Conception par agent orientée compétences.

HABILITATION À DIRIGER DES RECHERCHES

en

Informatique

par

Jean-Christophe ROUTIER

25 novembre 2005

Jury :

Mireille CLERBOUT, Université de Lille 1, examinatrice

Guillaume DEFFUANT, CEMAGREF, rapporteur

Jacques FERBER, Université de Montpellier 2, rapporteur

Michel OCCELLO, Université de Grenoble 2, rapporteur

Christophe KOLSKI, Université de Valenciennes, examinateur

Philippe MATHIEU, Université de Lille 1, examinateur

UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE
LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE - UMR CNRS 8022
Cité Scientifique 59655 Villeneuve d'Ascq cedex

à mes parents

Le Jury

Rapporteurs

Guillaume DEFFUANT *Chercheur, Habilité*	Cemagref - LISC 24, rue des landais BP 50085 63172 Aubière Cedex.
Jacques FERBER *Professeur*	LIRMM – Université Montpellier II 161, rue ADA 34392 Montpellier Cedex 5
Michel OCCELLO *Professeur*	IUT de Valence – UPMF Département Informatique 51 rue Barthelemy de Laffemas BP 29 26901 Valence cedex 9

Examinateurs

Mireille CLERBOUT *Professeur*	LIFL – Université des Sciences et Technologies de Lille Bât M3, Cité Scientifique 59655 Villeneuve d'Ascq cedex
Christophe KOLSKI *Professeur*	LAMIH - Université de Valenciennes et du Hainaut Cambrésis Le Mont Houy 59313 Valenciennes cedex 9
Philippe MATHIEU *Professeur*	LIFL – Université des Sciences et Technologies de Lille Bât M3, Cité Scientifique 59655 Villeneuve d'Ascq cedex

Remerciements

Le travail que je vais présenter dans ce document est une synthèse de mes activités de recherche au cours de ces dernières années. Bien évidemment un tel travail n'est jamais réalisé seul. Tout un environnement y contribue directement ou indirectement. Il est donc certainement impossible de désigner nominativement toutes ces personnes, je tiens cependant à plus particulièrement adresser mes remerciements à un certain nombre d'entre elles.

En premier lieu, j'adresse mes sincères remerciements aux rapporteurs, Guillaume DEFFUANT, Jacques FERBER et Michel OCCELLO qui m'ont fait l'honneur d'accepter d'évaluer mon travail. La rédaction d'un rapport est un travail difficile, conséquent et important. Je les remercie donc sincèrement de m'avoir consacrer une partie de leur emploi du temps chargé ainsi que d'avoir jugé favorablement mon travail.

Un grand merci également aux examinateurs, Mireille CLERBOUT et Christophe KOLSKI pour l'intérêt qu'ils ont porté à mon travail et de l'honneur qu'ils m'ont fait en acceptant de participer à mon jury.

J'adresse des remerciements particuliers à Philippe MATHIEU envers qui mon estime dépasse largement son statut directeur de recherches. En m'acceptant au sein de l'équipe SMAC qu'il venait de créer il a évidemment eu un impact très fort sur ces travaux. Par son remarquable travail de direction, il facilite énormément les activités des différents agents de l'équipe et le mien particulier, nous permettant d'être aussi bien réactifs que proactifs, tout en restant aussi cognitifs que possible. Nos points de vue sur le métier d'enseignant-chercheur en général convergent le plus souvent, ce qui ne nous empêche pas d'avoir des discussions passionnées sur ce sujet. Je lui adresse donc à nouveau de très sincères et amicaux remerciements.

Les étudiants ayant participé à ces recherches sont également à remercier. Je citerai particulièrement Yann SECQ, maintenant maître de conférences, avec qui j'espère que nous continuerons d'avoir ces discussions acharnées et passionnantes. Merci également à Patrick TESSIER, Julien ACROUTE et enfin Damien DEVIGNE avec qui je collabore le plus actuellement.

Mes remerciements vont également aux autres agents de l'équipe SMAC : Bruno, Jean-Paul, Sébastien, Julien, Marie-Hélène, Cédric et Laetitia, pour leur bonne humeur et les repas chaleureux du jeudi midi.

Enfin, il est bien évident que je ne pourrai jamais suffisamment remercier mes proches pour ce qu'ils ont fait et font pour moi : mes parents pour leur éducation ; Clélie, Flore et Flavien qui acceptent que je ne leur consacre pas toujours autant de temps qu'ils le souhaiteraient ; et enfin Karine dont la présence calme et patiente est pour moi un soutien inestimable, sans les sacrifices qu'elle accepte, en particulier ces derniers mois, la tâche aurait été autrement difficile, qu'elle en soit donc ici infiniment remerciée.

Table des matières

Préambule

Après une thèse en Programmation Logique, j'ai effectué une reconversion thématique pour travailler ces dernières années dans le domaine des Systèmes Multi-Agents au sein de l'équipe SMAC du LIFL créée en 1998 par le Professeur Philippe Mathieu. Mes travaux m'ont permis d'aborder, et de combiner, les deux thématiques qui me tiennent à cœur : le problème du développement d'applications et l'intelligence artificielle. Au sein de cette équipe, j'ai pu collaborer avec des personnes, et notamment Philippe Mathieu, avec lesquelles je partage une conception de l'activité de recherche, et plus largement du métier d'enseignant-chercheur en général.

Ainsi, dans le cadre de nos recherches, nous voulons proposer des modèles et des concepts mais sans perdre de vue la faisabilité de leur mise en œuvre. Ainsi il ne suffit pas d'avoir un discours sur ce que l'on veut obtenir et d'y apporter des propositions de solutions, il est important selon moi qu'il n'y ait pas de dichotomie avec ce que l'on peut réaliser et donc d'expérimenter ces solutions. Entendons nous bien, il ne s'agit pas de faire de l'ingénierie, fut elle de haut niveau, mais bien de proposer d'abord des nouveaux concepts ou des nouvelles approches et ensuite de les expérimenter et de les valider concrètement.

C'est cette approche que nous avons appliquée dans les deux projets principaux auxquels j'ai participé. Le premier s'appelle MAGIQUE. Il s'inscrit dans la thématique de la programmation d'applications orientée agents. MAGIQUE propose à la fois un modèle organisationnel pour les systèmes multi-agents et un modèle minimal d'agent permettant la construction incrémentale d'agents plus complexes par acquisition dynamique de compétences. Le second est le projet CoCoA. Plus orienté Intelligence Artificielle, ce projet s'intéresse à la simulation de comportements rationnels d'agents évoluant dans un environnement situé. A la différence de MAGIQUE où nous proposons un framework général ne visant pas d'aspect applicatif particulier ni de modèle comportemental des agents, CoCoA propose un moteur de comportements basé sur la notion d'interactions et spécifiquement dédié aux simulations ciblées. Le point commun entre ces différentes recherches est une approche similaire de la construction de nos agents à partir de leurs capacités, que nous appelons *compétences* dans le premier cas et *interactions* dans le second.

Ce document est une synthèse de mon activité de recherche[1] au sein de ces projets. Il se décompose en deux parties. La première est un résumé de mes travaux et la seconde est constituée d'une sélection d'articles sur ces mêmes travaux.

Il n'est pas simple dans une synthèse, notamment lorsqu'elle couvre environ six années de recherche, de choisir le niveau de détail à appliquer. J'ai donc essayé de couvrir l'ensemble de mes travaux en essayant de dégager les idées et concepts que nous avons mis en œuvre ainsi que leurs apports. L'ordre de présentation de ce document ne respecte pas nécessairement exactement

[1] Un résumé de l'ensemble de mes activités d'enseignant-chercheur est présenté en annexe à la fin de ce document.

l'historique des travaux, adapté dans un souci de progression de la présentation. De même tous les aspects de nos travaux ne reçoivent pas nécessairement une place équivalente, j'ai plus particulièrement insisté sur ceux dont la contribution me semblait la plus significative. L'existence de publications sur les différents points présentés devrait permettre au lecteur souhaitant plus de détails, de les y trouver. De plus, la partie consacrée à cette synthèse proposera deux niveaux de granularité.

Le premier chapitre propose en effet un parcours assez rapide de mes activités. J'en profite pour positionner l'ensemble des publications auxquelles j'ai contribuées ainsi que les encadrements de DEA ou thèse que j'ai effectués ces dernières années. Je m'attache également dans ce chapitre à montrer la cohérence qui existe entre les sujets de recherche que j'ai abordés, ainsi que leur complémentarité dans la thématique multi-agent.

Le second chapitre est consacré au projet MAGIQUE. J'y présente plus en détail les concepts que nous avons développés au sein de ce projet : le modèle d'agent minimal et les *compétences*, la structure organisationnelle hiérarchique du système multi-agent avec délégation automatique de requêtes, ainsi que la dynamicité de l'organisation. La mise en œuvre de ces concepts est également présentée, ainsi que la démarche méthodologique RIO pour la conception d'applications par agents. Le mémoire de DEA et la thèse de Yann Secq, soutenue en décembre 2003, s'inscrivent dans cette thématique.

Le troisième chapitre concerne le projet CoCoA. Ce projet propose un modèle pour la conception de simulations par agents situés dans lesquelles des comportements "de type humain" peuvent être simulés. Un champ d'applications naturel d'une telle recherche est le domaine des simulations comportementales dont les jeux vidéos sont un exemple. Notre proposition principale consiste à baser la description des simulations et leurs dynamiques sur la notion d'*interactions*. Dans ce projet nous nous intéressons également à la mise en place de stratégies d'équipe en nous basant sur le même formalisme. Les mémoires de DEA de Patrick Tessier, Damien Devigne et Julien Acroute, ainsi que la thèse en cours de Damien Devigne s'inscrivent dans ce projet.

La seconde partie est constituée d'une sélection de cinq articles, trois concernent le projet MAGIQUE et deux le projet CoCoA. Il a été nécessaire de faire un choix, d'abord sur le nombre de publications à placer dans cette partie et ensuite sur ces publications elles-mêmes. J'ai pris le parti d'un faible nombre de publications en orientant mon choix sur celles qui selon moi illustrent le mieux nos propositions et contributions dans chacune des thématiques. Toutes mes autres publications sont énumérées dans le paragraphe *Bibliographie* du chapitre 1, le lecteur qui le souhaite pourra donc y trouver les autres points abordés dans nos recherches mais non nécessairement présents dans les cinq publications retenues.

J'ai également décidé de laisser le texte des articles tels qu'ils ont été publiés, en langue anglaise en l'occurrence. Seule la forme a été retravaillée dans un souci d'homogénéité de ce document et j'ai donc simplement modifié la feuille de style.

Le chapitre 4 correspond à l'article présenté à la conférence AISB en 2001. C'est dans cet article que nous présentons pour la première fois notre notion d'agent atomique et d'agents construits par acquisition dynamique de compétences.

Le chapitre 5 est l'article que nous avons présenté à la conférence PRIMA2002. Nous y exposons trois principes à appliquer pour construire des organisations multi-agents dynamiques.

Le chapitre 6 a été retenu pour la conférence CEEMAS 2003. Nous y introduisons la démarche méthodologique RIO, pour "Rôles Interactions et Organisation".

Le chapitre 7 a été présenté à la 19^e conférence européenne sur la modélisation et la simulation, ECSM'05. Nous y proposons notre modèle d'interactions et d'agents pour des simulations par agents situés tels que les jeux vidéos.

Le chapitre 8 correspond à l'article de la conférence CIG'05. Nous y présentons comment notre modèle basé sur les interactions peut être appliqué pour modéliser des comportements d'équipes avec chef dans lesquelles les équipiers gardent une certaine autonomie de décision.

Les travaux que je présenterai dans ce document résultent d'un travail de collaboration : avec les étudiants que j'ai encadrés en DEA ou en thèse, ou avec les membres de l'équipe et en particulier Philippe Mathieu, directeur de l'équipe. C'est pourquoi j'utiliserai par la suite le "nous". La recherche est un travail coopératif : au sein d'une équipe de recherche par des échanges directs ou au sein d'une communauté à travers les publications et différentes rencontres. Il est donc bien souvent difficile de déterminer avec certitude l'exacte paternité d'une idée. Il en reste néanmoins que ces travaux sont exprimés ici selon mon point de vue qui n'engage que moi.

Partie I

Synthèse

Chapitre 1

Introduction

1.1 Thèse et après-thèse

Après avoir obtenu un diplôme d'ingénieur de l'ENSI de Caen, option "Informatique et Intelligence Artificielle", et le DEA "Intelligence Artificielle" de l'Université de Caen en 1990, c'est pendant mon service militaire en tant que VFI[1] que j'ai noué mes premiers contacts avec le LIFL[2] en 1991. J'ai ainsi rejoint pour une thèse financée par une allocation du ministère de l'enseignement supérieur et de la recherche l'équipe Méthéol[3] de ce laboratoire en septembre 1991. Cette équipe était dirigée par le Professeur Jean-Paul Delahaye et c'est sous la responsabilité de Philippe Devienne et Patrick Lebègue que s'est déroulée ma thèse.

Celle-ci concernait l'étude théorique des programmes logiques minimaux non triviaux, c'est-à-dire constitués d'une seule clause binaire récursive, d'un fait et d'un but :

$$\begin{cases} p(fait) \leftarrow \quad . \\ p(gauche) \leftarrow p(droit) \quad . \\ \leftarrow p(but) \quad . \end{cases}$$

où *fait*, *gauche*, *droit* et *but* sont des termes arbitraires. Au cours de ma thèse, nous avons résolu les problèmes, jusque là ouverts, de terminaison (Devienne, Lebègue, and Routier 1992b, Devienne, Lebègue, and Routier 1992a, Devienne, Lebègue, and Routier 1993b) et de satisfiabilité (Devienne, Lebègue, and Routier 1993a) de ces programmes. Ensuite, nous avons établi que ces programmes avaient la puissance de calcul des machines de Turing (Devienne, Patrick Lebègue, and Würtz 1994).

J'ai soutenu cette thèse (Routier 1994) le 1er février 2004 sous le titre :

Terminaison, Satisfiabilité et Pouvoir Calculatoire d'une Clause de Horn Binaire

J'ai alors été nommé en septembre 1994 à un poste de Maître de Conférences à l'UFR d'IEEA de l'Université des Sciences et Technologies de Lille, restant au sein de l'équipe Méthéol et du LIFL.

Mon activité de recherche est alors entrée dans une période délicate. Plusieurs facteurs ont contribué à son ralentissement. D'abord la charge naturelle liée aux tâches d'enseignement

[1] Volontaire Formateur Informatique
[2] Laboratoire d'Informatique Fondamentale de Lille
[3] Méthode et outils théoriques pour la programmation Logique

de tout nouvel enseignant-chercheur handicape bien souvent les premières années en poste. Il en est de même des charges administratives. Ma nomination a coïncidé à la mise en place du DEUG MIAS à l'Université de Lille 1, il fallait construire un nouvel enseignement et coordonner les équipes enseignantes, ce dont je me suis chargé avec Eric Wegrzynowski. Ce travail a débouché sur la rédaction d'un livre (Routier and Wegrzynowski 1997 - 2003 (2nde édition)) d'enseignement qui a naturellement monopolisé beaucoup de mon temps pendant une année. Même si je considère que la diffusion de connaissance et la rédaction de tels ouvrages est un des rôles des enseignants du supérieur, et si je ne regrette en rien l'expérience vécue à l'occasion de ce travail, il n'en reste pas moins que cela se fait au détriment de la recherche. Ensuite, les résultats de ma thèse offraient peu de perspectives de poursuite puisqu'ils concluaient sur des propriétés d'indécidabilité, me privant ainsi d'une dynamique immédiatement issue du travail de thèse. Cette période s'est soldée par la publication de (Devienne, Lebègue, Parrain, and Würtz 1996). Enfin, la dissolution de l'équipe de recherche à laquelle j'appartenais m'a obligé à une reconversion thématique. C'est donc ce que j'ai entrepris en 1998 en rejoignant l'équipe SMAC, "Systèmes Multi-Agents et Comportements", que venait de créer le Professeur Philippe Mathieu, également issu de l'équipe Méthéol. Je suis à l'heure actuelle toujours membre de cette équipe. Mon travail s'est depuis cette époque essentiellement réparti sur deux thématiques : d'abord le développement d'applications multi-agents physiquement distribués et ensuite la simulation de comportements rationnels à base d'agents cognitifs situés.

