distributed Parallel Processing », *in Autonomous Agents'99*, 1999.

⋆ **[Ghédira 01]**
K. Ghédira (dir.), *EcollA'01, École d'Intelligence Artificielle*, Hammamet (Tunisie), 24–26 mai 2001.

⋆ **[Ghesquière 98]**
H. Ghesquière, *L'Entreprise et son système d'information : pièges, illusions, défis*, mémoire de DES, UTC (Université de Technologie de Compiègne), imi (Institut du Management de l'Information), janvier 1998.

⋆ **[Ginot & Le Page 98]**
V. Ginot & C. Le Page, « Mobidyc, a Generic Multi-Agent Simulation for Modelling Populations Dynamics », *in Smaget : SMA & gestion de l'environnement et du territoire*, Clermont-Ferrand, 5 octobre 1998.

[Girard-Faugère 98]
M. Girard-Faugère, « Agents Logiciel : Quel est le coût de la distribution ? », rapport technique 1997.010, lip6, université Paris VI, 1998.

⋆ **[Girault 02]**
F. Girault, *L'Environnement comme espace de cognition*, Thèse de doctorat (informatique), université de Caen, décembre 2002.

[Girault & Stinckwich 99a]
F. Girault & S. Stinckwich, « Footux Team description: A Hybrid recursive based agent architecture », *in* M. M. Veloso (dir.), *Ijcai Third International Workshop on RoboCup*, pages 103–108, Stockholm (Suède), 31 juillet–6 août 1999, Springer-Verlag, Heidelberg.

[Girault & Stinckwich 99b]
F. Girault & S. Stinckwich, « Une architecture d'anticipation par réalité augmentée », *in* [Gleizes & Marcenac 99], pages 253–264.

⋆ **[Gleizes & Marcenac 99]**
M.-P. Gleizes & P. Marcenac (dir.), *Ingénierie des systèmes multi-agents, JFIADSMA'99*, Saint-Gilles, Île de la Réunion, 8–10 novembre 1999, Hermès, Paris.

⋆ **[Gorodetski *et al.* 02]**
V. Gorodetski, O. Karsayev, I. Kotenko & A. Khabalov, « Software Development Kit for Multi-agent Systems Design and Implementation », *in* [Dunin-Kęplicz & Nawarecki 02], pages 121–130.

⋆ **[Grand 98]**
M. Grand, *Patterns in Java*, vol. 1, John Wiley & Sons, Chichester, 1998.

⋆ **[Grand 99]**
M. Grand, *Patterns in Java*, vol. 2, John Wiley & Sons, Chichester, 1999.

⋆ **[Grasce 94]**
Grasce, Ingénierie système. De la conception orienté objet à la conception orientée projet, Aix-en-Provence, 1994.

⋆ **[GrassHopper 99]**
« Grasshopper — A Platform for Mobile Software Agents », Rapport technique, IKV++ GmbH, 1999.

⋆ **[Gruer *et al.* 00]**
P. Gruer, V. Hilaire & A. Koukam, « Towards Verification of Multi-Agents Systems », *in Fourth International Conference on MultiAgent Systems, icmas'2000*, pages 393–394, Boston, Massachusetts (USA), 10–12 juillet 2000.

⋆ **[Guerrin *et al.* 98]**
F. Guerrin, R. Courdier, S. Calderoni, J.-M. Paillat, J.-C. Soulié & J.-D. Vally, « Conception d'un modèle multi-agent pour la gestion des effluents d'élevage à l'échelle d'une localité rurale », *in* [Barthès *et al.* 98], pages 25–37.

⋆ **[Guessoum & Briot 98a]**
Z. Guessoum & J.-P. Briot, « Building Agents as an Extension of Active Objects — Application to the Simulation of Economic Models », *in 4e Colloque Africain sur la Recherche en Informatique, cari'98*, pages 39–54, Dakar (Sénégal), octobre 1998, Presses Universitaires de Dakar — inria. {Conférence invitée}

[Guessoum & Briot 98b]
Z. Guessoum & J.-P. Briot, « From Active Objects to Autonomous Agents », rapport technique 1998.015, lip6, université Paris VI, 1998.

⋆ **[Guessoum & Briot 99]**
Z. Guessoum & J.-P. Briot, « From Active Object to Autonomous Agents », *IEEE Concurrency*, D. Kafura & J.-P. Briot (dir.), vol. 7, n° 3, 68–76, juillet–septembre 1999. {Séries spéciales sur Acteurs et Agents}

[**Guessoum *et al.* 97**]
Z. Guessoum, J.-P. Briot & M. Dojat, « Des objets concurrents aux agents autonomes », *in* [Quinqueton *et al.* 97].

⋆ [**Guessoum & Occello 01**]
Z. Guessoum & M. Occello (dir.), *Plate-forme afia 2001 — Atelier Méthodologie et Environnements pour les Systèmes Multi-Agents*, Grenoble, 25–28 juin 2001.

[**Guillemet *et al.* 99**]
A. Guillemet, G. Haïk, T. Meurisse, J.-P. Briot & M. Lhuillier, « Mise en œuvre d'une approche componentielle pour la conception d'agents », *in* [Gleizes & Marcenac 99], pages 53–65.

⋆ [**Gutknecht & Ferber**]
O. Gutknecht & J. Ferber, « http://www.madkit.org », le site officiel de madkit.

⋆ [**Gutknecht & Ferber 98**]
O. Gutknecht & J. Ferber, « Un méta-modèle organisationnel pour l'analyse, la conception et l'exécution de systèmes multi-agents », *in* [Barthès *et al.* 98], pages 267–280.

⋆ [**Hameurlain 02**]
N. Hameurlain, « Formal Semantics for Behaviourial Substitutability of Agent Components: Application to Interaction Protocols », *in* [Dunin-Kęplicz & Nawarecki 02], pages 131–140.