1.2 MAGIQUE, Rio et Cocoa

Lorsque je l'ai rejointe, un axe de travail de l'équipe SMAC portait sur le modèle d'organisation de systèmes multi-agents MAGIQUE (pour Multi-AGent hiérarchIQUE), notamment dans le cadre de la thèse de Nour-Eddine Bensaïd. Je me suis en particulier attelé à la réalisation d'une API JAVA mettant en œuvre les concepts de MAGIQUE (Mathieu and Routier 2000-2001). Ce travail a amené une réflexion sur la notion de programmation orientée agents (*Agent Oriented Software Engineering*). Une première étape a concerné l'étude de la notion de services dans les systèmes multi-agents notamment dans le cadre du mémoire de DEA de Yann Secq (Secq 1999). Ces travaux ont amené à un enrichissement du modèle MAGIQUE auquel, outre le modèle organisationnel hiérarchique existant, nous avons ajouté un modèle d'agent. En effet notre réflexion sur la notion de services nous a conduits à envisager la construction d'un agent par un enrichissement dynamique et incrémental de *compétences* qui lui permettent de rendre des *services*. Nous avons ainsi établi qu'il était possible d'obtenir n'importe quel agent à partir d'un agent élémentaire, ou minimal, simplement doté de deux compétences initiales : une compétence de communication sans laquelle il serait autiste et une compétence d'acquisition/administration de ses compétences lui permettant d'évoluer (Routier, Mathieu, and Secq 2001).

Dans le même temps nous avons étudié la dynamicité de l'organisation du système multi-agents en énonçant et expérimentant différents principes (Mathieu, Routier, and Secq 2002). Ceux-ci s'appuient à la fois sur l'architecture de MAGIQUE, dont le principal avantage est de fournir un mécanisme de délégation de services par défaut, et sur l'évolution dynamique des agents. Nous obtenons ainsi des systèmes multi-agents souples et dynamiques tant du point de vue structurel que du point de vue individuel. Ces avantages renforcent à la fois la facilité de conception des applications en soulageant le concepteur d'un certain nombre de préoccupations, et la fiabilité du système en permettant une réorganisation dynamique de celui-ci. Nous avons appliqué ces

différents principes et notre approche orientée agent de la conception d'applications à différents domaines, tels que le calcul distribué (Mathieu, Routier, and Secq 2002c, Mathieu, Routier, and Secq 2002d) et une application de travail co-opératif (Mathieu and Routier 2001, Mathieu and Routier 2002). Ces expériences nous ont permis de faire progresser notre réflexion sur une démarche méthodologique de conception des applications à base d'agents.

Nous avons ainsi proposé la méthode RIO pour "Rôles, Interactions et Organisations" (Mathieu, Routier, and Secq 2002a, Mathieu, Routier, and Secq 2003c, Mathieu, Routier, and Secq 2003e, Mathieu, Routier, and Secq 2003a) qui a constitué le cœur de la thèse de Yann Secq, que j'ai co-encadrée avec le Professeur Philipe Mathieu, et qu'il a soutenue en décembre 2003 (Secq 2 décembre 2003). Cette méthode propose un formalisme graphique de représentation des protocoles d'interaction qui en constitue une spécification exécutable (Mathieu, Routier, and Secq 2003a, Mathieu, Routier, and Secq 2003b). On y exprime notamment les rôles impliqués dans ces interactions, appelés *micro-rôles*, ainsi que les messages échangés entre les micro-rôles participants. Les micro-rôles sont ensuite regroupés pour former des *rôles composites* qui peuvent finalement être attribués aux agents effectivement impliqués dans l'application et qui sont intégrés à l'organisation choisie.

La seconde thématique que j'ai abordée concerne la simulation de comportement pour agents situés. Ce projet s'appelle COCOA pour "Cognitive Collaborative Agents". Il a été amorcé à l'occasion d'une coopération avec la société de jeu Cryo Interactive dans le cadre d'un projet PRIAMM[4] qui s'est déroulé de juin 2000 à janvier 2001 (Mathieu, Routier, and Urro 2001). Le premier objectif de cette recherche est de proposer un cadre permettant de réaliser des simulations centrées individus nécessitant la mise en œuvre de comportements rationnels. L'article (Mathieu, Picault, and Routier 2005) publié dans la revue "*Pour la Science*" présente une synthèse rapide de cette problématique. Le second objectif est la réalisation d'un environnement destiné aux Game Designers afin de les aider dans la conception de comportements évolués. Les jeux vidéos constituent, avec la présence de personnages virtuels qui doivent manifester des comportements réalistes, un domaine d'application et un contexte d'expérimentation tout indiqué. Cependant l'approche utilisée dans les jeux actuels est, malgré ses nombreux défauts, essentiellement une approche réactive basée le plus généralement sur des scripts. Au contraire, dans notre recherche nous considérons des agents cognitifs motivés par des buts, capables de raisonner sur leurs connaissances et aptes à tenir compte du contexte pour tenter de réaliser ces buts. A cette fin l'agent dispose d'un moteur de comportement qui lui permet de déterminer l'action à effectuer afin de remplir au mieux ses objectifs. Ce moteur établit un plan d'actions en fonction des connaissances de l'agent et notamment de sa situation dans l'environnement (Devigne, Mathieu, and Routier 2004). On se place donc ici dans le cadre d'applications multi-agents spatialement situées. Les agents sont positionnés et évoluent dans un environnement muni d'un espace euclidien dont ils ont une vision partielle. Cet environnement est intrinsèquement dynamique et non monotone.

La principale contribution de ce travail consiste en une approche orientée *interaction* de la modélisation de simulations. Les interactions sont des éléments de connaissance qui définissent les règles qui régissent le monde simulé. Elles spécifient dans quel contexte une action peut être réalisée et ses conséquences. Les agents impliqués dans la simulation sont caractérisés par les interactions qu'ils peuvent subir, et pour les agents *animés* par les interactions qu'ils peuvent effectuer. La dynamique des simulations est basée sur cette dualité entre les agents qui peuvent effectuer une action et d'autres qui peuvent la subir (Mathieu, Picault, and Routier 2003, Devigne,

[4] Programme pour l'Innovation dans l'Audiovisuel et le Multimédia

Mathieu, and Routier 2005b). Cette thématique est plus orientée “Intelligence Artificielle”. Nous y proposons un modèle d’agent situé cognitif. Cela nous amène à être confrontés à de nombreux problèmes de l’IA : représentation de connaissance, planification, sélection d’actions, etc. Le regroupement de ces problèmes constitue une difficulté supplémentaire.

Ce travail a notamment a été développé à travers plusieurs mémoires de DEA que j’ai co-encadrés (Tessier 2002, Devigne 2003, Acroute 2005) ainsi que la thèse en cours de Damien Devigne (Devigne 2003-en cours). Dans le cadre de cette thèse, nous considérons également la modélisation de comportements d’équipes (Devigne, Mathieu, and Routier 2005c, Devigne, Mathieu, and Routier 2005a) se basant sur le même concept d’interaction. Nous avons notamment proposé une construction de plans d’équipe avec chef laissant une part d’autonomie aux différents équipiers.

Ces travaux m’ont amené à participer à différents groupes de travail et projets. Au niveau national je participe aux groupes du GdR I3 (“Information - Interaction - Intelligence”) ASA (Architecture des Systèmes multi-Agents) et MFI (Modèles Formels de l’Interaction). J’ai également participé aux projets de CPER (contrat de plan état-région) successifs dans lesquels l’équipe était ou est actuellement impliquée : d’abord le projet *Ganymède* puis ses successeurs *COLORS* (“Composants Logiciels Réutilisables et Sûrs”), *NIPO* (“Nouvelles Interactions Personnes-Organisations”) et *Formasciences* sur les nouveaux usages, et actuellement *MIAOU* (“Modèles d’Interaction et Architectures Orientées Usages”) et *MOSAIQUES* (“MOdèles et InfraStructures pour Applications ubIQUitairES”). J’ai également participé à un projet PRIAMM en collaboration avec la société Cryo Interactive.

1.3 Cohérence et complémentarité

Comme présentés dans le paragraphe précédent, mes travaux au sein de l’équipe SMAC concernent essentiellement deux thématiques : d’abord les plateformes multi-agents et la conception d’applications distribuées par une approche multi-agent, avec le projet MAGIQUE, ensuite la simulation de comportements rationnels à base d’agents à travers le projet CoCoA. Les points communs de ces recherches dépassent, évidemment, le simple terme *agent*. Les approches des problèmes sont similaires. En même temps, les thèmes se complètent pour couvrir de multiples problématiques du multi-agent.

1.3.1 Cohérence de l’approche

La notion centrale dans le projet MAGIQUE est celle de *compétence*. Dans le projet CoCoA, il s’agit des *interactions*. Les principes portés par ces deux notions sont similaires : elles permettent de définir les compétences et rôles des agents. Dans chacun des cas, nous soutenons une approche dans laquelle les construction et définition des différents agents se font à partir d’un noyau commun par attribution de capacités. Dans MAGIQUE, les agents sont construits dynamiquement à partir de notre noyau d’agent minimal par acquisition de compétences. Dans CoCoA, les agents impliqués dans les simulations sont caractérisés par les interactions qu’ils peuvent subir ou effectuer. Les agents animés sont tous dotés du même moteur de comportements et ce sont leurs buts ainsi que les différentes interactions dont ils sont dotés qui les amènent à se comporter différemment.

Dans les deux cas on obtient une construction incrémentale des agents. Les apports de cette approche sont les mêmes. Il s'agit essentiellement d'offrir les avantages de la modularité. Les compétences d'une part et les interactions d'autres part constituent des composants potentiellement réutilisables dans différentes applications. Ils représentent les éléments de connaissances et d'expertise des applications (ou simulations) réalisées. En fait il s'agit des notions centrales de nos approches, les agents n'étant à chaque fois que des réceptacles paramétrés par ces éléments. Compétences et interactions permettent, même implicitement, d'attribuer des rôles aux agents. Dans MAGIQUE c'est parce qu'un agent détient une compétence qu'il pourra se voir déléguer la réalisation d'une requête et jouera ainsi le rôle sous-jacent. Dans CoCoA, c'est parce qu'un agent peut effectuer une interaction qu'il pourra l'utiliser pour réaliser un but et ainsi tenir le rôle implicitement associé à cette interaction.

Si les approches "compétences" et "interactions" constituent la cohérence thématique des projets, les démarches des projets sont également similaires. En particulier, nous tenons à mettre en œuvre concrètement les concepts développés. Il s'agit pour nous d'un élément important que de ne pas se limiter à émettre des idées ou concepts mais de les confronter à la réalité d'une implémentation. Nous refusons donc une démarche qui serait purement théorique et littéraire. Dans les thèmes que nous étudions une telle démarche nous semble aberrante. Cette mise en œuvre concrète oblige à considérer la faisabilité des approches et se confronter à la réalisation de ses idées oblige à considérer les problèmes jusque dans leurs détails révélant des difficultés ou subtilités qui auraient pu ne pas être soulevées dans une étude purement conceptuelle.

Que ce soit avec MAGIQUE ou avec CoCoA nous avons donc développé un contexte reprenant les idées permettant ainsi de les valider expérimentalement. L'objectif est à chaque fois le même. Il s'agit d'offrir un cadre facilitant la réalisation d'applications à base d'agents. Avec MAGIQUE, nous avons réalisé une API[5] permettant de construire des applications multi-agents distribuées. Cette API met en œuvre notre modèle d'agent minimal et propose une organisation par défaut hiérarchique des systèmes multi-agents développés, tout en permettant la mise en place d'autres modèles. Avec CoCoA nous fournissons un environnement de programmation pour la réalisation de simulations centrées individus à base d'agents situés. Cet environnement permet de créer les différents éléments d'une simulation puis, lors de l'exécution de celle-ci, d'examiner l'évolution de différents éléments et notamment celle du moteur de planification des agents animés. A travers ces développements, outre la validation des idées, nous cherchons à mettre en avant la conception d'applications à base d'agents et une approche "agent oriented software engineering". Dans les deux cas nous visons à offrir un contexte générique, un *framework*, pour les applications cibles.

1.3.2 Complémentarité des thèmes

Si, comme nous venons de le voir, il y a une certaine cohérence dans les propositions apportées dans les deux thématiques, des différences existent néanmoins dans les sujets abordés ou leur traitement. Les thèmes abordés sont également complémentaires et couvrent des aspects différents du domaine du multi-agent.

Ainsi avec le projet MAGIQUE, nous nous sommes intéressés à la notion d'architecture et de plateforme multi-agent, cherchant à offrir un contexte de développement d'applications à

[5] Application Programming Interface

base d'agents sans nous intéresser précisément au modèle interne des agents ni aux aspects applicatifs. Avec MAGIQUE nous proposons donc un framework cherchant à faciliter la construction d'applications multi-agents sans nous attacher à un type d'application en particulier. Ce framework offre les outils permettant au concepteur de l'application de se dégager d'un certain nombre de préoccupations telles que la distribution ou les communications entre les agents. Il peut se concentrer sur les aspects applicatifs essentiels. Cependant nous ne proposons pas de modèle de comportement par défaut des agents. Ainsi dans MAGIQUE notre proposition se trouve en amont des préoccupations applicatives et des modèles comportementaux particuliers.

A la différence, dans CoCoA nous nous intéressons à un type d'applications ciblé : les simulations centrées individus par agents situés. Cette fois nous proposons un modèle d'agent particulier : le moteur comportemental des agents animés est fourni. Le travail du concepteur de la simulation consiste ici en la définition des interactions et des propriétés de ces agents. La démarche est ici plus orientée intelligence artificielle que dans MAGIQUE. Pour la mise en œuvre de nos simulations, nous avons nécessairement été confrontés à de nombreux problèmes d'IA qui, sans qu'ils soient les sujets centraux du projet, n'en ont pas moins dû être abordés. Il s'agit notamment de la représentation et de la manipulation de la connaissance et de la planification.

Une autre différence concerne les aspects multi-agents. Avec MAGIQUE nous nous sommes intéressés aux aspects organisationnels en proposant une structure hiérarchique du système multi-agent dotée d'un mécanisme de délégation de réalisation de services. Dans CoCoA, les aspects multi-agents se traduisent dans un premier temps essentiellement par une cohabitation des agents qui évoluent en concurrence dans le même environnement. Dans un second temps, nous nous sommes également intéressés aux comportements d'équipes dirigées ou non par un leader. Ainsi dans MAGIQUE le multi-agent est abordé du point de vue organisationnel alors que le point de vue est plus comportemental dans CoCoA.

Enfin, d'autres différences existent entre les problématiques de ces projets : dans CoCoA les agents sont situés ce qui n'est a priori pas le contexte des applications développées avec MAGIQUE. Cela implique également que la notion d'environnement doive être explicitement définie dans CoCoA alors qu'elle est implicite avec MAGIQUE. Ensuite, MAGIQUE vise des applications distribuées sur le réseau alors que les simulations construites avec CoCoA sont destinées à tourner "en local". Le parallélisme de l'exécution des différents agents est donc géré par du multi-threading dans MAGIQUE alors qu'il n'est que simulé dans CoCoA.

1.4 Conclusion

Ainsi à travers ces deux projets principaux, un large éventail des facettes du domaine du multi-agent a été abordé et étudié. La philosophie de base commune appliquée dans chacun des projets est de considérer que, plus que les agents, ce sont les capacités qu'ils portent qui sont importantes. Notre principe de base est donc de construire ces agents en leur affectant des compétences à partir d'une base commune. Dans le cadre de MAGIQUE, il s'agit de partir d'un noyau minimal élémentaire et de l'enrichir dynamiquement. Dans CoCoA, le modèle comportemental fourni est alimenté par l'attribution des interactions que l'agent peut effectuer. Dans le premier cas, les agents évoluent au sein d'une organisation multi-agent à la base hiérarchique permettant de construire des applications distribuées. Dans le second, les agents sont situés dans un

environnement et permettent la conception de simulations centrées individus où des comportements complexes peuvent être mis en œuvre.

Pour moi, la trame principale de ces recherches est la promotion d'une conception par agents favorisant une vision incrémentale de la construction des agents et plaçant au centre la notion de compétences. Cette approche est symbolisée par notre modèle d'agent minimal d'une part et l'approche orientée interaction des simulations d'autre part. Ce point de vue est complété par une réflexion sur les notions d'organisation multi-agent dans le cadre de MAGIQUE, et par une étude de la modélisation de comportements par agents situés dans le cadre de CoCoA.

A chaque fois, nous nous sommes attachés à mettre en œuvre ces aspects conceptuels de manière concrète : l'API MAGIQUE dans un cas et un environnement de conception de simulations dans l'autre. Ces réalisations nous ont permis de donner corps à nos idées et d'évaluer leur faisabilité et leur pertinence.

Actuellement, c'est au projet CoCoA que je consacre mes activités de recherche. Le travail en cours concerne l'amélioration du modèle comportemental de l'agent ainsi que les stratégies d'équipe. Un des objectifs à court terme est également la réalisation d'une simulation d'envergure valorisant ce modèle comportemental.

Bibliographie

Acroute, J.. 2005. "Apport réactif aux comportements cognitifs de la plateforme de simulation cocoA," Master's thesis, Master Recherche mention Informatique de l'Université de Lille 1, co-dirigé avec le Professeur Philippe Mathieu.

Devienne, P.; P. Lebègue; A. Parrain; and J.-C. R. J. Würtz. 1996. "Smallest Horn clause programs," *Journal of Logic Programming*, 27(3).

Devienne, P.; P. Lebègue; and J.-C. Routier. 1992a. "The halting problem of one binary Horn clause is undecidable," in *proceedings of "Workshop on termination" à l'occasion de JICSLP'92*, 5–14.

Devienne, P.; P. Lebègue; and J.-C. Routier. 1992b. "Weighted Sytems of Equations revisited," in *proceedings of Workshop on Static Aanalysis WSA'92*, 163–173.

Devienne, P.; P. Lebègue; and J.-C. Routier. 1993a. "The emptiness problem of one binary Horn clause is undecidable," in *Proceedings of 1993 International Symposium on Logic Programming, ILPS'93*, ed. by D. Miller, 250–265. MIT Press.

Devienne, P.; P. Lebègue; and J.-C. Routier. 1993b. "The halting problem of one binary Horn clause is undecidable," in *Proceedings of STACS'93*, ed. by P. Enjalbert, A. Finkel, and K. Wagner, no. 665 in LNCS, 48–57. Springer-Verlag.

Devienne, P.; J.-C. R. Patrick Lebègue; and J. Würtz. 1994. "One Binary Horn Clause is Enough," in *Proceedings of STACS'94*, ed. by P. Enjalbert, E. Mayr, and K. Wagner, no. 775 in LNCS, 21–32. Springer-Verlag.

Devigne, D.. 2003. "Simulation de comportements pour agents rationnels situés et étude du GraphPlan," Master's thesis, DEA d'Informatique de l'Université de Lille 1, co-dirigé avec le Professeur Philippe Mathieu.

Devigne, D.. 2003-en cours. “Modélisation de comportement d’équipes en environnement Multi-Agents situés,” Ph.D. thesis, Université des Sciences et Technologies de Lille, co-dirigée avec le Professeur Philippe Mathieu.

Devigne, D.; P. Mathieu; and J.-C. Routier. 2004. “Planning for Spatially Situated Agents,” in *Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT’04)*, ed. by I. Press, 385–388.

Devigne, D.; P. Mathieu; and J.-C. Routier. 2005a. “Gestion d’équipes et autonomie des agents,” in *actes des JFSMA2005*, ed. by Cépaduès.

Devigne, D.; P. Mathieu; and J.-C. Routier. 2005b. “Interaction-Based Approach For Game Agents,” in *Proceedings of ECMS/SCS/IEEE 19th European Conference on Modelling and Simulation. ECMS 2005*, ed. by Y. Merkuryev, R. Zobel, and E. Kerckhoffs, 7056714.

Devigne, D.; P. Mathieu; and J.-C. Routier. 2005c. “Teams of cognitive agents with leader: how to let them some autonomy,” in *Proceedings of IEEE Symposium on Computational Intelligence Games. CIG’05*, ed. by G. Kendall, and S. Lucas, 256–262.

Mathieu, P.; S. Picault; and J.-C. Routier. 2003. “Simulation de comportements pour agents rationnels situés,” in *Actes des Secondes Journées Francophones sur les Modèles Formels de l’Interaction, MFI’03*, ed. by A. Herzig, B. Chaib-draa, and P. Mathieu, 277–282.

Mathieu, P.; S. Picault; and J.-C. Routier. 2005. “Les agents intelligents,” *Pour La Science*, (332), 44–52.

Mathieu, P.; J. Routier; and Y. Secq. 2002a. “Dynamic Organization of Multi-Agent Systems,” in *Proceedings of the AAMAS’02 Conference*, 451–452.

Mathieu, P.; and J.-C. Routier. 2000-2001. “Tutoriel de Magique,” Equipe SMAC - LIFL.

Mathieu, P.; and J.-C. Routier. 2001. “Une contribution du multi-agent aux applications de travail coopératif,” *TSI Hermès Science Publication. Réseaux et Systèmes Répartis. Calculateurs Parallèles.*, 13. Numéro spécial télé-applications, 207–226.

Mathieu, P.; and J.-C. Routier. 2002. “A Multi-Agent Approach to Co-operative Work,” in *Proceedings of CADUI’02*, ed. by C. Kolski, and J. Vanderdonckt, 367–380. Kluwer Academics Press.

Mathieu, P.; J.-C. Routier; and Y. Secq. 2001. “Dynamic Skill Learning: A Support to Agent Evolution,” in *Proceedings of AISB’01*, 25–32.

Mathieu, P.; J.-C. Routier; and Y. Secq. 2002b. “Principles for dynamic multi-agent organizations,” in *Proceedings of 5th Pacific Rim International Workshop on Multi-Agents. PRICAI2002-PRIMA2002*, ed. by K. Kuwabara, and J. Lee, vol. 2413 of *LNAI*, 109–122. Springer-Verlag.

Mathieu, P.; J.-C. Routier; and Y. Secq. 2002c. “RAGE: An agent framework for easy distributed computing,” in *Proceedings of AISB’02*, 20–24.

Mathieu, P.; J.-C. Routier; and Y. Secq. 2002d. “Using agents to build a distributed calculus framework,” *The Interdisciplinary Journal of Artificial Intelligence and the Simulation of Behaviour*, 1(2), 197–208.

Mathieu, P.; J.-C. Routier; and Y. Secq. 2003a. "Bridging the gap between semantic and pragmatic," in *Proceedings of The 2003 International Conference on Information and Knowledge Engineering, IKE'03*, ed. by H. Arabnia, vol. I, 308–314.

Mathieu, P.; J.-C. Routier; and Y. Secq. 2003b. "RIO: Rôles, Interactions et Organisations," in *Actes des Secondes Journées Francophones sur les Modèles Formels de l'Interaction, MFI'03*, ed. by A. Herzig, B. Chaib-draa, and P. Mathieu, 179–188.

Mathieu, P.; J.-C. Routier; and Y. Secq. 2003c. "Runnable specifications of interactions protocols for open multi-agent systems," in *Proceedings of the 2003 International Conference on Information and Knowledge Engineering, IKE'03*, ed. by N. Goharian, vol. II, 431–437.

Mathieu, P.; J.-C. Routier; and Y. Secq. 2003d. "RIO : Roles, Interactions and Organizations," in *Proceedings of the 3rd International/Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003*, ed. by V. Marik, J. Müller, and M. Pechoucek, 147–157.

Mathieu, P.; J.-C. Routier; and Y. Secq. 2003e. "Towards a Pragmatic Methodology for Open Multi-agent Systems," in *Proceedings of 14th International Symposium on Methodologies for Intelligent Systems, ISMIS 2003*, ed. by N. Zhong, Z. W. Raś, S. Tsumoto, and E. Suzuki, no. 2871 in LNAI, 206–210. Springer-Verlag.

Mathieu, P.; J.-C. Routier; and P. Urro. 2001. "Un modèle de simulation agent basé sur les interactions," in *Actes des Secondes Journées Francophones sur les Modèles Formels de l'Interaction, MFI'01*.

Routier, J.-C.. 1994. "Terminaison, Satisfiabilité et Pouvoir Calculatoire d'une Clause de Horn Binaire," Ph.D. thesis, Université de Lille 1.

Routier, J.-C.; and E. Wegrzynowski. 1997 - 2003 (2nde édition). *Débuter la Programmation avec* SCHEME. International Thomson Publishing France.

Secq, Y.. 1999. "Notion de service et d'échange de services dans les systèmes multi-agents," Master's thesis, DEA d'Informatique de l'Université de Lille 1, co-dirigé avec le Professeur Philippe Mathieu.

Secq, Y.. 2 décembre 2003. "RIO : Rôles, Interactions et Organisations, une méthodologie pour les systèmes multi-agents ouverts," Ph.D. thesis, Université des Sciences et Technologies de Lille, co-dirigée avec le Professeur Philippe Mathieu.

Tessier, P.. 2002. "Planification et exécution dans un environnement non monotone," Master's thesis, DEA d'Informatique de l'Université de Lille 1, co-dirigé avec Professeur Philippe Mathieu.

Chapitre 2

De MAGIQUE à RIO

Le premier projet auquel j'ai contribué au sein de l'équipe SMAC est le projet MAGIQUE. Le point de départ de ce projet est la proposition par Philippe Mathieu d'une organisation hiérarchique de systèmes multi-agents, MAGIQUE signifiant "Multi AGent hiérarchIQUE" (Bensaid and Mathieu 1997, Bensaid and Mathieu 1997b, Bensaid 1999). Ce projet a évolué pour proposer un modèle d'agent minimal, développer les aspects dynamiques des organisations multi-agents et, enfin, amener à la démarche méthodologique RIO basée sur les notions de rôles, d'interactions et d'organisation.

Ces recherches rejoignent les travaux sur les modèles organisationnels, les plate-formes multi-agents et les méthodes de développement d'applications à l'aide de systèmes multi-agents. Cette thématique correspondait notamment aux préoccupations du groupe ASA, pour "Architecture des Systèmes multi-Agents", du GdR I3. Plus largement le projet MAGIQUE s'inscrit dans le thème "Agent Oriented Software Engineering" (Jennings and Wooldridge 2000, Occello and Koning 2000, Petrie 2001) puisque l'objectif principal du projet est de fournir un contexte facilitant le développement d'applications grâce aux agents.

Les notions développées dans le projet MAGIQUE s'intègrent ainsi à différentes autres études :

- celles menées sur les organisations : à base de groupes (Ferber and Gutknecht 1998a), holonique (Gerber, Siekmann, and Vierke 1999, Adam, Mandiau, and Kolski 2001) ou le pair à pair par exemple.
- celles menées sur les plate-formes : Madkit (Ferber, Gutknecht, and Michel 2000, J.Ferber, Gutknecht, and Michel 2000) du LIRMM, Volcano (Ricordel and Demazeau 2002), Dima (Guessoum 1996),
- celles sur les méthodes : Voyelles (Ricordel 2001, da Silva and Demazeau 2002), Gaia (Wooldridge, Jennings, and Kinny 2000).

Les paragraphes suivants vont tenter de synthétiser les différents résultats de nos recherches sans nécessairement respecter la progression temporelle. Ainsi si le modèle organisationnel a été le moteur fondateur de MAGIQUE, je présenterai en premier lieu le modèle d'agent minimal, puis l'organisation hiérarchique et ses apports pour enfin m'intéresser à leur mise en œuvre dans l'API MAGIQUE. Je finirai par une brève présentation de RIO.

2.1 Le modèle d'agent minimal

Le caractère "agent" d'une application se situe le plus souvent dans l'esprit du programmeur (Shoham 1993, Travers 1996). De ce fait, de nombreuses définitions du terme *agent* sont apparues (Franklin and Grasser 1996), et il n'est pas difficile de regrouper un grand nombre de définitions différentes. L'intersection entre toutes ces définitions n'est le plus souvent pas vide : elles diffèrent, parfois très légèrement, à cause de quelques fonctionnalités considérées comme basiques pour chacun des modèles. Partant de ce constat, nous avons essayé de proposer une base sur laquelle pourraient s'appuyer toutes les autres définitions. Notre proposition consiste en la définition d'un *agent atomique* susceptible d'évoluer dans différentes directions pour correspondre aux différentes notions introduites par chacun.

La notion centrale de notre proposition est celle de *compétence*. Elle correspond à un ensemble cohérent de capacités qui peuvent être attribuées à un agent. Le principe est de partir d'un agent atomique et de lui *enseigner* les compétences nécessaires à l'obtention de l'agent recherché. Le résultat dépend des compétences enseignées et il est donc possible d'obtenir différents types d'agents. De plus, l'éducation doit pouvoir être réalisée alors que l'agent est actif. Celui-ci peut donc évoluer dynamiquement.

Du point de vue du programmeur, cette approche favorise la réutilisabilité et la modularité, puisqu'une fois qu'une compétence a été développée, elle peut être exploitée dans différents contextes. Dans cette vision, une compétence peut être considérée comme un composant logiciel (pour une autre approche de la construction d'agents à partir de composants, voir par exemple (Horling and Lesser 1998)).

Ces travaux ont été publié dans (Routier, Mathieu, and Secq 2001).

2.1.1 Définitions

Dans la mesure où un agent est *quelqu'un qui agit*, à partir du moment où l'on convient qu'une *compétence* désigne un "ensemble cohérent de capacités", la définition suivante devrait pouvoir être relativement consensuelle :

Définition 1 *Un* agent *est une entité douée de compétences.*

Toute propriété communément attachée à la notion d'agent – telle que proactivité, interactivité, intelligence, etc. – peut en effet s'exprimer en terme de compétences. Il semble donc raisonnable de dire que toutes les définitions d'agents, comme celle citées dans (Franklin and Grasser 1996), peuvent être obtenues à partir de celle-ci : les différences entre deux définitions d'agent proviennent en effet des fonctionnalités de base exigées des agents, c'est-à-dire de leurs compétences. Ainsi un agent dont le cycle de décision s'appuie sur un réseau de Pétri est un agent qui dispose d'une compétence d'interprétation de ce réseau ; un autre qui peut crypter ses messages est un agent doué d'une compétence de codage/décodage, etc. En se plaçant plus au niveau des concepts, les notions de rôles et de groupes, qui sont au centre du modèle Aalaadin (Ferber and Gutknecht 1998), peuvent également se traduire en termes de compétences et donc ce modèle pourrait être décrit en de tels termes. Cependant, pour cette même raison, la définition est probablement trop vague puisqu'elle permet trop de liberté d'interprétation selon les compétences attachées à l'agent. C'est pourquoi nous allons la préciser en fixant un ensemble minimal de compétences.

Nous affirmons que simplement deux compétences initiales sont nécessaires et suffisantes pour définir cet *agent atomique* à partir duquel toutes les autres définitions d'agent peuvent

être établies. Ces compétences sont : pour la première, une compétence qui permet à l'agent d'apprendre de nouvelles compétences, et pour la seconde, une compétence de communication (avec les autres agents – qui pourraient être humains ou logiciels).

Ces compétences sont en effet *nécessaires*. Sans la "compétence d'acquisition", un tel agent ne serait qu'une coquille vide incapable de faire quoique ce soit. Sans la "compétence de communication", un agent est isolé du "reste du monde" et perd de ce fait tout intérêt. De plus la communication est nécessaire à l'acquisition de nouvelles compétences.

Et elles sont *suffisantes* puisqu'il suffit à un agent d'utiliser sa compétence d'interaction pour entrer en contact avec un agent compétent et d'utiliser sa compétence d'acquisition pour apprendre de nouveaux talents auprès de celui-ci. Ainsi, n'importe quelle capacité peut être "donnée" à un agent par apprentissage auprès d'un "enseignant".

En conséquence nous proposons la définition d'agent suivante :

Définition 2 *Un* agent atomique *est une entité douée de deux compétences : une pour interagir et une pour apprendre de nouvelles compétences. Un* agent *est un agent atomique qui a appris des compétences au travers de communications.*

Nous soutenons que les types d'agents proposés dans les différentes définitions existantes s'intègrent dans cette définition.

Notons que ce ne sont pas les compétences elles-mêmes qui sont importantes mais plutôt les fonctionnalités qu'elles représentent. Ainsi on peut imaginer que la compétence de communication utilisée par un agent évolue au cours de son cycle de vie, parce qu'il en a appris une nouvelle par exemple. Ce qui est important ce n'est pas qu'il ait telle compétence d'interaction particulière, mais plutôt qu'il ait toujours la capacité de communiquer (et il en est de même avec la compétence d'apprentissage).

2.1.2 Compétences

Les compétences constituent la substantifique moelle des agents dans notre vision. Une compétence correspond à un ensemble de fonctionnalités qui peuvent être exploitées par un agent. D'un point de vue plus pragmatique, une compétence doit être vue comme un composant dont l'interface publique désigne les capacités qu'un autre agent peut exploiter.

La granularité et le degré de complexité d'une compétence ne peuvent pas être définitivement énoncés. Les capacités d'analyser un message XML ou d'additionner deux entiers peuvent chacune représenter une compétence bien que leurs complexités soient très probablement considérées comme se situant à des niveaux différents. De plus, savoir si il faut grouper dans une seule compétence les quatre opérations arithmétiques de base (addition, soustraction, multiplication et division) ou les séparer en quatre compétences ne peut pas être clairement établi. Cependant, il devrait être possible d'avoir un assentiment général sur le fait que les capacités d'analyse XML et l'addition doivent se situer dans des compétences différentes. Une compétence doit en effet représenter un ensemble *cohérent* de capacités.