⋆ [**Hilaire *et al.* 00**]
V. Hilaire, A. Koukam, P. Gruer & J.-P. Müller, « Formal Specification and Prototyping of Multi-Agent Systems », *in* A. Omicini, R. Tolksdorf & F. Zambonelli (dir.), *Engineering Societies in the Agent World, First International Workshop, ESAW2000, Berlin, Germany, August 21, 2000, Revised Papers*, vol. 1972 de *Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, 2000.

[**Hoare 87**]
C. A. R. Hoare, *Processus séquentiels communicants*, Masson, Paris, 1987.

⋆ [**Hodgson *et al.* 99**]
A. Hodgson, R. Rönnquist & P. Busetta, « Specification of Coordinated Agent Behaviour (The SimpleTeam Approach) », rapport technique 99-05, Agent Oriented Softwre Pty. Ltd., Melbourne (Australie), octobre 1999.

⋆ [**Holland 95**]
J. H. Holland, *Hidden Order — How Adaptation Builds Complexity*, Addison Wesley, Reading, Massachusetts (USA), juillet 1995, xxi–185 pages.

⋆ [**Howden *et al.* 01**]
N. Howden, R. Rönnquist, A. Hodgson & A. Lucas, « Jack Intelligent Agents™ — Summary of an Agent Infrastructure », *in* 5^{th} *International Conference on Autonomous Agents*, 2001.

⋆ [**Hung & Pasquale 99**]
E. Hung & J. Pasquale, « Agent Usage Patterns: Bridging the Gap Between Agent-Based Application and Middleware », rapport technique CS1999-0638, Department of Computer Science and Engineering, University of California, San Diego, Californie (USA), 17 novembre 1999.

⋆ [**Iglesias *et al.* 98**]
C. A. Iglesias, M. Garijo & J. C. González, « A Survey of Agent-Oriented Methodologies », *in* J. P. Müller, M. P. Singh & A. S. Rao (dir.), *Intelligent Agents V — Fifth International Workshop on Agent Theories, Architectures, and Languages, atal'98*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg, 1998.

⋆ **[Jean 97]**
M. R. Jean, « Émergence et SMA », *in* [Quinqueton *et al.* 97]. {Groupe de travail « Collectif »}

⋆ **[Jennings 99]**
N. R. Jennings, « Agent-Based Computing: Promise and Perils », *in Ijcai'99*, pages 1429–1436, Stockholm (Suède), 31 juillet–6 août 1999, Springer-Verlag, Heidelberg.

[Jennings 00]
N. R. Jennings, « On agent-based software engineering », *Artificial Intelligence Journal*, n° 117, 277–296, 2000.

⋆ **[Jeon *et al.* 00]**
H. Jeon, C. Petrie & M. R. Cutkosky, « Jatlite: A Java Agent Infrastructure with Message Routing », *IEEE Internet Computing*, mars–avril 2000.

[Jézéquel *et al.* 02]
J.-M. Jézéquel, H. Hussmann & S. Cook (dir.), *«UML»2002 The Unified Modeling Language*, n° 2460 *in* Lecture Notes in Computer Science, Dresde (Allemagne), septembre–octobre 2002, Springer-Verlag, Heidelberg.

⋆ **[JFIADSMA 93]**
Premières journées francophones IAD & SMA. Hermès, Paris, 1993.

⋆ **[Johnson 97]**
R. E. Johnson, « Frameworks = (Components + Patterns) », *Communications of the ACM*, vol. 40, n° 10, 39–42, octobre 1997.

[Jonker & Treur 98a]
C. M. Jonker & J. Treur, « Compositional Verification of MAS: a Formal Analysis of Pro-activeness and Reactiveness », *in* W. de Roever, H. Langmaack & A. Pnueli (dir.), *Proceedings of the International Workshop on Compositionality, COMPOS'97*, vol. 1536 de *Lecture Notes in Computer Science*, pages 350–380. Springer-Verlag, Heidelberg, 1998.

[Jonker & Treur 98b]
C. M. Jonker & J. Treur, « A Generic Multi-Agent Architecture for Interactive Diagnostic Tasks with Clarification », *in* J. Cuena (dir.), *Ifip'98 Conference IT&knows'98*. Chapman & Hall, 1998.

[Jonker *et al.* 98]
C. M. Jonker, J. Treur & W. de Vries, « Compositional Verification of Agents in Dynamic Environments: a Case Study », *in* F. van Harmelen (dir.), *KR'98 Workshop on Verification and Validation of KBS*, 1998.

[Jonker *et al.* 00]
C. M. Jonker, J. Treur & W. de Vries, « Reuse and Abstraction in Verification: Agents Acting in dynamic Environments », *in First International Workshop on Agent-Oriented Software Engineering, aose'2000*, Limerick (Irlande), 10 juin 2000.

⋆ **[Jung 99]**
C. G. Jung, *Theory and Practice of Hybrid Agents*, PhD thesis, Technischen Fakultät der Universität des Saarlandes, Saarebruck (Allemagne), septembre 1999.

⋆ **[Jung & Fischer 98]**
C. G. Jung & K. Fischer, « Methodological Comparison of Agent Models », rapport technique RR-98-01, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (Allemagne), 1998.

⋆ **[Kendall 98a]**
E. A. Kendall, « Agent Roles and Role Models: New Abstractions for Intelligent

Agent Systems Analysis and Design », *in Intelligent Agent for Information and Process Management,* AIP'98, 1998.

[**Kendall 98b**]
E. A. Kendall, « Goals and Roles: The Essentials of Object Oriented Business Process Modelling », *in Workshop on Object Oriented Business Process Modelling,* ecoop'98, juin 1998.