Convenir qu'à une compétence doit correspondre une et une seule capacité (ou réciproquement) pourrait sembler raisonnable, mais la réponse n'est pas aussi simple, notamment car il existe des dépendances entre les capacités, et dans tous les cas cela risque de ne pas résister à la réalité des programmeurs... Les problèmes qui apparaissent ici sont les mêmes que ceux habituellement (et universellement) rencontrés en génie logiciel, et en conception objet en particulier, concernant la décomposition en objets.

2.1.3 Avantages

Ce paradigme offrant une construction dynamique d'agents à partir de compétences procure de nombreux avantages. D'abord, l'autonomie des agents est potentiellement accrue puisqu'ils peuvent acquérir les compétences qui leur manquent. L'agent peut de même faire évoluer et améliorer les compétences dont il est déjà doté. Remarquons que d'un point de vue opérationnel cette évolutivité dynamique apporte une souplesse supplémentaire. Quand une compétence d'un agent doit être changée (a priori pour l'améliorer), il n'est plus nécessaire d'accomplir le cycle classique (et pénible) : "l'arrêter, modifier le source, compiler et redémarrer". La nouvelle compétence peut être dynamiquement enseignée à l'agent. Cela peut être particulièrement important pour un agent dont la durée de vie est longue et qui est dédié à un rôle qui ne peut tolérer la moindre interruption.

Ensuite, le système multi-agent tire profit de ce principe. Il y gagne en robustesse puisqu'un agent détenteur d'une compétence critique qui doit quitter le système peut céder à un autre agent cette compétence et ainsi garantir la pérennité et la cohérence du système. L'efficacité est également potentiellement améliorée puisqu'un agent submergé par des requêtes pour exploiter une de ses compétences peut choisir de l'enseigner à d'autres agents (qu'il peut éventuellement créer et éduquer dans ce but spécifique) afin d'alléger sa charge.

Avec ce principe d'évolution dynamique d'un agent, il n'est plus possible d'utiliser le terme de "classe" d'agents. Même si pour différents agents vous partez d'un base commune, dans la mesure où ils peuvent, et vont probablement, recevoir une éducation différente due à leurs "expériences" individuelles, ils vont bientôt diverger. Il sera de ce fait impossible de les considérer comme appartenant à une même "classe". Cette notion n'a définitivement plus de sens dans ce contexte. Cela constitue une différence forte entre une telle programmation orientée agents et la programmation orientée objets.

2.2 Le modèle d'organisation hiérarchique

L'origine du projet MAGIQUE est sa proposition d'un modèle organisationnel pour systèmes multi-agents qui s'appuie sur la notion de hiérarchie évolutive (Bensaid and Mathieu 1995, Bensaid and Mathieu 1997) et d'"appel à la cantonnade". Les agents mis en œuvre correspondent au modèle présenté dans la section précédente.

Dans MAGIQUE une société d'agents est définie à la base comme une hiérarchie, c'est-à-dire un arbre dont chaque nœud est un agent et dont les branches représentent les liens d'accointances par défaut. MAGIQUE correspond donc à la notion d'"acquaintance model" telle que donnée dans (Wooldridge, Jennings, and Kinny 2000) ou (Kinny, Georgeff, and Rao 1996). Cependant cette notion de "modèle d'accointances" ne fait que préciser que les agents interagissent entre eux et qu'ils doivent pour cela utiliser des chemins de communication. La notion d'organisation doit aller plus loin en fournissant un modèle de "construction" de ce réseau d'accointances. C'est pourquoi MAGIQUE accompagne la proposition de la structure hiérarchique d'un mécanisme par défaut de délégation d'une requête d'exécution de compétences : "l'appel à la cantonnade". Le principe en est le suivant, quand un agent souhaite utiliser une compétence :

- l'agent "possède" la compétence, il l'"invoque" directement,
- l'agent ne "possède" pas la compétence, plusieurs cas de figure possibles :

- il a une accointance particulière pour cette compétence, il lui demande alors de la réaliser pour lui,
- sinon, il est racine d'une hiérarchie et un membre de sa hiérarchie possède cette compétence, il transmet (récursivement *via* la hiérarchie) la délégation de réalisation de cette compétence à qui de droit,
- sinon, il demande à son supérieur hiérarchique de trouver quelqu'un de compétent pour lui et celui-ci réapplique ce même mécanisme récursivement.

Du point de vue de la programmation, l'invocation de compétence est très facile à réaliser. Là où en programmation objet vous écrivez :

```
object.ability(arg...);
```

pour faire un appel à une méthode `ability`, vous devez maintenant écrire :

```
perform("ability",arg...);
```

Cela a pour effet que la compétence nommée "`ability`" sera "invoquée" (sans avoir à savoir par qui[1]).

La primitive `perform` est dédiée à l'invocation de compétence pour lesquelles on n'attend pas de réponse. Il existe principalement trois autre primitives : `ask` quand une réponse asynchrone est désirée, `askNow` pour une réponse immédiate et `concurrentAsk` pour une invocation concurrente.

L'avantage de cette délégation du point de vue du programmeur, en comparaison des appels nominatifs, est qu'il n'a pas besoin de connaître explicitement les agents qui seront présents dans l'application au moment du codage. Les références se situent en effet au niveau des compétences. Il suffit pour le concepteur de savoir que la compétence est présente dans le système multi-agents, sans nécessairement connaître l'agent compétent. Le code produit gagne alors en réutilisabilité. L'application multi-agent gagne, quant à elle, en robustesse car l'agent réalisateur d'une compétence n'étant pas nécessairement prédéfini, il peut éventuellement varier au fil du temps, si un agent devient indisponible ou surchargé par exemple.

Ainsi quand un agent souhaite exploiter une compétence, peu importe qu'il la possède ou non. Dans les deux cas, la manière d'invoquer la compétence est la même. Si la réalisation de la compétence doit être déléguée à un autre, cela est fait de manière transparente pour lui, même si il n'a pas de lien d'accointance direct pour celle-ci. Un tel mécanisme facilite le développement en découplant la compétence de tout agent ou système multi-agent. De plus il favorise la fiabilité du système en améliorant la tolérance aux pannes.

Un autre point fondamental dans MAGIQUE est la possibilité pour un système multi-agents d'évoluer dynamiquement. En effet avec seulement les communications hiérarchiques, le modèle serait probablement trop rigide. C'est pourquoi MAGIQUE offre la possibilité de créer des liens directs (i.e. en dehors de la hiérarchie) entre deux agents. Nous les appelons *liens d'accointances* (par opposition aux liens *hiérarchiques* même si ceux-ci désignent également des accointances).

[1] Bien évidemment, MAGIQUE offre également la possibilité de préciser un destinataire si besoin est, ceci est d'ailleurs indispensable lorsque deux agents n'ont pas de relation d'accointance à travers une hiérarchie mais sont en relation directe.

La décision de la création de tels liens dépend de la politique de l'agent. L'idée visée est cependant la suivante : après quelques temps d'évolution du système multi-agents, si il apparaît des requêtes privilégiées (fréquentes) entre deux agents pour une compétence, la décision peut être prise de créer dynamiquent un lien d'accointance entre ces deux agents pour la compétence concernée. L'intérêt est évidemment de diminuer les communications et de mettre en évidence des liens "naturels" d'interaction entre les agents. Ces liens correspondent aux communications *horizontales*. Une stratégie de création de liens d'accointance peut donc être spécifiée au niveau des agents. Cette approche reprend le mode de fonctionnement des interactions au sein d'une entreprise.

Ainsi, dans MAGIQUE, l'organisation hiérarchique n'est proposée que comme support par défaut. Cette structure est destinée à évoluer en accord avec le comportement dynamique du système multi-agents en favorisant les relations les plus fréquentes. Il en résulte, qu'après un certain temps, le système multi-agents devrait plus ressembler à un graphe. L'intérêt est que, la structuration étant dynamique et auto-adaptative, le concepteur d'un système multi-agents peut se reposer en partie sur ce mécanisme lors de la conception de l'organisation de son système multi-agents.

La dynamicité dans MAGIQUE opère à plusieurs niveaux. Elle est d'abord *individuelle* puisqu'un agent peut acquérir ou oublier des compétences. Dans MAGIQUE, cela se traduit par un échange effectif de compétences entre agents, éventuellement distants, à l'initiative des agents eux-mêmes. La dynamicité est également *relationnelle* puisque des liens d'accointances peuvent être créés lorsque des relations favorisées apparaissent entre deux agents de la hiérarchie. Cela a pour effet de supprimer les communications récurrentes le long de l'arborescence puisque alors la communication entre les agents devient directe. Couplée avec le mécanisme de délégation, cette création dynamique de relations de communication contribue à rendre *non déterministe* le fonctionnement d'une application multi-agent : deux exécutions successives ne produiront pas nécessairement la même structure d'accointances et donc de communications et donc les compétences ne seront pas réalisées nécessairement par les mêmes agents. La dynamicité est enfin *organisationnelle* ou *architecturale* puisque des agents peuvent être créés ou détruits afin d'adapter le système multi-agents à certaines contraintes.

Pour caractériser cette dynamicité nous avons identifié trois principes simples qui peuvent être utilisés pour améliorer le comportement global et qui entraînent une organisation dynamique de la structure sociale :

1. *avoir un bon carnet d'adresses*,
2. *partager la connaissance*,
3. *recruter de nouveaux collaborateurs compétents*.

Afin de pouvoir appliquer ces principes les agents doivent d'abord avoir la possibliité de créer dynamiquement de nouveaux liens d'accointances afin d'auto-adapter l'organisation (Ghanea-Hercock 2000). Cependant, ils doivent disposer d'un moyen pour trouver le "bon agent". C'est pourquoi un mécanisme de recherche de routage par défaut des messages et une structure d'accointances par défaut doivent être fournis pour permettre d'atteindre le "bon agent", au moins par des intermédiaires. Ensuite il faut que les agents puissent apprendre de nouvelles

compétences auprès des agents (et donc les agents doivent être capables d'enseigner aux autres, voir (Clement 2000)). Un mécanisme qui permet cela doit être fourni tout en tenant compte de l'aspect distribué. Enfin, les agents doivent pouvoir créer dynamiquement de nouveaux agents, et en utilisant la capacité d'acquérir des compétences, ces agents peuvent être adaptés aux besoins, il suffit de leur inculquer les bonnes compétences.

Ces différents critères sont offerts par les propriétés de dynamicité offertes par les modèles d'agent et d'organisation de MAGIQUE. Nous avons mené différentes expérimentations concernant ces trois principes d'évolution dynamique des systèmes multi-agents. Ce travail a été publié dans (Mathieu, Routier, and Secq 2002).

2.3 API et applications

Les deux sections précédentes ont présenté les principaux concepts et principes mis en œuvre dans MAGIQUE : des agents construits par acquisition dynamique de compétences s'intégrant dans une organisation par hiérarchique offrant un mécanisme de délégation automatique de services et pouvant évoluer dynamiquement.

Comme je l'ai indiqué dans la première partie, nous partageons au sein de l'équipe SMAC le souci de mettre en pratique les concepts que nous proposons. MAGIQUE a donc été concrétisé sous le forme d'une API Java dont l'objectif est de permettre le développement d'applications réparties à base d'agents. Nous avons voulu cette API comme un framework général sans orientation applicative. Elle fournit donc les éléments de base pour des applications agents, les aspects particuliers étant à développer comme des bibliothèques constituées de compétences. En plus de l'implémentation de nos concepts, une de nos préoccupations lors de l'écriture de cette API a été de la rendre aussi facile à appréhender que possible pour quelqu'un ayant une connaissance raisonnable du langage Java. Par différentes expériences avec des étudiants et le retour de différents utilisateurs, il semblerait que cet objectif ait été raisonnablement atteint.

2.3.1 Mise en place du modèle

Après avoir défini les notions de compétence et de message, l'étape suivante a consisté en la création de notre agent atomique (classe `AtomicAgent`) doté des deux compétences minimales lui permettant d'acquérir des nouvelles compétences et de communiquer (cf. figure 2.1).

La notion d'agent MAGIQUE (classe `Agent`) a ensuite été mise en place conformément au modèle. C'est-à-dire que nous avons développé les différentes compétences liées au modèle organisationnel MAGIQUE. Il s'agit essentiellement des compétences permettant la mise en œuvre de la hiérarchie (connexion à un supérieur, notion d'équipes, etc.) et de la gestion de la délégation de services. Ces compétences sont ensuite enseignées dynamiquement à l'agent atomique :

```
public class Agent extends AtomicAgent {
...
   protected void initBasicSkills() throws SkillAlreadyAcquiredException {
      addSkill(new fr.lifl.magique.skill.system.DisplaySkill());
      addSkill(new fr.lifl.magique.skill.system.AddSkillSkill(this));
      addSkill(new fr.lifl.magique.skill.system.LearnSkill(this));
      addSkill(new fr.lifl.magique.skill.system.ConnectionSkill(this));
      addSkill(new fr.lifl.magique.skill.magique.BossTeamSkill(this));
      addSkill(new fr.lifl.magique.skill.magique.ConnectionToBossSkill(this));
      addSkill(new fr.lifl.magique.skill.magique.KillSkill(this));
   }
```

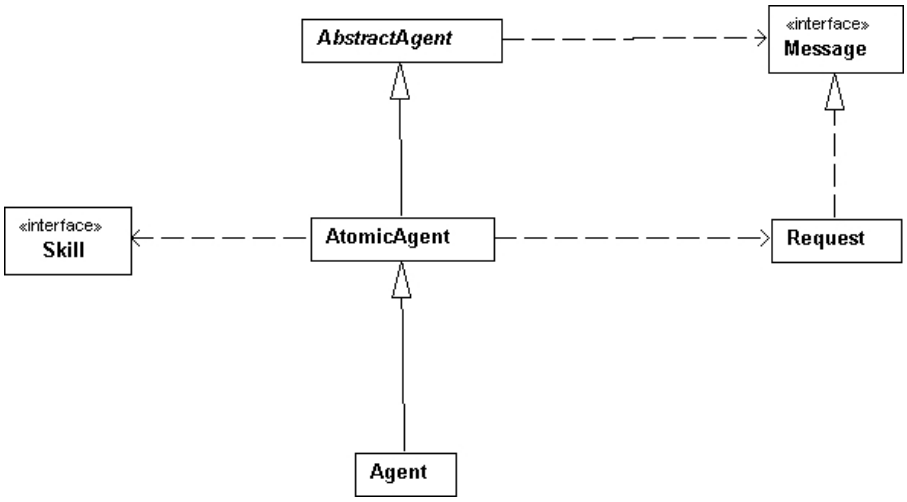

Figure 2.1: Les dépendances entre les principales classes de l'API MAGIQUE. La classe `Agent` correspond aux agents MAGIQUE.

```
}
```

L'extrait de code ci-dessus est directement issu du source de l'API. La méthode `initBasicSkills` est appelée dès la fin de la construction d'une instance de type `Agent` (en fait `AtomicAgent`). Comme on peut le constater (et le comprendre facilement) les compétences propres à l'API MAGIQUE (en gras) sont dynamiquement ajoutées à l'agent via les appels à la méthode `addSkill`. Les agents MAGIQUE sont donc bien obtenus à partir de l'agent atomique par enrichissement dynamique de compétences. L'existence d'une classe spécifique aux agents MAGIQUE ne se justifie en effet que pour offrir des méthodes facilitant le développement. La mise en place de tout autre modèle d'agent se ferait de la même manière, une fois les compétences propres à ces modèles développés il faut les enseigner à l'agent atomique.

Il en est de même des agents impliqués dans une application. Leur construction est simple une fois les compétences développées puisqu'il suffit de lui enseigner celles-ci. Il peut ensuite être intégré à une hiérarchie existante :

```
import fr.lifl.magique.*;
...
   // création d'un agent atomique
   Agent myAgent = createAgent("myName");
   // l'agent apprend dynamiquement des compétences
   myAgent.addSkill("SkillOne");
   myAgent.addSkill("SkillTwo");
   myAgent.addSkill("SkillThree", params);
   // il rejoint une hiérarchie (= SMA)
   myAgent.connectToBoss("bossName@host.dom.XX:4444");
   ...
```

La création de compétence ne cache pas de difficulté particulière. Il s'agit de développer une classe Java dont les méthodes publiques seront les capacités que pourra exploiter l'agent compétent. La signature de la méthode sert de sémantique pour les messages.

```
import fr.lifl.magique.*;
import fr.lifl.magique.skill.*;
...
public class ASkill implements Skill {
   public ASkill() {...}

   // l'agent pourra utiliser <ability>
   public void ability(...)  {
      ...
   }
}
```

S'appuyant sur notre modèle d'agent minimal, la structure que nous proposons permet la construction d'agents participant à des structures organisationnelles différentes de celle de MAGIQUE. René Mandiau et Emmanuel Adam de l'Université de Valenciennces ont ainsi utilisé MAGIQUE et développé les compétences permettant la mise en place d'une organisation holonique (Adam and Mandiau 2005).