⋆ [**Kendall *et al.* 96**]
E. A. Kendall, M. T. Malkoum & C. H. Jiang, « The Layered Agent Pattern Language », *in* [Plop 96].

⋆ [**Kendall *et al.* 97**]
E. A. Kendall, M. T. Malkoum & C. H. Jiang, « The Layered Agent Pattern Language », *in* [Plop 97].

⋆ [**Kendall *et al.* 98**]
E. A. Kendall, P. V. Murali Krishna, C. V. Pathak & C. B. Suresh, « Patterns of Intelligent and Mobile Agents », *in* P. Sycara & M. J. Wooldridge (dir.), *Agents'98 — Proceedings of the Second International Conference on Autonomous Agents*. ACM Press, mai 1998.

[**Kiss 96**]
G. Kiss, « Agent Dynamics », *in* G. M. P. O'Hare & N. R. Jennings (dir.), *Foundations of Distributed Artificial Intelligence*, New York (USA), 1996.

⋆ [**Klügl *et al.* 98**]
F. Klügl, F. Puppe, U. Raub & J. Tautz, « Simulating Multiple Emergent Phenonema — Exemplified in an Ant Colony », *in* [Adami *et al.* 98].

⋆ [**Knuth 86**]
D. E. Knuth, *The TEXbook*, Addison Wesley, Reading, Massachusetts (USA), 1986, x–483 pages.

⋆ [**Kristensen 95**]
B. B. Kristensen, « Object-Oriented Modeling with Roles », *in Proceedings of the 2nd International Conference on Object-Oriented Information Systems*, pages 57–71, Dublin (Irlande), 1995, Springer-Verlag, Heidelberg.

⋆ [**Kristensen & Olsson 96**]
B. B. Kristensen & J. Olsson, « Roles & Patterns in Analysis, Design and Implementation », *in* Oois'96 *— Proceedings of the 3rd International Conference on Object-Oriented Information System*, Londres (Royaume Uni), 1996.

⋆ [**Kristensen & Østerbye 96**]
B. B. Kristensen & K. Østerbye, « Roles: Conceptual Abstraction Theory & Practical Language Issues », *Theory and Practice of Object Systems,* tapos, 1996.

⋆ [**Kuwana *et al.* 96**]
Y. Kuwana, I. Shimoyama, Y. Sayama & H. Miura, « A Robot that Behaves like a Solkworm Moth in the Pheromone Stream », *in* [Langton & Shimohara 96].

⋆ [**Labbani-Igbida 98**]
O. Labbani-Igbida, *Contribution à une méthodologie de conception de comportements collectifs émergents dans une colonie de robots miniatures et autonomes*, Thèse de doctorat (informatique), université de Franche-Comté, janvier 1998.

[**Labrou & Finin 97**]
Y. Labrou & T. Finin, « A Proposal for a new KQML Specification », rapport technique CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, 3 février 1997.

⋆ **[Langton & Shimohara 96]**
C. Langton & K. Shimohara (dir.), *Artificial Life V — proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*, Nara-Ken New Public Hall, Nara (Japon), 16–18 mai 1996, A Bradford book,MIT Press, Cambridge, Massachusetts (USA).

⋆ **[Lansky & Georgeff 87]**
A. L. Lansky & M. P. Georgeff, « Reactive Reasoning and Planning », *in Proceedings of the Sixth National Conference on Articial Intelligence, AAAI*, pages 677–682, Seattle, Washington (USA), 1987.

[Laublet]
P. Laublet, « Méthodes de développements d'applications à objets », Techniques de l'Ingénieur, n° H 3 228.

[Le Ber *et al.* 98]
F. Le Ber, A. Dury & V. Chevrier, « Un modèle multi-agent pour la simulation en agronomie : usages et comparaisons », *in* [Barthès *et al.* 98], pages 11–24.

⋆ **[Le Moigne 90]**
J.-L. Le Moigne, *Modélisation des systèmes complexes*, Dunod, Paris, 1990.

⋆ **[Le Strugeon *et al.* 93]**
E. Le Strugeon, R. Mandiau & G. Libert, « Proposition d'organisation dynamique d'un groupe d'agents en fonction de la tâche », *in* [JFIADSMA 93], pages 217–227.

[Lebarbé 02]
T. Lebarbé, *Hiérarchie Inclusive des Unités Linguistiques en Analyse Syntaxique Coopérative — Le segment, unité intermédiaire entre chunk et phrase dans le traitement linguistique par système multi-agent*, Thèse de doctorat (informatique), université de Caen, mai 2002.

⋆ **[Lebarbé & Girault 01]**
T. Lebarbé & F. Girault, « Tapas : traitement et analyse pas perception augmentée en syntaxe », *Revue de l'association française de linguistique appliquée*, 2001.

[Lescaudron *et al.* 91]
L. Lescaudron, J.-P. Briot & M. Bouabsa, « Prototyping Programming Environments for Object-Oriented Concurrent Languages: a Smalltalk-Based Experience », *in 5th Conference on the Technology of Object-Oriented Languages and Systems, tools USA'91*, pages 449–462, Goleta, Californie (USA), août 1991, Prentice-Hall.

⋆ **[Lhuillier 98]**
M. Lhuillier, *Une approche à base de composants logiciels pour la conception d'agents. Principes et mise en œuvre à travers la plate-forme Maleva.*, Thèse de doctorat (informatique), université Paris VI, février 1998.

⋆ **[Lind 00]**
J. Lind, « Issues in Agent-Oriented Software Engineering », *in First International Workshop on Agent-Oriented Software Engineering, aose'2000*, Limerick (Irlande), 10 juin 2000.

⋆ **[Ma & Shi 00]**
G. Ma & C. Shi, « Modeling Social Agents in BDO Logic », *in Fourth International Conference on MultiAgent Systems, icmas'2000*, pages 411–412, Boston, Massachusetts (USA), 10–12 juillet 2000.

[Magnin 96]
L. Magnin, *Modélisation et simulation de l'environnement dans les systèmes multi-agents. Application aux robots footballeurs*, Thèse de doctorat (informatique), université Paris VI, novembre 1996.

⋆ [**Manhes 98**]
S. Manhes, *Les patterns métier : extraction dans l'existant logiciel*, mémoire de DEA et de stage SILR3, irin (Institut de Recherche en Informatique de Nantes) & ireste (Institut de Recherche et d'Enseignement Supérieur aux Techniques de l'Électronique), 16 septembre 1998.

⋆ [**Massingill *et al.* 99**]
B. L. Massingill, T. G. Mattson & B. A. Sanders, « Patterns for Parallel Application Programs », *in* [Plop 99].

⋆ [**McCormick & Malveaux 98**]
H. W. McCormick & R. Malveaux, « AntiPatterns », *Dr. Dobb's Journal*, pages 44–50, juin 1998.

⋆ [**McCormick *et al.* 98**]
H. W. McCormick, R. Malveaux, T. Mowbray & W. J. Brown, *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*, John Wiley & Sons, Chichester, 1998.

⋆ [**Meira *et al.* 00**]
N. Meira, I. Conde e Silva & A. Silva, « An Agent Pattern Language for a More Expressive Approach », *in* [EuroPLoP 00].

[**Merlat 98**]
W. Merlat, *Adaptation dynamique de l'organisation dans les SMA. Application à la conception des Systèmes Coopératifs Distribués et Ouverts*, Thèse de doctorat (informatique), université Paris VI, 17 décembre 1998.

⋆ [**Meszaros & Doble 98**]
G. Meszaros & J. Doble, « A Pattern Language for Pattern Writing », *in* [Plop 98].

[**Meurisse & Briot 01**]
T. Meurisse & J.-P. Briot, « Une approche à base de composants pour la conception d'agents », *Technique et Science Informatiques, TSI*, M. Dao & C. Dony (dir.), vol. 20, n° 4, 583–602, avril 2001. {Numéro spécial : Réutilisation}

[**Mezura *et al.* 99**]
C. Mezura, M. Occello, Y. Demazeau & C. Baeijs, « Récursivité dans les systèmes multi-agents : vers un modèle opérationnel », *in* [Gleizes & Marcenac 99], pages 41–52.

⋆ [**Minar *et al.* 96**]
N. Minar, R. Burkhart, C. Langton & M. Askenazi, « The swarm Simulation System: A Toolkit for Building Multi-agent Simulations », rapport interne 96-06-042, Santa Fe Institute, Santa Fe, Nouveau Mexique (USA), 21 juin 1996. {Disponible à `http://www.swarm.org/intro-papers.html`}

⋆ [**Minar *et al.* 99**]
N. Minar, M. Gray, O. Roup, R. Krikorian & P. Maes, « Hive: Distributed Agents for Networking Things », *in* *Asa/ma'99*, 3 août 1999, URL `http://hive.media.mit.edu`.

⋆ [**Monekosso *et al.* 02**]
N. Monekosso, P. Remagnino & A. Szarowicz, « An Improved Q-Learning Algorithm Using Synthetic Pheromones », *in* [Dunin-Kęplicz & Nawarecki 02], pages 197–207.

[**Morand 94**]
B. Morand, « Quelques arguments sur les méthodes orientées objets », *in* [Grasce 94].

[**Morand 00**]
B. Morand, « Le diagramme : à la périphérie ou au cœur de la cognition ? », *in Actes*

des 9e Journées de Rochebrune, Rencontres interdisciplinaires sur les représentations graphiques dans les systèmes complexes naturels et artificiels, février 2000.

⋆ **[Mulder *et al.* 97]**
M. Mulder, J. Treur & M. Fischer, « Agent modelling in concurrent MetateM and desire », *in Intelligent Agents IV — Workshop on Agent Theories, Architectures and Languages, atal'97*, Providence, Rhode Island (USA), juillet 1997.

⋆ **[Müller 96]**
J. P. Müller, *The design of intelligent agents: a layered approach*, vol. 1177 de *Lecture Notes in Computer Science*, Springer-Verlag, Heidelberg, décembre 1996.

⋆ **[Müller 98]**
J.-P. Müller, « Vers une méthodologie de conception de systèmes multi-agents de résolution de problèmes par émergence », *in* [Barthès *et al.* 98], pages 355–371.

⋆ **[Müller *et al.* 01]**
J.-P. Müller, J. Baez, M. Amiguet & A. Nagy, « La plate-forme moca : réification de la notion d'organisation au dessus de madkit », *in* [El Fallah Seghrouchni & Magnin 01], pages 307–310.

⋆ **[Müller & Pischel 93]**
J. P. Müller & M. Pischel, « The Agent Architecture InteRRaP: Concept and Application », rapport de recherche RR-93-26, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (Allemagne), 1993.

⋆ **[Müller & Quinqueton 96]**
J.-P. Müller & J. Quinqueton (dir.), *IAD & SMA, JFIADSMA'96*. Hermès, Paris, 1996.

⋆ **[Nakamura & Kurumatani 96]**
M. Nakamura & K. Kurumatani, « Formation Mechanism of Pheromone Pattern and Control of Foraging Behavior in an Ant Colony Model », *in* [Langton & Shimohara 96].

[Nanard 02]
J. Nanard, « Les patrons dans la conception et la réalisation d'hypermédias — Vers une opérationalisation », *Revue d'Interaction Homme-Machine*, vol. 3, n° 1, 41–77, 2002.