2.3.2 Délégation, accointances et échange de compétences

Les primitives d'invocation de compétences précédemment évoquées sont évidemment mises en place: `perform`, `ask`, `askNow`. Elles permettent les invocations de services sont le destinataire est nommé ou non, ce second cas mettant en œuvre le mécanisme de délégation automatique.

Les connexions entre agents sont réalisées par les primitives `connectTo` pour le connexions "directes" et `connectToBoss` pour la mise en place des liens hiérarchiques. Enfin, l'API offre les primitives permettant l'échange dynamique de compétences entre les agents, y compris si ceux-ci sont physiquement distribués. Cet échange ne nécessite pas que soit réalisée une distribution de bibliothèques applicatives sur l'ensemble des machines impliquées dans l'application. Le cas échéant, lors de l'échange d'une compétence entre deux agents, le bytecode nécessaire est automatiquement transféré entre les plate-formes MAGIQUE (cf. Figure 2.2). Ce mécanisme est rendu possible par la présence d'"agents plate-forme" qui sont des agents atomiques auxquels nous avons ajouté les compétences nécessaires. Le déploiement d'applications multi-agents se trouve ainsi simplifié facilitant le travail du développeur.

L'API et son utilisation sont présentées plus en détail dans le tutoriel MAGIQUE (Mathieu and Routier 2000-2001).

2.3.3 IDE

Pour encore faciliter la mise en place de systèmes multi-agents applicatifs, nous avons conçu un environnement de développement et de déploiement (cf. Figure 2.3). Celui-ci permet de constituer les agents en leur assignant leurs compétences, de construire la hiérarchie puis de déployer automatiquement les agents, sans qu'il soit nécessaire d'avoir préalablement déployé du code applicatif sur les machines distantes et ce grâce à l'existence des agents plate-formes et à l'échange dynamique de compétences. Cet environnement est en effet développé à base d'agents atomiques (non spécifiquement MAGIQUE) dotés des compétences propres à cet environnement. Enfin une console d'administration permet d'interagir directement avec les agents.

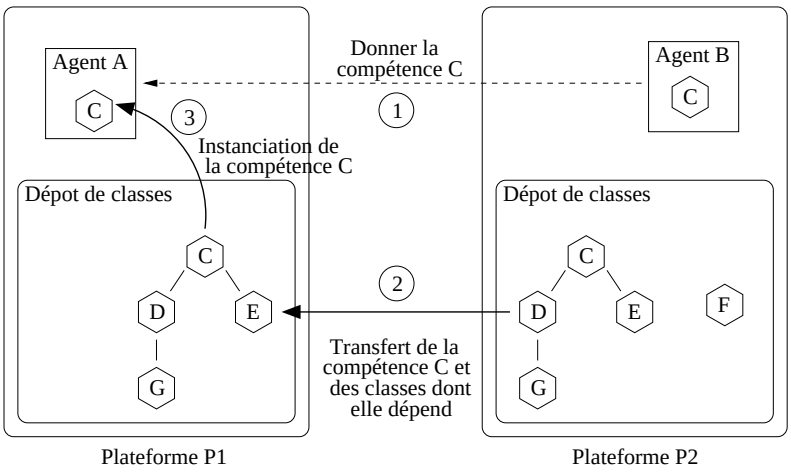

Figure 2.2: Echange dynamique de code entre plate-formes

2.3.4 Exploitation

Les agents proposés par l'API MAGIQUE n'ont aucune capacité applicative particulière. Cependant, avec cette base, le constructeur d'applications se voit dégager des soucis liées à la distribution, la communication entre les agents et, au moins partiellement, à la mise en place de son organisation. Il peut s'appuyer sur la hiérarchie de base et son mécanisme de délégation et laisser le système multi-agents évoluer en utilisant les possibilités de réorganisation dynamique.

Le concepteur peut alors se consacrer à l'essentiel : le développement des aspects applicatifs. Cela correspond à l'écriture de compétences. Le concepteur doit donc déterminer les fonctionnalités nécessaires à son application, les regrouper en ensemble cohérent pour construire les compétences puis affecter ces compétences à des agents MAGIQUE pour les faire évoluer vers les agents applicatifs souhaités. L'attribution de compétences à des agents revient à leur attribuer des rôles : celui induit par la compétence. Les agents sont ensuite intégrés à une hiérarchie. Si le concepteur a tout intérêt à réfléchir à l'organisation logique de celle-ci, il peut néanmoins se reposer partiellement sur le mécanisme de délégation automatique qui garantit qu'une requête d'exploitation atteindra l'agent compétent dès que la compétence est présente dans le système multi-agents,

2.3.5 Applications

Nous avons utilisé les concepts de MAGIQUE et l'API qui les met en œuvre pour le développement de différentes applications. A travers ces expériences nous avons pu vérifier les avantages apportés par le modèle notamment au niveau de la souplesse de conception. Nous avons ainsi pu valider notre approche par compétences. La démarche de conception par agent est toujours la même. D'abord, le travail d'analyse consiste à identifier les compétences mises en œuvre dans l'application et les rôles qu'elles induisent. Ensuite vient le développement de ces compétences. La construction des agents est ensuite aisée puisqu'il suffit de leur "ajouter" les compétences qui leur sont dues. Le travail de distribution sur le réseau et d'échange de messages est pris en charge par les fonctionnalités offertes de base par MAGIQUE.

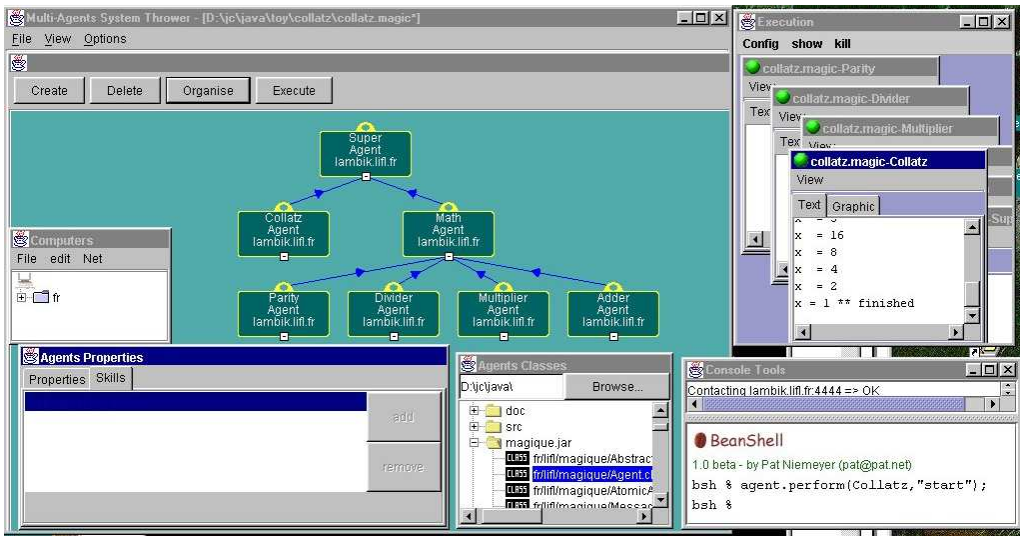

Figure 2.3: Environnement de développement et de déploiement MAGIQUE.

Parmi ces applications, j'en citerai deux en particulier. La première se situe dans le domaine du travail coopératif. Elle fournit un support permettant la tenue d'une conférence entre intervenants physiquements distribués. La seconde s'inscrit dans le calcul distribué. Il s'agit du framework RAGE permettant la définition de tâches de calcul et leur distribution sur un réseau de machines.

L'apport de la première application a été pour nous de tester la conception d'application avec MAGIQUE avec la création d'agents applicatifs par ajout de compétences à partir de l'agent atomique. Mais surtout, cette application offrait un exemple pour l'échange dynamique de compétence entre les agents permettant une évolution dynamique de rôles. La seconde application a été l'occasion à nouveau de valider l'approche par compétences et aussi d'aborder un cas faisant intervenir plus de rôles et une hiérarchie plus complexe. De plus cette application nécessitait des créations dynamiques d'agents en fonction des tâches de calcul à réaliser.

Remarquons que dans les deux cas, la présence d'agents dans l'application n'est pas nécessairement perceptibles par l'utilisateur de l'application. C'est essentiellement l'approche et la conception de l'application qui sont "orientées agents".

Travail coopératif

L'application "diapo-conférence". L'objectif de cette application de travail coopératif est de fournir un support permettant la tenue d'une conférence entre des intervenants physiquement distribués. Chacun des intervenants dispose d'un ensemble de ressources documentaires ("*diapositives*"). Dans une première version de cette application, ces ressources sont décrites par des documents HTML. L'application doit permettre à l'utilisateur de diffuser ces documents auprès des autres. Il ne peut cependant le faire qu'à la seule condition qu'il soit détenteur de la *télécommande* (unique) associée à l'application. Cette télécommande lui permet de parcourir et de diffuser ses ressources. Mais elle tient également lieu de "pointeur" que le conférencier peut utiliser pour

attirer l'attention des autres utilisateurs sur un point précis du document diffusé, ce pointeur étant visible par tous les utilisateurs en temps réel.

On distingue donc deux *rôles* dans cette application : celui de *conférencier* pour celui qui détient la télécommande, et celui d'*auditeur* pour les autres participants. Comme dans une conférence réelle, le conférencier peut à tout moment céder la télécommande à un auditeur et les rôles respectifs tenus par les intervenants évoluent alors dynamiquement. Une attention particulière a été portée au respect de la conformité avec la tenue des conférences réelles.

Les figures 2.4 et 2.5 donnent un aperçu de l'application du point de vue du conférencier et du point de vue d'un auditeur quelconque. On peut distinguer différents outils sur les deux vues : la visionneuse sur la gauche des écrans (on peut remarquer que celle de l'auditeur a la même taille que celle du conférencier), le pointeur de souris que l'on peut apercevoir dans la visionneuse du conférencier est visualisée par un point dans la fenêtre de l'auditeur. D'autres outils comme la télécommande (active chez le conférencier et passive chez l'auditeur), les fenêtres d'équipe et de messagerie, ainsi que l'assistant en haut de chaque écran (il fait une annonce dans la fenêtre du conférencier) sont également visibles.

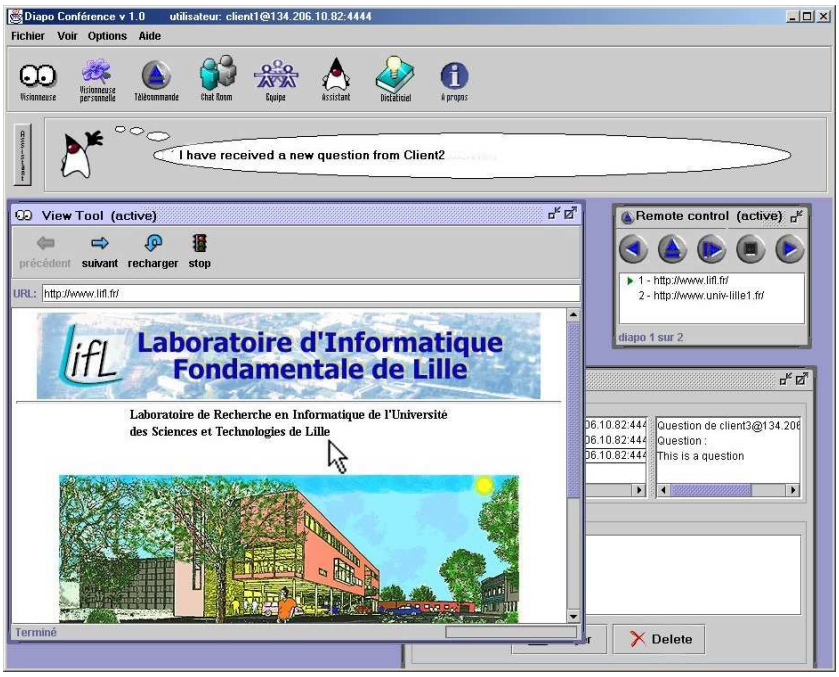

Figure 2.4: L'application du côté *conférencier*. On peut entre autre voir la fenêtre de diffusion des diapositives, l'assistant qui fait part d'un évènement.

Il est possible pour le conférencier de céder cette télécommande à un auditeur et d'induire ainsi un changement de rôle pour chacun des participants. C'est cet aspect qui nous intéressait le plus dans le cadre expérimental offert par cette application. Cet échange fonctionnel se traduit en effet par un transfert effectif de la compétence de gestion de la télécommande entre les agents et induit un changement de rôles des agents impliqués dans l'échange de compétence. Nous reviendrons sur ce point un peu plus loin.

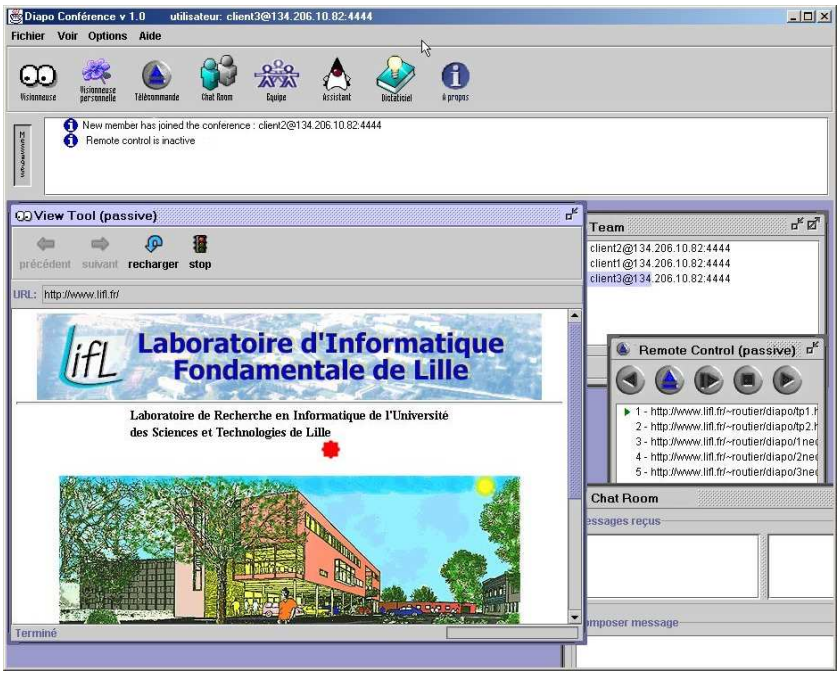

Figure 2.5: L'application du côté *auditeur*, on peut remarquer le pointeur rouge représentant la souris du conférencier et dont il reproduit les déplacements.

Les autres fonctionnalités offertes par l'application sont des outils qui favorisent la coopération et la prise de conscience par chacun des autres intervenants. Citons principalement : la "visualisation" de l'assistance, un système de messagerie et un assistant. Ce dernier a trois fonctions principales : il permet une mise en évidence d'événements, fournit une aide pour contribuer à la coopération et offre une modélisation des autres intervenants en vue de futures collaborations.

Nous allons aborder maintenant rapidement la conception de cette application avec MAGIQUE.

La réalisation avec MAGIQUE Il convient de déterminer les agents intervenants et les compétences de chacun, puis de définir l'organisation du système et les interactions possibles.

Nous avons déjà clairement identifié deux de ces rôles : ceux de *conférencier* et d'*auditeur*. Il existe cependant un troisième rôle qui est celui du *coordinateur*. Il peut être vu comme le représentant de l'*environnement* et constitue le support de l'interaction et concrétise par son existence la cohabitation de l'ensemble des participants à la conférence.