[Ndumu *et al.* 99]
D. T. Ndumu, H. S. Nwana, L. C. Lee & J. C. Collis, « Visualising and Debugging Distributed Multi-agents Systems », *in Autonomous Agents'99*, 1999.

⋆ **[Nicolle 02]**
A. Nicolle, « L'Organisation sociale et les systèmes multi-agents », *in Les cahiers du greyc*, n° 4, 2002.

[Nicolle & de Almeida 99]
A. Nicolle & V. S.-D. de Almeida, « Vers un modèle des interactions langagières », *in* B. Moulin, S. Delisle & B. Chaib-draa (dir.), *Analyse et Simulation de Conversations : de la théorie des actes de discours aux systèmes multi-agents*, pages 133–169. Limonest : L'interdisciplinaire, 1999.

⋆ **[Nicolle & Morand 93]**
A. Nicolle & B. Morand, « Processus de conception des SI : objectifs, fonctions et statut des modèles », *in Troisième table ronde francophone sur la conception, 01 Design'93*, novembre 1993.

[Nwana & Ndumu 96]
H. S. Nwana & D. T. Ndumu, « An introduction to agent technology », *British Telecom Technology Journal*, vol. 14, n° 4, octobre 1996.

⋆ [**Nwana *et al.* 98**]
H. S. Nwana, D. T. Ndumu & L. C. Lee, « Zeus: An Advanced Tool-Kit for Engineering Distributed Multi-Agent Systems », *in Paam'98*, pages 377–392, mars 1998.

⋆ [**Nwana *et al.* 97**]
H. S. Nwana, D. T. Ndumu, L. C. Lee & J. C. Collis, « Zeus: A Collaborative Agents Tool-Kit », *in 2nd UK Workshop on Foundations of Multi-Agent Systems, fomas'97*, pages 45–52, 1997.

[**Nwana *et al.* 99**]
H. S. Nwana, D. T. Ndumu, L. C. Lee & J. C. Collis, « Zeus: A Toolkit for Building Distributed Multi-Agent Systems », *in Agents'99 — Proceedings of the Third International Conference on Autonomous Agents*, 1999.

⋆ [**Occello & Demazeau 97**]
M. Occello & Y. Demazeau, « Vers une approche de conception et de description récursive en univers multi-agent », *in* [Quinqueton *et al.* 97].

⋆ [**Occello & Koning 00**]
M. Occello & J.-L. Koning, « Multiagent Oriented Software Engineering: An Approach Based on Model and Software Reuse », *in* P. Petta & J. P. Müller (dir.), *Second International Symposium From Agent Theory to Agent Implementation*, Vienne (Autriche), 25–28 avril 2000.

⋆ [**OCM 00**]
Objets Composants Modèles, OCM'2000, 18 mai 2000.

⋆ [**Odell *et al.* 00**]
J. Odell, H. Van Dyke Parunak & B. Bauer, « Extending UML for Agents », *in First International Workshop on Agent-Oriented Software Engineering, aose'2000*, Limerick (Irlande), 10 juin 2000.

⋆ [**Oechslein *et al.* 02**]
C. Oechslein, F. Klügl, R. Herrler & F. Puppe, « UML for Behaviour-Oriented Multi-Agent Simulations », *in* [Dunin-Kęplicz & Nawarecki 02], pages 217–226.

⋆ [**Omicini 00**]
A. Omicini, « Soda: Societies and Infrastructures in the Analysis and Design of Agent-based Systems », *in First International Workshop on Agent-Oriented Software Engineering, aose'2000*, Limerick (Irlande), 10 juin 2000.

⋆ [**Ovalle & Garbay 93**]
A. Ovalle & C. Garbay, « Identification cognitive, modélisation conceptuelle et implantation technologique : vers une méthodologie pour la conception de SMA », *in* [JFIADSMA 93], pages 33–44.

⋆ [**Parsons & Giorgini 99**]
S. Parsons & P. Giorgini, « An approach to using degrees of belief in BDI agents », *in* B. Bouchon-Meunier, R. R. Yager & L. A. Zadeh (dir.), *Information, Uncertainty, Fusion*. Kluwer, Dordrecht, 1999.

⋆ [**Patterns'Page**]
« `http://hillside.net/patterns` », la page dédiée aux patterns, très fournie en liens et en bibliographies.

⋆ [**Pavel & Winter 96**]
B.-U. Pavel & M. Winter, « Towards Pattern-Based Tools », *in* [EuroPLoP 96].

⋆ [**Peschansky *et al.* 00**]
F. Peschansky, T. Meurisse & J.-P. Briot, « Les Composants Logiciels : évolution technologique ou nouveau paradigme ? », *in* [OCM 00], pages 53–65.

[**Piaggio *et al.* 96**]
M. Piaggio, A. Sgorbissa & R. Zaccaria, « A Hybrid Robotic System for a Mobile Robot », *Frontiers in Artificial Intelligence and Applications*, vol. 35, 249–258, 1996.

⋆ [**Plop 94**]
« `http://st-www.cs.uiuc.edu/~plop` », page officielle des conférences plop (Pattern Languages of Programs), 1994.

⋆ [**Plop 96**]
3rd Conference on Pattern Languages of Programming, Plop'96, Monticello, Illinois (USA), septembre 1996.

⋆ [**Plop 97**]
Plop'97 (Pattern Languages of Programs), Monticello, Illinois (USA), 1997.

⋆ [**Plop 98**]
Plop'98 (Pattern Languages of Programs), Monticello, Illinois (USA), 1998.

⋆ [**Plop 99**]
Plop'99 (Pattern Languages of Programs), Monticello, Illinois (USA), 15–18 août 1999.

⋆ [**Plop 00**]
Plop'00 (Pattern Languages of Programs), Monticello, Illinois (USA), 13–16 août 2000.

⋆ [**Printz**]
J. Printz, « Génie logiciel », Techniques de l'Ingénieur, n° H 3 208.

⋆ [**Quinqueton *et al.* 97**]
J. Quinqueton, M.-C. Thomas & B. Trousse (dir.), *Intelligence artificielle et systèmes multi-agents, JFIADSMA'97*, Nice, 1997, Hermès, Paris.

⋆ [**Rao & Georgeff 91**]
A. S. Rao & M. P. Georgeff, « Modeling Rational Agents within a BDI-architecture », note technique 14, Australian Artificial Intelligence Institute, février 1991.

⋆ [**Rapicault 99**]
P. Rapicault, « Vers une meilleure intégration des design patterns à la conception », 7 juin 1999. {Gracq (Groupe de Travail en Acquisition et Ingénierie des Connaissances)}

⋆ [**Ricordel 01**]
P.-M. Ricordel, *Programmation Orientée Multi-Agents : Développement et Déploiement de Systèmes Multi-Agents Voyelles*, Thèse de doctorat (informatique), institut national polytechnique de Grenoble, 25 octobre 2001.

⋆ [**Riveill *et al.***]
M. Riveill, R. Balter & F. Boyer, « Communication synchrone entre programmes par RPC et RMI », Techniques de l'Ingénieur, n° H 2 738.

[**Riveill & Merle**]
M. Riveill & P. Merle, « Programmation par composants », Techniques de l'Ingénieur, n° H 2 759.

[**Rolland a**]
C. Rolland, « Application d'une méthode de conception orientée objet et événement », Techniques de l'Ingénieur, n° H 3 258.

[**Rolland b**]
C. Rolland, « Conception de bases de données : une méthode orientée objet et événement », Techniques de l'Ingénieur, n° H 3 248.

[**Roux a**]
F.-G. Roux, « Estimation de projets logiciels — Méthodes algorithmiques », Techniques de l'Ingénieur, n° H 5 210.

[**Roux b**]
F.-G. Roux, « Mise au point des logiciels », Techniques de l'Ingénieur, n° H 5 250.

⋆ [**Rumbaugh *et al.* 97**]
J. Rumbaugh, M. Blaha, F. Eddy, W. Premerlani & W. Lorensen, *OMT — Modélisation et conception orientée objet*, Masson, Paris, 1997. {Traduction de *Object-Oriented Modelling and Design*, mêmes auteurs, Prentice-Hall, 1991}

⋆ [**Sauvage 02a**]
J.-M. Sauvage, « How To Operate with Strong Emergence Concept: from psycho-cognitive system to social system », *in* 5^{th} *European Conference on Systems Science*, 16–19 octobre 2002.

⋆ [**Sauvage 97**]
S. Sauvage, *Un modèle organisationnel multi-agent pour la simulation de systèmes complexes*, mémoire de DEA, université de Caen, septembre 1997.

⋆ [**Sauvage 01a**]
S. Sauvage, « Conception de SMA : le *pattern Marques* », *in* [Guessoum & Occello 01], pages 47–57.

⋆ [**Sauvage 01b**]
S. Sauvage, « Des *patterns* pour la conception de SMA », *in* [El Fallah Seghrouchni & Magnin 01], pages 149–162.

⋆ [**Sauvage 01c**]
S. Sauvage, « Patterns orientés SMA : le *pattern Marques* », *in* [Ghédira 01], pages 176–187.

⋆ [**Sauvage 02b**]
S. Sauvage, « MAS Oriented Patterns », *in* [Dunin-Kęplicz & Nawarecki 02], pages 283–292.

[**Scheutz & Logan 01**]
M. Scheutz & B. Logan, « Affective vs. Deliberative Agent Control », *in Symposium on Emotion, Cognition, and Affective Computing — AISB'01 Convention*, mars 21–24 2001.

⋆ [**Searle 69**]
J. L. Searle, *Speech Acts*, Cambridge University Press, 1969.

⋆ [**Shakespeare 82**]
W. Shakespeare, *The Illustrated Stratford* Shakespeare, Chancellor Press, Londres, 1982. {Œuvres complètes}

⋆ [**Shannon & Weaver 49**]
C. E. Shannon & W. Weaver, *The Mathematical Theory of Communication*, University of Illinois Press, 1949. {Édition de 1998}

⋆ [**Silva & Delgado 98**]
A. Silva & J. Delgado, « The Agent Pattern for Mobile Agent Systems », *in* [EuroPLoP 98].

⋆ [**Silvestre & Verlhac**]
P. Silvestre & D. Verlhac, « Stratégie de conception des systèmes d'information », Techniques de l'Ingénieur, n° H 5 170.

⋆ [**Simon 69**]
H. A. Simon, *Sciences des systèmes. Sciences de l'artificiel*, Dunod, Paris, 1969. {Traduction de J.-L. Le Moigne, édition 1991}

⋆ [**Simon 95**]
H. A. Simon, « Artificial intelligence: an empirical science », *Artificial Intelligence Journal*, n° 77, 95–127, 1995.

⋆ **[Singh *et al.* 99]**
M. P. Singh, A. S. Rao & M. P. Georgeff, « Formal Methods in DAI: Logic-Based Representation and Reasoning », *in* [Weiss 99], chap. 8, pages 331–376.

[Siu 96]
S. Siu, *Openness and Extensibility in Design-Pattern-Based Parallel Programming Systems*, Masters thesis, university of Waterloo, Ontario (Canada), 1996.

[Siu *et al.* 96]
S. Siu, M. De Simone, D. Goswami & A. Singh, « Design Patterns for Parallel Programming », *in Parallel and Distributed Processing Techniques and Applications, PDPTA'96*, San Jose, Californie (USA), août 1996.