Les compétences correspondant aux différentes fonctionnalités brièvement décrites précédemment doivent ensuite être développées. La compétence de gestion de la *télécommande* est la plus intéressante car c'est elle qui crée la différence entre les deux rôles *conférencier* et *auditeur*. L'évolution dynamique des rôles en cours de session (de *conférencier* à *auditeur* et réciproquement) implique une évolution dynamique de la compétence connue par les agents. Il est en effet nécessaire d'accorder au nouveau conférencier les fonctions pour mener la conférence et de les retirer à l'ancien conférencier. L'évolution dynamique des rôles *conférencier*$\longleftrightarrow$*auditeur* est facilement prise en compte par MAGIQUE. Notre plate-forme a en effet l'avantage de permet-

tre l'évolution dynamique des compétences des agents par échanges entre eux. Les agents apprennent et oublient *effectivement* des compétences. Ces échanges de compétences sont réels, indépendamment de toute hypothèse sur le code de ces compétences et de la distribution sur le réseau des agents. Le *conférencier* peut donc, comme cela se passe lors du passage de micro dans une conférence réelle, donner la compétence de gestion de la télécommande au nouveau *conférencier* et de ce fait la *perdre* (cf. figure 2.6). Il ne s'agit pas ici de la simple modification de la valeur d'un attribut quelconque, mais d'une mutation concrète de l'agent et du rôle qu'il tient et donc en quelque sorte de ses droits dans l'application.

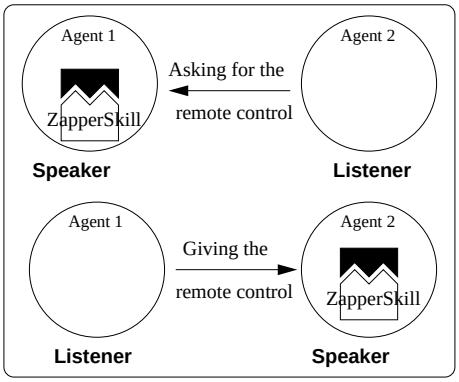

Figure 2.6: Evolution des rôles par échange de la compétence "Télécommande". Il s'agit d'un échange effectif de la télécommande. L'agent conférencier donne la compétence Télécommande à l'auditeur et n'a plus possibilité d'accéder aux services qu'elle offre.

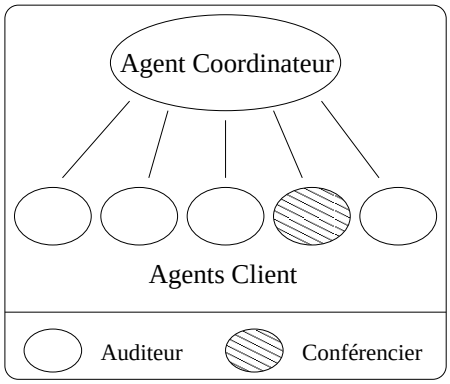

Figure 2.7: L'organisation hiérarchique du système multi-agents

Il faut bien sûr ensuite faire le choix de la structure organisationnelle pour le système multi-agents. Avec MAGIQUE cette structure est par défaut hiérarchique mais dans le cas de cette application, la hiérarchie est réduite à sa plus simple expression. Les arités d'occurrence des différents rôles sont assez triviale également : il n'y a qu'un coordinateur, qu'un conférencier (même si l'agent tenant ce rôle peut évoluer) et un nombre quelconque d'auditeurs (cf. Figure 2.7).

Enfin, la construction des agents se fait très simplement, il suffit d'"enseigner" à l'agent de base les compétences requises une par une. La couche MAGIQUE se charge ensuite de gérer les connexions des agents et le transport des messages dus aux interactions entre agents. Ces communications n'apparaissent pas explicitement au niveau du code, puisqu'elles sont induites par les invocations de compétences et donc prises en charge par MAGIQUE.

Ces travaux ont été publiés dans (Mathieu and Routier 2001, Mathieu and Routier 2002).

Rage

Le second exemple d'application que je présenterai est un framework de calcul distribué, RAGE, qui a été implémenté avec notre plate-forme MAGIQUE. Cet exemple illustre à nouveau

la facilité de conception et d'implémentation de telles classes d'applications, lorsque l'on analyse ces systèmes en terme de rôles et de compétences.

Le framework RAGE, acronyme de Reckoner AGEnt, est un cadre de développement destiné aux scientifiques non-informaticiens et dédié à la réalisation de calculs distribués. Ce système supporte l'exécution de plusieurs calculs simultanément et peut voir sa puissance de calcul augmenter ou diminuer dynamiquement en fonction de la disponibilité des plate-formes hébergeant les calculs. Dans RAGE, il y a deux types de clients : ceux qui fournissent des calculs à effectuer, et ceux qui proposent leur puissance de calcul. Le premier type de client correspond au scientifique désirant effectuer des calculs, et nécessite donc l'écriture d'un programme. Tandis que le second client est fourni par le framework, et correspond à un programme que les utilisateurs n'ont qu'à exécuter pour augmenter la puissance de calcul de RAGE, à la manière du Grid Computing.

La *simplicité* d'utilisation est le critère principal de ce framework. Cet environnement étant destiné à des non-informaticiens, nous avons défini un faible nombre de concepts que l'utilisateur devra s'approprier pour pouvoir alimenter RAGE en calculs. Ces concepts sont principalement celui de *tâche* et de *résultat*. Une *tâche* est un algorithme qui pourra être distribué dans le système, tandis qu'un *résultat* représente les données produites par la réalisation d'une *tâche* et devant être sauvegardées.

Les principales étapes que nous avons suivies lors de l'analyse et la conception de RAGE sont les mêmes que pour l'application précédente. On commence par la définition des rôles impliqués dans le système, de leurs interactions et de leurs compétences, puis vient la définition de l'organisation des rôles au sein du système et enfin l'association des rôles aux agents concrets.

Identifier les différentes entités qui interviennent dans le système revient à déterminer les fonctions que le système doit fournir, il faut ensuite regrouper ces fonctions de manière homogène. Dans le cas de notre framework de calcul distribué, nous avons identifié les rôles suivants : *Boss* qui est responsable des interactions avec les utilisateurs, *Task Dispatcher* qui se charge de la répartition des *tâches* et gère aussi la tolérance aux pannes, *Platform Manager* gère les agents qui ont à effectuer les calculs des *tâches*, et doit s'assurer que l'approvisionnement en *tâche* est toujours suffisant, *Reckoner Agent* qui calcule les *tâches* et retourne les *résultats*, *Repositories Manager* qui gère le stockage et l'accès aux *résultats* et enfin *Result Repository* qui réalise un miroir de la base de données des *résultats*.

Dans la mesure où nous avons implémenté ce système avec MAGIQUE, nous avons concrétisé ces rôles en développant les compétences nécessaires. Par exemple, le rôle *PlatformManager* est défini par une compétence qui définit son comportement : maintenir un *pool* de tâches et gérer leur distribution aux *Reckoner Agent*s. Un rôle est ainsi défini dans MAGIQUE par un ensemble de compétences. Les interactions ne sont pas réifiées et se trouvent enfouies dans le code des compétences.

Après avoir décrit les rôles, il est nécessaire de définir l'architecture du système à savoir l'organisation des agents dans le système multi-agents. Lorsque l'on travaille avec MAGIQUE, cela signifie que l'organisation doit satisfaire deux contraintes : d'abord un groupement logique des rôles, mais aussi un schéma de communication par défaut facilitant la délégation de requête.

La figure 2.8 représente la hiérarchie des rôles choisie dans RAGE : elle ne définit pas comment les rôles sont associés aux agents concrets, ni comment ces agents sont répartis sur le réseau, mais décrit la topologie des liens hiérarchiques. Cette topologie est l'organisation initiale, qui pourra évoluer au gré des interactions entre les agents.

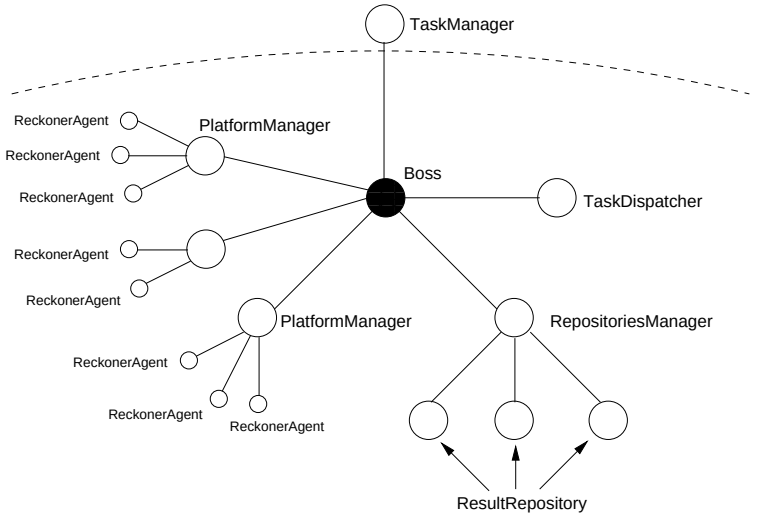

Figure 2.8: La hiérarchie d'agent dans Rage

Cette insfrastucture étant mise en place, l'exploitation de ce framework est simple. On y distingue deux types d'utilisateurs. Celui qui définit le calcul et celui qui prête de la puissance de calcul pour réaliser le calcul. Le premier se contente de définir ce qu'est une tâche élémentaire de son calcul, il s'appuie sur la structure `Task` proposée dans le framework et doit simplement programmer le calcul réalisé par la tâche, sa condition de terminaison/interruption et le résultat qu'elle produit une fois terminée. Par exemple, dans le cadre d'une application de factorisation d'entiers pour casser des clés en cryptographie, une telle tâche pourrait être un simple test de primalité d'un entier. La distribution d'un ensemble de telles tâches entraine la parallélisation massive du calcul. C'est le framework qui prend en charge automatiquement cette distribution et la récolte des résultats. Le client utilisateur n'a donc qu'à lancer un `ReconerAgent` sur sa machine sans être conscient de l'existence des autres agents. Les calculs qu'il réalisera (les tâches) lui seront automatiquement envoyées par la plate-forme.

Ces travaux ont été publiés dans (Mathieu, Routier, and Secq 2002c, Mathieu, Routier, and Secq 2002d).

2.4 RIO

La réalisation de ces différentes applications nous a amenés à une réflexion sur la démarche de conception d'applications multi-agents. Ce travail a été publié dans (Mathieu, Routier, and Secq 2002a, Mathieu, Routier, and Secq 2003c, Mathieu, Routier, and Secq 2003a) et a constitué le cœur de la thèse de Yann Secq (Secq 2 décembre 2003) qu'il a soutenue en décembre 2003 et que j'ai co-encadrée avec le professeur Philippe Mathieu. RIO adopte les principes de la programmation orientée interactions (Singh 1996). Notre approche est similaire à celle proposée dans Gaïa (Wooldridge, Jennings, and Kinny 2000) dans la mesure où nous modélisons les notions de Rôles (abstraits et concrets), de protocoles d'Interaction et d'Organisation, d'où le nom RIO donnée cette démarche. Cependant, à notre sens, Gaïa propose une analyse trop

générale qui ne permet pas un passage facile à la phase de conception. Une de nos préoccupations est de faciliter cette transition. C'est pourquoi RIO propose une description graphique des protocoles d'interaction qui constitue une spécification exécutable de ces interactions à travers un mécanisme automatique de génération des codes associés. Le déploiement est quant à lui facilité par l'utilisation de nos agents atomiques qui peuvent être dynamiquement enrichis de nouvelles compétences.

2.4.1 Spécification exécutable de protocoles d'interaction

Des travaux tels que AGENTUML (Odell, Parunak, and Bauer 2000, Huget 2001) qui étendent les diagrammes de séquence d'UML ou ceux de (Labrou and Finin 1994) à base de réseaux de Pétri colorés ont pour objectif de permettre la spécification de protocoles d'interaction entre agents.

Notre approche emprunte les mêmes principes : représenter l'interaction de manière globale. Cependant, à la différence des travaux précédents, notre objectif est la production de *spécification exécutable* des interactions. C'est-à-dire que, non seulement nous souhaitons décrire le protocole, mais aussi que cette description permette grâce à une génération du code gérant l'interaction, son intégration directe dans le système. Notre modèle a pour but de faciliter la spécification et le déploiement de protocoles d'interactions au sein de systèmes multi-agents. Le concepteur décrit à travers un formalisme graphique la spécification de son interaction (cf. Figure 2.9), les autres étapes étant automatisées.

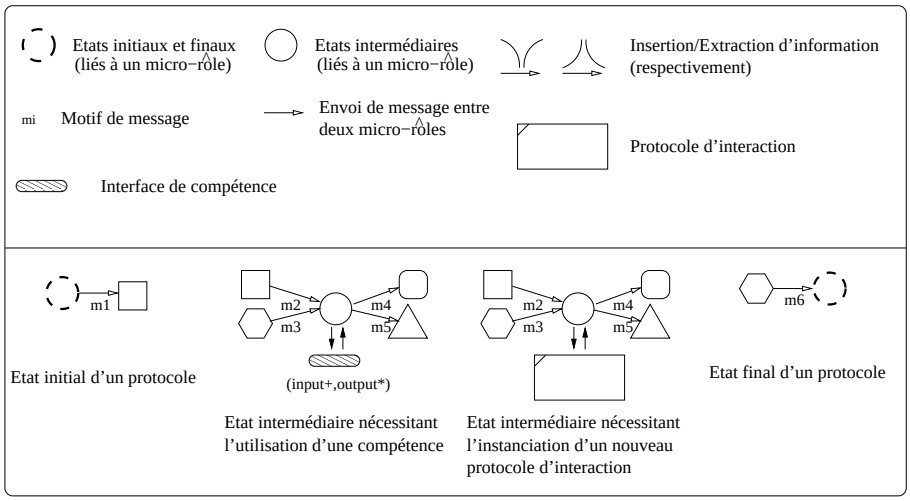

Figure 2.9: Éléments permettant la définition d'un protocole d'interaction.

La description graphique du protocole d'interaction spécifie le déroulement d'une conversation entre les différentes entités impliquées. Celles-ci sont identifiées par ceux que nous avons appelé des *micro-rôles*. Un micro-rôle correspond à l'abstraction d'un participant dans la conversation modélisée. Ils sont représentés par des nœuds du graphe dans notre formalisme. Ils peuvent être associés à un moment de l'interaction à une interface de compétence ou à l'initialisation d'un nouveau protocole. Les arcs qui relient ces micro-rôles correspondent aux envois de messages.

Ils sont étiquetés par des éléments permettant le typage des messages échangés. Le concepteur dispose ainsi d'une vue globale de l'interaction : participants, flux des messages et leur nature, compétences nécessaires.

Une fois le protocole d'interaction décrit dans son ensemble, nous proposons un processus de transformation permettant de produire du code exécutable (cf. Figure 2.10). La description du protocole étant globale, il est nécessaire de générer pour chacun des micro-rôles la vue locale du protocole d'interaction. C'est cette vue locale, une fois traduite en code exécutable, qui peut ensuite être dynamiquement distribuée aux agents du système assumant le micro-rôle concerné.

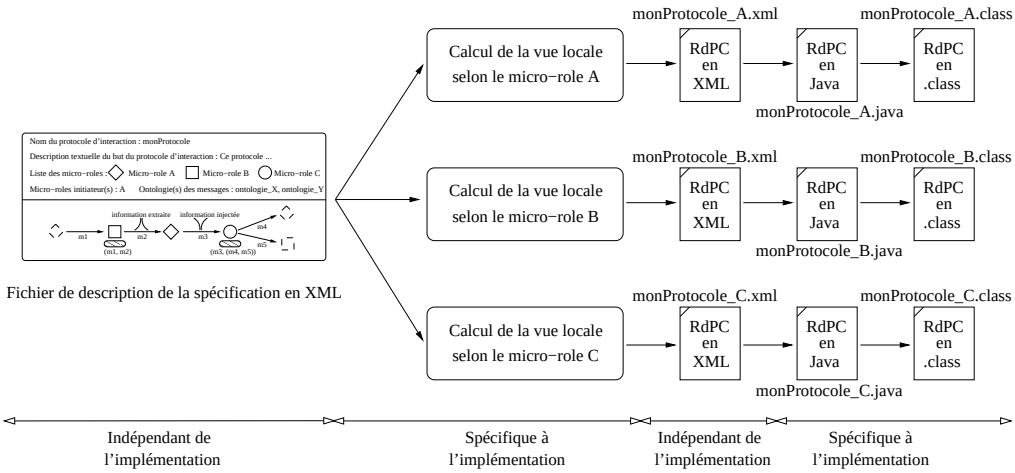

Figure 2.10: Spécification exécutable : de la description graphique au code

Ces travaux ont été publiés dans (Mathieu, Routier, and Secq 2003b)

2.4.2 RIO : démarche méthodologique de conception de systèmes multi-agents

La démarche que nous proposons avec RIO se décompose en quatre phases (cf. Figure 2.11). Les deux premières concernent des spécifications qui peuvent être réutilisées entre différentes applications alors que les deux dernières sont a priori propres à l'application considérée.