⋆ **[Sloman]**
A. Sloman, « The Irrelevance of Turing Machines to AI », À paraître dans un livre publié par Matthias Scheutz.

⋆ **[Sloman & Poli 95]**
A. Sloman & R. Poli, « Sim_agent: A toolkit for exploring agent designs », *in* M. J. Wooldridge, J. P. Müller & M. Tambe (dir.), *Intelligent Agents II — Workshop on Agent Theories, Architectures, and Languages, atal'95*, vol. 1037 de *Lecture Notes in Artificial Intelligence*, pages 392–407. Springer-Verlag, Heidelberg, 1995.

[Soulié 01]
J.-C. Soulié, *Vers une approche multi-environnements pour les agents*, Thèse de doctorat (informatique), université de La Réunion, janvier 2001.

⋆ **[Stinckwich 94]**
S. Stinckwich, *Modèles organisationnels et réflexifs des architectures à objets concurrents. Implémentation en Smalltalk-80*, Thèse de doctorat (informatique), université de Savoie, Chambéry–Annecy, 1994.

[Stratulat 02]
T. Stratulat, *Systèmes d'agents normatifs : concepts et outils logiques*, Thèse de doctorat (informatique), université de Caen, décembre 2002.

⋆ **[Stroustrup 99]**
B. Stroustrup, *Le langage C++*, Référence, CampusPress, Paris, 3e édition, mai 1999, xiii–994 pages. {Traduit de l'américain par Chr. Eberhardt}

⋆ **[Sunyé 99]**
G. Sunyé, *Mise en œuvre de patterns de conception : un outil*, Thèse de doctorat (informatique), université Paris VI, juillet 1999.

[Tambe 95]
M. Tambe, « Recursive Agent and Agent-group Tracking in a Real-Time, Dynamic Environment », *in* V. Lesser (dir.), *First International Conference on Multi-Agent Systems, icmas'95*, San-Francisco, Californie (USA), juin 1995, MIT Press, Cambridge, Massachusetts (USA).

⋆ **[Theilhard de Chardin 55]**
P. Theilhard de Chardin, *Le Phénomène humain*, Points Essais, Seuil, Paris, 1955.

⋆ **[Tolksdorf 98]**
R. Tolksdorf, « Coordination Patterns of Mobile Information Agents », *in* M. Klusch & G. Weiss (dir.), *Cooperative Information Agents II*, vol. 1435 de *Lecture Notes in Artificial Intelligence*, pages 246–261. Springer-Verlag, Heidelberg, 1998.

[Treur & Willems 95]
J. Treur & M. Willems, « Formal Notions for Verification of Knowledge-Based Systems », *in* M. Ayel & M. C. Rousset (dir.), *European Symposium on the Validation*

and Verification of Knowledge-Based Systems, eurovav'95, pages 189–199, Chambéry, 1995.

[**UML 98**]
«UML»'98 Beyond the notation, Mulhouse, essai (École Supérieure des Sciences Appliquées pour l'Ingénieur), université de Haute-Alsace, 3 & 4 juin 1998.

⋆ [**Van Dyke Parunak 97**]
H. Van Dyke Parunak, « "Go to the Ant": Engineering Principles from Natural Multi-Agent Systems », *Annals of Operations Research*, n° 75, 69–101, 1997.

⋆ [**van Welie & Trætteberg 00**]
M. van Welie & H. Trætteberg, « Interaction Patterns in User Interfaces », *in* [Plop 00].

⋆ [**Varela 96**]
F. J. Varela, *Invitations aux sciences cognitives*, Points Sciences, Seuil, Paris, nouvelle édition, février 1996. {Traduction de Pierre Lavoie}

⋆ [**Vercouter *et al.* 00**]
L. Vercouter, P. Beaune & C. Sayettat, « Towards open distributed information systems by the way of a multi-agent conception framework », *in* AAAI *Agents 2000 conference*, Barcelone (Espagne), 3–7 juin 2000.

⋆ [**Wegner 96**]
P. Wegner, « The Paradigm Shift from Algorithms to Interaction », octobre 1996.

⋆ [**Weiß 99**]
G. Weiss (dir.), *Multiagent Systems — A Modern Approach to Distributed Artificial Intelligence*, MIT Press, Cambridge, Massachusetts (USA), 1999.

⋆ [**Wood & DeLoach 00**]
M. F. Wood & S. A. DeLoach, « An Overview of the Multiagent Systems Engineering Methodology », *in First International Workshop on Agent-Oriented Software Engineering,* aose'2000, Limerick (Irlande), 10 juin 2000.

[**Wooldridge 97**]
M. J. Wooldridge, « Agent-Based Software Engineering », IEE *Transactions on Software Engineering*, vol. 1, n° 144, 26–37, février 1997.

[**Wooldridge 98**]
M. J. Wooldridge, « Agents and Software Engineering », AI*IA *Notizie XI*, n° 3, septembre 1998.

[**Wooldridge 00**]
M. J. Wooldridge, « The Computational Complexity of Agent Design Problems », *in Fourth International Conference on MultiAgent Systems,* icmas'2000, Boston, Massachusetts (USA), 10–12 juillet 2000.

[**Wooldridge & Jennings 95**]
M. J. Wooldridge & N. R. Jennings, « Intelligent Agents: Theory and Practice », *Knowledge Engineering Review*, janvier 1995.

[**Wooldridge & Jennings 98**]
M. J. Wooldridge & N. R. Jennings, « Pitfalls of Agent-Oriented Development », *in* P. Sycara & M. J. Wooldridge (dir.), *Agents'98 — Proceedings of the Second International Conference on Autonomous Agents*. ACM Press, mai 1998.