La première phase consiste en l'identification des interactions et des micro-rôles qui y prennent part. Le choix de la granularité de ces protocoles a un impact sur leur réutilisabilité. A la fin de cette phase, le concepteur dispose donc d'un ensemble de protocoles d'interaction. De plus, à chacun des micro-rôles sont associées les interfaces de compétences nécessaires à l'interaction.

La phase suivante consiste en la définition des *rôles composites* obtenus par agrégation de micro-rôles de différentes interactions. Il s'agit ici d'un regroupement permettant d'exprimer qu'un rôle donné est généralement impliqué dans plusieurs tâches. La notion de rôle composite donne une cohérence logique entre les micro-rôles qui représentent les différentes facettes d'un code. Un micro-rôle définit donc un rôle abstrait regroupant un ensemble de micro-rôles cohérents.

La troisième phase correspond en la réunion des rôles composites au sein d'*agents abstraits* introduisant ainsi des dépendances entre ces rôles. Il s'agit ici en fait de définir une société

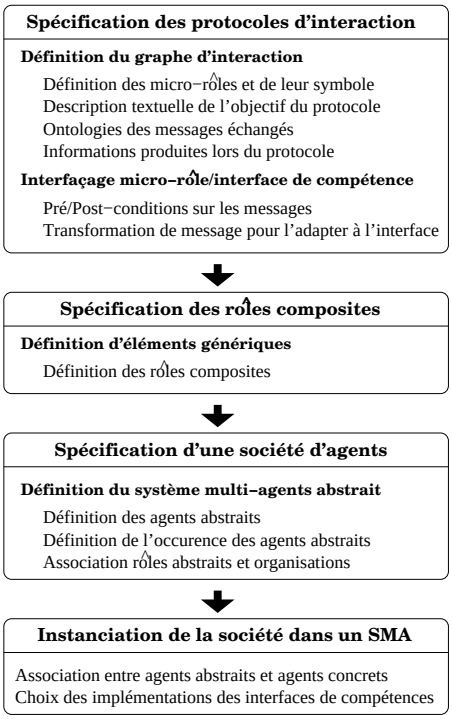

Figure 2.11: La démarche méthodologique RIO

abstraite d'agents ou, en plus de ces patrons d'agents, on précise la cardinalité de ces agents et la structure organisationnelle retenue, ce qui revient à choisir le modèle d'accointance.

Enfin, il faut déployer la société d'agents. Les agents abstraits sont alors projetés sur les agents concrets du système en respectant les arités définies. C'est également lors de cette quatrième phase qu'est réalisée la liaison entre une interface de compétences et sa réalisation.

La Figure 2.12 reprend l'ensemble de ce processus illustré par un rapide exemple.

En adoptant les principes de la programmation orientée interactions et en proposant un modèle de spécifications exécutables des protocoles d'interaction, RIO propose une méthodologie pour la réalisation d'applications multi-agents distribuées. Le principal apport est de donner au concepteur une vue globale de chaque interaction tout en laissant au système le travail de projection de cette vue globale sur l'ensemble des vues locales des différentes entités. La démarche proposée permet également une construction incrémentale des systèmes multi-agents applicatifs puisque, de même que notre agent générique peut se voir enrichi de nouvelles compétences, le système peut être enrichi par des interactions.

Bibliographie

Adam, E.; and R. Mandiau. 2005. "Roles and hierarchy in multi-agent organizations," in *Proceedings of Multi-Agent Systems and Applications IV, 4th International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2005*, ed. by M. Pechoucek, P. Petta, and L. Varga, no. 3690 in LNAI, 539–542. Springer-Verlag.

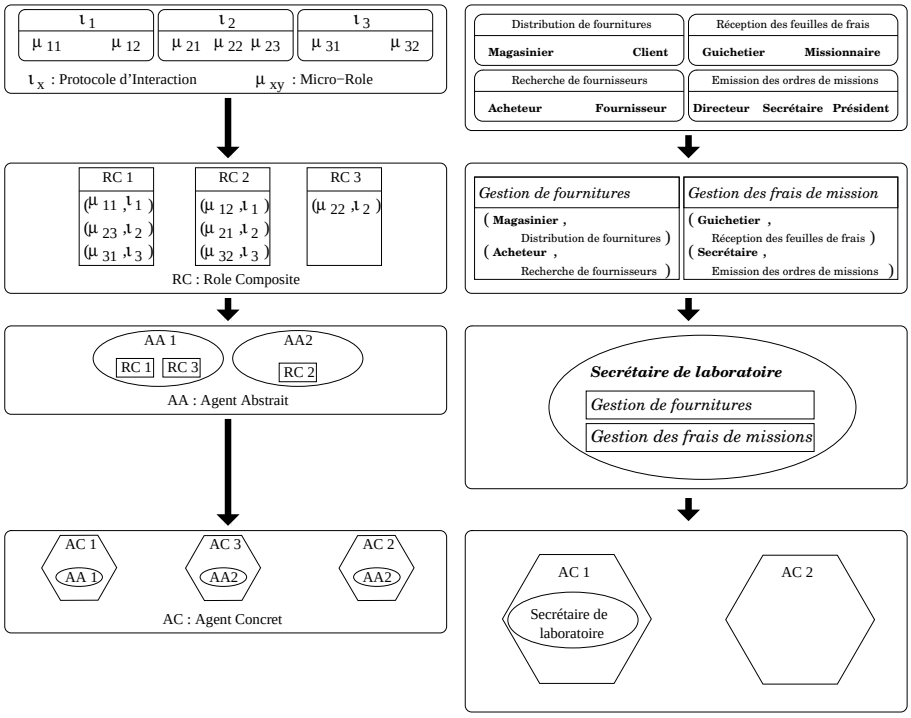

Figure 2.12: Synthèse de la démarche RIO mise en parallèle avec l'exemple d'un agent "secrétaire de laboratoire" impliqué dans la gestion de fourniture et celle des frais de mission.

Adam, E.; R. Mandiau; and C. Kolski. 2001. "Application of a holonic multi-agent system for cooperative work to administrative processes," *Journal of Applied Systems Studies*, 2, 110–115.

Bensaid, N.. 1999. "MAGIQUE, une architecture multi-agents hiérarchique," Ph.D. thesis, Université des Sciences et Technologies de Lille.

Bensaid, N.; and P. Mathieu. 1995. "Un modèle d'architecture multi-agents entièrement écrit en Prolog," in *IV Journées Francophones de Programmation Logique, JFPL'95*, 381–385, Dijon-France. teknea, Toulouse-France.

Bensaid, N.; and P. Mathieu. 1997a. "A Hybrid and Hierarchical Multi-Agent Architecture Model," in *Proceedings of PAAM'97*, 145–155.

Bensaid, N.; and P. Mathieu. 1997b. "A Hybrid Architecture for Hierarchical Agents," in *Proceedings of International Conference on Computationnal Intelligence and Multimedia applications, ICCIMA'97*, 91–95.

Clement, R.. 2000. "To Buy or to Contract Out: Self-Extending Agents in Multi-Agent Systems," in *SOMAS: A Workshop on Self Organisation in Multi Agent Systems*.

da Silva, J. T.; and Y. Demazeau. 2002. "Vowels Co-ordination Model," in *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, 1129–1136.

Ferber, J.; and O. Gutknecht. 1998a. “A meta-model for the analysis and design of organizations in multi-agent systems,” in *Proceedings of ICMAS’98*.

Ferber, J.; and O. Gutknecht. 1998b. “A meta-model for the analysis and design of organizations in multi-agent systems,” in *Proceedings of ICMAS’98*.

Ferber, J.; O. Gutknecht; and F. Michel. 2000. “The MadKit Agent Platform Architecture,” Discussion paper, W3C.

Franklin, S.; and A. Grasser. 1996. “Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agent,” in *Proceedings of the 3rd International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag.

Gerber, C.; J. Siekmann; and G. Vierke. 1999. “Holonic Multi-Agent Systems,” Discussion paper, DFKI GmbH.

Ghanea-Hercock, R.. 2000. “Spontaneous Group Formation in Multi-Agent Systems,” in *SOMAS: A Workshop on Self Organisation in Multi Agent Systems*.

Guessoum, Z.. 1996. “Environnement de développement et de conception de systèmes multi-agents,” Ph.D. thesis, Université de Paris 6.

Horling, B.; and V. Lesser. 1998. “A Reusable Component Architecture for Agent Construction,” Discussion paper, UMass Computer Science.

Huget, M.-P.. 2001. “Une ingénierie des protocoles d‘interaction pour les SMA,” Ph.D. thesis, Université de Paris 9.

Jennings, N.; and M. Wooldridge. 2000. *Handbook of Agent Technology*chap. Agent-Oriented Software Engineering. AAAI/MIT Press.

J.Ferber; O. Gutknecht; and F. Michel. 2000. “MadKit: une plate-forme multi-agent générique,” Discussion paper, FIPA.

Kinny, D.; M. Georgeff; and A. Rao. 1996. “A Methodology and Modelling Technique for Systems of BDI Agents,” Discussion paper, Australian AI Institute.

Labrou, Y.; and T. W. Finin. 1994. “A Semantics Approach for KQML - A General Purpose Communication Language for Software Agents,” in *CIKM*, 447–455.

Mathieu, P.; J. Routier; and Y. Secq. 2002a. “Dynamic Organization of Multi-Agent Systems,” in *Proceedings of the AAMAS’02 Conference*, 451–452.

Mathieu, P.; and J.-C. Routier. 2000-2001. “Tutoriel de Magique,” Equipe SMAC - LIFL.

Mathieu, P.; and J.-C. Routier. 2001. “Une contribution du multi-agent aux applications de travail coopératif,” *TSI Hermès Science Publication. Réseaux et Systèmes Répartis. Calculateurs Parallèles.*, 13. Numéro spécial télé-applications, 207–226.

Mathieu, P.; and J.-C. Routier. 2002. “A Multi-Agent Approach to Co-operative Work,” in *Proceedings of CADUI’02*, ed. by C. Kolski, and J. Vanderdonckt, 367–380. Kluwer Academics Press.

Mathieu, P.; J.-C. Routier; and Y. Secq. 2001. "Dynamic Skill Learning: A Support to Agent Evolution," in *Proceedings of AISB'01*, 25–32.

Mathieu, P.; J.-C. Routier; and Y. Secq. 2002b. "Principles for dynamic multi-agent organizations," in *Proceedings of 5th Pacific Rim International Workshop on Multi-Agents. PRICAI2002-PRIMA2002*, ed. by K. Kuwabara, and J. Lee, vol. 2413 of *LNAI*, 109–122. Springer-Verlag.

Mathieu, P.; J.-C. Routier; and Y. Secq. 2002c. "RAGE: An agent framework for easy distributed computing," in *Proceedings of AISB'02*, 20–24.

Mathieu, P.; J.-C. Routier; and Y. Secq. 2002d. "Using agents to build a distributed calculus framework," *The Interdisciplinary Journal of Artificial Intelligence and the Simulation of Behaviour*, 1(2), 197–208.

Mathieu, P.; J.-C. Routier; and Y. Secq. 2003a. "RIO: Rôles, Interactions et Organisations," in *Actes des Secondes Journées Francophones sur les Modèles Formels de l'Interaction, MFI'03*, ed. by A. Herzig, B. Chaib-draa, and P. Mathieu, 179–188.

Mathieu, P.; J.-C. Routier; and Y. Secq. 2003b. "Runnable specifications of interactions protocols for open multi-agent systems," in *Proceedings of the 2003 International Conference on Information and Knowledge Engineering, IKE'03*, ed. by N. Goharian, vol. II, 431–437.

Mathieu, P.; J.-C. Routier; and Y. Secq. 2003c. "RIO : Roles, Interactions and Organizations," in *Proceedings of the 3rd International/Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003*, ed. by V. Marik, J. Müller, and M. Pechoucek, 147–157.

Occello, M.; and J.-L. Koning. 2000. "Multiagent oriented software engineering: An approach based on model and software reuse.," in *Proceedings of Second International Symposium "From Agent Theory to Agent Implementation"*, 25–28.

Odell, J.; H. Parunak; and B. Bauer. 2000. "Extending UML for Agents," .

Petrie, C.. 2001. "Agent-Oriented Software Engineering," *Lecture Notes in AI*.

Ricordel, P. M.. 2001. "Programmation Orientée Multi-Agents :Développement et Déploiement de Systèmes Multi-Agents Voyelles," Ph.D. thesis, Institut National Polytechnique de Grenoble.

Ricordel, P.-M.; and Y. Demazeau. 2002. "VOLCANO : a vowels-oriented multi-agent platform.," in *Proceedings of the International Conference of Central Eastern Europe on Multi-Agent Systems (CEEMAS'01), From Theory to Practice in Multi-Agent Systems*, 252–262.

Secq, Y.. 2 décembre 2003. "RIO : Rôles, Interactions et Organisations, une méthodologie pour les systèmes multi-agents ouverts," Ph.D. thesis, Université des Sciences et Technologies de Lille, Co-dirigée avec le Professeur Philippe Mathieu.

Shoham, Y.. 1993. "Agent-Oriented Programming," *Artificial Intelligence*, 60, 51–92.

Singh, M. P.. 1996. "Toward Interaction-Oriented Programming," Discussion Paper TR-96-15, Department of Computer Science, North Carolina State University.

Travers, M.. 1996. "Programming with Agents: New metaphors for thinking about computation," Ph.D. thesis, MIT.

Wooldridge, M.; N. Jennings; and D. Kinny. 2000. “The Gaia Methodology for Agent-Oriented Analysis and Design,” *Journal of Autonomous Agents and Multi-Agent Systems*.

Chapitre 3

CoCoA : simulation de comportements rationnels.

Mon second projet au sein de l'équipe SMAC est le projet CoCoA, pour Collaborative Cognitive Agents. Ce projet est né à l'occasion d'un contrat PRIAMM avec la société éditrice de jeux vidéos Cryo Interactive.

Son objectif est la proposition d'un modèle permettant la simulation de comportements rationnels pour des agents évoluant dans des environnements situés. Modéliser des comportements réalistes devient très délicat dès que les interactions deviennent complexes. Il est notamment difficile de réutiliser (au sens du Génie Logiciel) les interactions antérieurement définies et difficile d'adapter le comportement de l'agent aux changements externes (l'environnement) et internes (les capacités de l'agent). Il devient alors capital de séparer le modèle d'interactions des interactions elles-mêmes. L'intérêt du monde des jeux vidéo pour une telle thématique est évidemment d'accroitre le réalisme des univers de jeux. Les entités qui y apparaissent doivent manifester des comportements les plus réalistes possibles. Cependant, cette problématique ne se limite pas aux jeux vidéo mais concerne en fait les simulations dans lesquelles la notion de comportement individuel a un sens et les simulations centrées individus en général. Les domaines d'application sont multiples, que ce soit des applications éducatives, le cinéma, ou les simulations de situations d'urgence (Querrec, Reignier, and Chevaillier 2001).

D'autres projets internationaux effectuent des travaux similaires sur les comportements d'agents pour les jeux vidéos. On peut notamment citer les projets Excalibur (Nareyek 2000), Soar (Laird and Duchi 2000, Laird and Rosenbloom 2005) et Cog (Brooks and al. 1999). Cependant, l'arrêt du premier vient d'être annoncé, le second a une approche plus réactive et proche de celles des scripts, enfin l'objectif du troisième est autre, plus ambitieux et à nettement plus long terme, puisqu'il vise à construire un robot humanoïde intelligent. Notre approche est cognitive. L'objectif de nos travaux est d'offrir un outil de modélisation générique de comportements d'agents pour les concepteurs de simulations en insistant sur la séparation du déclaratif et du procédural pour une meilleure réutilisabilité. Nous nous attachons notamment à rendre le moteur comportemental de l'agent indépendant des interactions que celui-ci peut être amené à exécuter.

Dans le cadre de ce travail, nous avons conçu un modèle original basé sur une dualité des interactions possibles entre les agents. Celui-ci offre une grande diversité de type d'agents, et un comportement adaptatif en fonction de leur environnement et de leur capacité. Un système de planification est fourni à chaque agent pour décider de l'action à effectuer en fonction du contexte.

Notre travail a abouti à la réalisation d'une plate-forme logicielle opérationnelle qui nous permet maintenant de concevoir des applications de simulation avec des interactions complexes. Plus récemment nous avons également orienté nos travaux sur la gestion d'équipes d'agents rationnels.