[**Wooldridge *et al.* 99**]
M. J. Wooldridge, N. R. Jennings & D. Kinny, « A Methodology for Agent-Oriented Analysis and Design », *in* AAAI *Agents'99 conference*, 1999.

⋆ [**Wooldridge *et al.* 00**]
M. J. Wooldridge, N. R. Jennings & D. Kinny, « The Gaia Methodology for Agent-Oriented Analysis and Design », *Journal of Autonomous Agents and Multi-Agent Systems*, 2000.

[**Yim *et al.* 00**]
H. Yim, K. Cho, J. Kim & S. Park, « Architecture-Centric Object-Oriented Design Method for Multi-Agent Systems », *in Fourth International Conference on MultiAgent Systems, icmas'2000*, pages 469–470, Boston, Massachusetts (USA), 10–12 juillet 2000.

[**Yonezawa *et al.* 86**]
A. Yonezawa, J.-P. Briot & E. Shibayama, « Object-Oriented Concurrent Programming in ABCL/1 », *Sigplan Notices*, N. Meyrowitz (dir.), vol. 21, n° 11, 258–268, novembre 1986. {Numéro spécial, Conference on Object-Oriented Programming Systems, Languages and Applications, oopsla'86}

[**Yoo & Briot 01**]
M.-J. Yoo & J.-P. Briot, « Une approche componentielle pour la modélisation d'agents mobiles coopérants », rapport technique 2001.013, lip6, université Paris VI, septembre 2001.

[**Yoo *et al.* 98**]
M.-J. Yoo, J.-P. Briot & J. Ferber, « Using Components for Modeling Intelligent and Collaborative Mobile Agents », *in* D. S. Milojicic (dir.), *IEEE Seventh International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, wetice'98*, Stanford University, Californie (USA), 1998.

⋆ [**Zambonelli *et al.* 00a**]
F. Zambonelli, N. R. Jennings, A. Omicini & M. J. Wooldridge, « Agent-Oriented Software Engineering for Internet Applications », *in* A. Omicini, F. Zambonelli, M. Klusch & R. Tolksdorf (dir.), *Coordination of Internet Agents: Models, Technologies and Applications*, chap. 13, Springer-Verlag, Heidelberg, 2000.

⋆ [**Zambonelli *et al.* 00b**]
F. Zambonelli, N. R. Jennings & M. J. Wooldridge, « Organisational Abstractions for the Analysis and Design of Multi-Agent Systems », *in First International Workshop on Agent-Oriented Software Engineering, aose'2000*, Limerick (Irlande), 10 juin 2000.

328 entrées au total (204 références).

Table des figures

Table des matières

Index

Le lecteur peut ici trouver les références faites aux motifs de conception, les citations des différents auteurs, les différents exemples (plates-formes, méthodes, etc.) et l'emplacement des définitions des divers sigles. Les numéros de pages où l'on peut trouver une définition ou un passage important sont en gras. Les chiffres romains sont les numéros de pages de l'avant-propos.

IMPRIMATUR POUR LA THESE

Conception de systèmes multi-agents: un thésaurus de motifs orientés agent

de M. Sylvain SAUVAGE

REALISEE EN COTUTELLE AVEC

L'UNIVERSITE DE CAEN/BASSE NORMANDIE F

et

L' UNIVERSITE DE NEUCHATEL CH

FACULTE DES SCIENCES

La Faculté des sciences de l'Université de Neuchâtel, sur le rapport des membres du jury

Mmes A. Nicolle (co-directrice de thèse, Caen F),
D. Boulanger (Lyon F),
MM. P.-J. Erard (Neuchâtel)
et J.-P. Müller (co-directeur de thèse, Neuchâtel)

autorise l'impression de la présente thèse.

Neuchâtel, le 24 novembre 2003

La doyenne:

Martine Rahier

Résumé

Capitaliser et diffuser l'expérience sur les systèmes multi-agents sont deux mécanismes clefs que le classique développement par méthodes et outils ne permet pas d'intégrer. Notre hypothèse est que, parmi les techniques existantes collectant et formalisant l'expérience, les motifs *(patterns)* sont la plus apte à rendre compte des concepts agent et à s'adapter aux problèmes variés du développement de SMA.

Nous présentons plusieurs motifs orientés agent qui abordent toutes les phases du développement, depuis l'analyse jusqu'à la mise en œuvre, y compris la reconception (antimotifs). Nos motifs exposent différents niveaux d'abstraction, comme les métamodèles ou les métaphores, très utilisées par le paradigme agent. Nos motifs sont intégrés dans un thésaurus qui révèle leurs liens.

Par ce thésaurus et au travers de son utilisation sur un exemple, nous montrons que la technique des motifs permet d'expliciter et de diffuser les concepts agent et qu'elle est une aide pragmatique et ouverte.

Title

MultiAgent Systems Design: a Thesaurus of Agent Oriented Design Patterns

Abstract

Capitalising and diffusing experience about multiagent systems are two key mechanisms the classical approach of methods and tools can't address. Our hypothesis is that, among available techniques that collect and formalise experience, design patterns are the most able technique allowing to express the agent concepts and to adapt itself to the various MAS developing problems.

We present several agent oriented patterns which cover all the development stages, from analysis to implementation, including re-engineering (antipatterns). Our patterns show different abstraction levels, such as meta-models or metaphors (which are widely used in the agent paradigm). A thesaurus integrates our patterns and reveals their links.

With this thesaurus, and through an example of its usage, we show that the pattern technique allows explaining and diffusing the agent concepts while being a pragmatic and open help.

Discipline

Informatique, intelligence artificielle.

Mots-clefs

Intelligence Artificielle Répartie — Génie Logiciel — Systèmes, Analyse de — Systèmes, Conception de.

Greyc — CNRS UMR 6072
Université de Caen — campus II
BP 5186 — F–14032 Caen cedex France