Enfin, précisons que ce qui nous intéresse est l'exécution de la simulation et non son résultat. Ainsi, à la différence de simulations basées sur des équations mathématiques, nous ne cherchons pas à connaître l'état d'un système à l'issue de la simulation mais nous voulons examiner le déroulement de la simulation "pas à pas". Ce sont les comportements individuels des agents qui sont simulés et la simulation résulte de la somme de ces comportements observés. Il s'agit donc de simulations *centrées-individus*. Un telle approche permet des applications différentes des simulations orientées "population" ou de celles où ce sont les interactions sociales qui sont étudiées (Deffuant, Ferrand, Bernard, and Azembourg 1999). De plus, la progressivité de leur déroulement permet d'apporter un niveau explicatif au résultat de la simulation dans la mesure où la simulation peut être suivie action par action, et donc justifiée pas à pas.

Je commencerai par discuter de la position de cette thématique par rapport à sa motivation initiale qui est celle des jeux vidéos et des problématiques de ce domaine. Je présenterai ensuite notre proposition basée sur les interactions avant de décrire le moteur comportemental des agents impliqués dans les simulations. Je parlerai ensuite d'équipes d'agents qui s'appuient sur les mêmes concepts.

3.1 Simulation de comportements et jeux vidéos

Le monde du jeu vidéo est confronté aux problèmes posés par la simulation de comportements de types humains. Il est largement accepté que l'Intelligence Artificielle est le nouveau challenge des jeux vidéos après que le graphisme ait occupé ces dernières années l'essentiel des ressources R&D de cette industrie. Aux difficultés scientifiques et techniques dues à l'Intelligence Artificielle viennent s'ajouter des contraintes économiques. Il s'agit essentiellement du temps requis pour le développement des jeux. Celui-ci dépend dans une large mesure de la possibilité de réutiliser des éléments issus de réalisations précédentes. L'apparition de moteurs graphiques et physiques ont permis cela pour le développement des parties visuelles des jeux. Dans le cas de l'intelligence artificielle des personnages, les techniques employées jusque maintenant rendent difficiles, voire impossible, cette réutilisation.

La recherche en intelligence artificielle a, elle aussi, à gagner de cet intérêt du monde du jeu vidéo. Le poids économique de cette industrie en fait un domaine d'application pouvant justifier à lui seul cette recherche. Il suffit de regarder combien l'informatique graphique a pu en profiter ces dernières années pour s'en convaincre. L'intérêt pour l'intelligence artificielle n'est pas seulement économique, les jeux vidéos constituent en effet un champ d'expérimentation idéal pour la simulation de comportements rationnels. Selon John Laird, ils constituent même la "killer application" pour le développement d'une IA de type humain (Laird and van Lent 2000). Les personnnages non joueurs impliqués dans les jeux vidéos doivent en effet être perçus comme des entités autonomes offrant des comportements de plus en plus réalistes. Les comportements proposés doivent correspondre à ce qu'attend le partenaire ou adversaire humain. Les personnages virtuels doivent pouvoir s'adapter au contexte et aux nouvelles situations rencontrées. De plus, ceux-ci doivent tenir compte des autres entités du jeux et la mise en place de stratégies d'équipes est souvent utile. Dans cette optique, les agents doivent amener l'observateur humain à penser

qu'ils se comportent de manière intelligente et rationnelle, c'est-à-dire comme un humain aurait pu se comporter dans une situation similaire pour accomplir ses propres buts. L'objectif ultime est que dans un contexte où se mèlent les joueurs humains et virtuels, il ne soit pas possible de faire la différence entre eux. On arriverait ainsi à une satisfaction partielle du test de Turing. Notamment, un des avantages apportés par les jeux est qu'ils proposent des contextes dans lesquels la réalité est simplifiée. Les mondes simulés réduisent ainsi les comportements possibles et les choix offerts aux personnages à chaque instant simplifiant la tâche qui autrement serait inabordable à moyenne échéance. En ce sens, si test de Turing il y a, il s'agit bien d'un test dégradé.

Un certain nombre de propositions ont été faites concernant les jeux vidéos et les agents (Nareyek 2004) mais la plupart concernent des agents réactifs (Nareyek 1998). Si ils constituent une réponse possible à la simulation de comportements, les agents réactifs offrent cependant des comportements limités influencés essentiellement par le court terme plutôt que par des buts. Leur capacité à réaliser des tâches dépend de leur environnement immédiat plus que d'une réelle volonté. De plus, dans le cadre des jeux vidéos, ces comportements réactifs sont quasi systématiquement mis en place à l'aide de scripts. Les défauts de cette approche sont reconnus et ont été souvent soulignés (Tozour 2002). Les comportements qu'ils permettent sont le plus souvent trop figés et peu adaptatifs. Même si certains essaient de contourner leurs défauts (Ponsen and Spronck 2004), les améliorations apportées ne sont que partielles. De plus l'utilisation des scripts est une des raisons principales pour laquelle il est quasiment impossibles de réutiliser des éléments de simulation de comportements d'un jeu vers un autre. Au contraire, notre proposition, que je détaillerai par la suite, se base sur des agents proactifs, cognitifs, dirigés par des buts.

Le fait que cette thématique se trouve au confluent de plusieurs domaines dont elle cumule les problèmes constitue déjà en soi une difficulté. D'une part on trouve des préoccupations de l'IA "classique" telle que la représentation de la connaissance et sa manipulation, ainsi que la construction de plans d'actions. D'autre part, la naturelle concurrence des mondes simulés ainsi que la nécessité de mettre en place des situations de coordination entre agents et des stratégies d'équipes amène à considérer les problèmes de l'IA distribuée. Enfin, les préoccupations de génie logiciel doivent être présentes afin de permettre une réutilisabilité des différents éléments de comportements entre différentes simulations.

3.2 Approche et contexte

Comme je l'ai annoncé précédemment, les préoccupations sont multiples. Non seulement il s'agit de modéliser et simuler des comportements réalistes, mais il faut aussi que la proposition prenne en compte les contraintes de conception des simulations et de réutilisabilité des éléments d'une simulation à une autre.

Notre approche se veut donc générique. Nous stipulons qu'un même formalisme peut être utilisé pour modéliser et concevoir des comportements réalistes, c'est-à-dire crédibles, destinés à évoluer dans des mondes artificiels. Ainsi dans notre proposition, le moteur comportemental cognitif reste le même d'une simulation à l'aure et les composants comportementaux peuvent être, au moins partiellement, réutilisés. Le principe fondateur est de baser la dynamique des simulations et la représentation des connaissances sur les interactions entre les agent impliqués dans les simulations : certains agents peuvent effectuer des interactions que d'autres agents peuvent

subir. Ces interactions permettent de modéliser les règles qui régissent le monde simulé et constituent les éléments de connaissance réutilisables entre simulations. Le moteur comportemental des agents actifs est en effet le même et produit des comportements qui dépendent du contexte dans lequel l'agent évolue ainsi que des interactions dont il est doté. Ainsi donc, un changement dans les interactions d'un agent animé entraine une adaptation de son comportement pour prendre en comte ces nouvelles capacités.

Les agents mis en œuvre dans ces simulations sont donc situés dans leur environnement. Celui-ci est muni d'un espace euclidien. Il a donc une géographie et les notions de "position", "voisinage", "déplacement", etc. ont un sens et une influence. A tout moment ces agents ont une perception partielle de leur environnement. Cette perception leur permet de se construire une représentation de l'environnement. Cependant ce dernier est dynamique et concurrent, en conséquence les informations dont dispose un agent sur son environnement sont non monotones. Une information acquise par l'agent peut devenir fausse ou non pertinente après un certain temps. Il en résulte que la connaissance qu'a l'agent de l'environnement est généralement incomplète et potentiellement fausse. Le raisonnement de l'agent s'appuie cependant sur *sa* représentation de l'environnement, il peut donc être amené à commettre des erreurs ou à devoir réviser ses intentions en raison de ce décalage entre ses croyances et la réalité.

Nous distinguons dans nos simulations deux types d'agents. Les agents *inanimés* correspondent en fait aux objets qui peuplent le monde simulé. Plus intéressants, les agents *animés* ont des capacités et disposent d'un moteur comportemental. Ce sont les acteurs des simulations. C'est de ces agents dont il était question dans les paragraphes précédents et ce sont essentiellement d'eux dont nous parlerons par la suite. Ils ont généralement des buts et leur moteur comportemental se base sur leurs connaissances et capacités pour proposer un comportement permettant de satisfaire au mieux et "rationnellement" ces buts. Attardons nous quelques instants sur ce "rationnel". Il n'est pas évident de trouver le terme qui qualifie le mieux les comportements que nous cherchons à produire. Le terme "intelligent" semble trop prétentieux et ambitieux, il implique trop de capacités et entraine trop de débats pour que nous le retenions. On sait bien les discussions qui existent autour d'une définition (y en a-t-il "une" d'ailleurs ?) du terme *intelligence artificielle*. C'est pourquoi nous lui préférons les termes de "rationnels" ou "crédibles" pour qualifier les comportements de nos agents, "raisonnables" ou "plausibles" seraient également des possibilités. Ils sont moins connotés mais restent néanmoins subjectifs. Ainsi nous définissons comme *rationnel* un comportement si, à tout moment, la décision d'effectuer une action aurait raisonnablement pu être prise par un humain qui aurait eu les mêmes informations que l'agent et qui disposerait des mêmes capacités. L'évaluation de ce critère ne peut donc être faite que par des juges extérieurs à la simulation.

Une conséquence évidente de ce point de vue est que nous ne cherchons pas à obtenir des comportements optimaux, que ce soit en terme de déplacements ou de nombre d'actions effectuées par exemple. Ainsi nous ne visons pas l'obtention du "meilleur" comportement (et encore faudrait il définir le critère d'évaluation). Ce qui importe le plus c'est d'éviter les comportements aberrants, c'est-à-dire manifestement non raisonnables. Il est probablement important de rappeler maintenant que dans le type de simulations que nous visons, les comportements sont exécutés "pas à pas" et qu'il est donc possible de suivre en détail la démarche de l'agent. Ainsi l'état final de l'environnement à l'issue de la simulation n'est pas le plus important. Il importe plus de pouvoir observer et étudier le déroulement du comportement individuel des agents.

Ces derniers points : rationnalité des comportements observés et exécution pas à pas, prennent une importance particulière du fait que nos environnements de simulation sont *situés*. Pour pouvoir exécuter les actions commmandées par leur comportement et interagir avec les autres éléments de la simulation, les agents vont en effet être amenés à se déplacer dans l'environnement. Dans une exécution "pas à pas" ces déplacements vont constituer une part importante du comportement observé et représenteront donc un facteur majeur pour évaluer la rationnalité du comportement, d'autant plus qu'ils sont particulièrement visibles pour l'observateur extérieur. Le caractère situé de nos simulations va donc avoir un impact sur les plans d'action des agents. A la séquence d'actions calculée pour résoudre les objectifs de l'agent vont venir s'ajouter les déplacements nécessaires pour atteindre les positions de l'environnement concernées dans le cadre de la simulation. Evidemment pour un même objectif et une même séquence d'actions, les déplacements nécessaires varieront pour des environnements différents. D'autre part, il est possible que, pour pouvoir être exécuté, un déplacement requiert l'exécution d'autres actions qui devront être intégrées à la séquence initiale. Ainsi on peut distinguer ce que nous appelons le *plan abstrait* du *plan d'exécution*. Le premier correspond au plan d'actions dans lequel les positions relatives ou absolues des agents ne sont pas prises en compte, il est indépendant d'un environnement situé. Le second correspond au plan dû à l'exécution dans un environnement particulier.

Le plan abstrait fournit les actions permettant la résolution effective des objectifs. C'est lorsque l'action a que doit exécuter un agent se révèle être une interaction avec un autre agent de la simulation, et c'est souvent le cas, que le caractère situé intervient. Le plus souvent des contraintes de proximité entre les deux agents s'appliquent afin que l'interaction puisse effectivement avoir lieu. En conséquence il est fort possible qu'avant de pouvoir exécuter son (inter)action, l'acteur devra se déplacer pour s'approcher de sa cible. Ce déplacement d devra donc être intégré au plan pour permettre le déroulement de la simulation. Le plan initial "*exécuter a*", devient donc *exécuter d puis a* (Cf. Figure 3.1).

Mais cela soulève un nouveau problème lorsqu'un déplacement requiert à son tour une planification pour pouvoir être exécuté correctement. C'est par exemple le cas lorsque ce déplacement amène l'agent à franchir une porte fermée. L'ouverture de cette porte et la ou les actions qu'elle nécessite devront être également intégrées au plan. Le déplacement *d* est donc décomposé en deux sous-déplacements d_1 et d_2 interrompus par l'ouverture de la porte. On en arrive donc au plan d'exécution *exécuter* d_1*, ouvrir la porte, exécuter* d_2 *puis a*. Les choses se complexifie encore un peu plus lorsque l'ouverture de la porte n'est pas atomique et que la résolution de cet objectif nécessite à son tour une planification. Planification qui entraine la production d'un plan abstrait et de son plan d'exécution contenant ses propres déplacements qui ont un impact sur ceux déjà planifiés. C'est par exemple le cas lorsque la porte en question est cadenassée et que l'agent doit donc préalablement prendre la clé nécessaire et donc se déplacer jusque celle-ci avant d'aller à la porte. Le sous-plan préalable à l'ouverture de la porte est donc *exécuter le déplacement* d'_1*, prendre la clef, exécuter* d'_2*, décadenasser la porte*. Il remplace le déplacement d_1 qui n'a plus lieu d'être. Le schéma récursif de ce raisonnement est évident et l'on peut imaginer que la récupération de la clef engendre de nouvelles actions. On le voit sur ce petit exemple, un plan abstrait initial simple comme l'était *exécuter a* se voit fortement enrichi du fait du contexte situé. Ce point est également repris dans la publication (Devigne, Mathieu, and Routier 2004). Le mémoire de DEA de Damien Devigne s'intéressait déjà à ce problème (Devigne 2003).

Les travaux sur les robots mobiles font eux aussi intervenir une planification pour la résolution d'objectifs et un déplacement. Il est clair que ces robots évoluent en environnement situé. Dans ces travaux, planification et déplacement sont cependant traités séparément par des

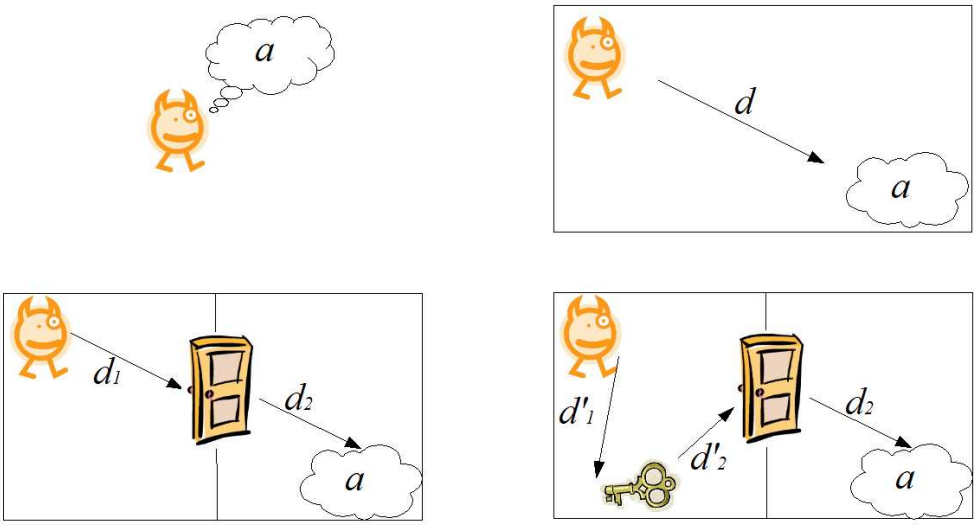

Figure 3.1: Influence du situé sur la planification. En haut à gauche, le plan abstrait. En haut à droite, dans un contexte situé le déplacement doit être prévu pour atteindre l'endroit d'exécution de l'action. En bas à gauche, l'environnement impose une planification, ici ouverture d'une porte. En bas à droite, cas où l'agent sait - ou croit - que la porte est cadenassée et doit donc prendre la clef avant de s'y rendre. (les actions *ouvrir* et *décadenasser* n'apparaissent pas sur les figures)

modules différents. On retrouve donc d'une certaine manière notre plan abstrait et le plan de d'exécution/déplacement. Cependant dans ces travaux, l'environnement, même si il est constitué d'obstacles inconnus qu'il faut détourner ou légèrement déplacer, n'amène pas à devoir planifier des actions permettant les déplacements comme je l'ai décrit ci-dessus. De plus dans ces travaux, la demande de rationnalité n'est pas la même que la notre, on demande que le robot accomplisse sa tâche, qu'il ait parfois eu passagèrement des choix d'actions "bizarres" importe peu. Pour ces raisons, si il y a une certaine proximité, la problématique de ces travaux diffère de la notre sur ces points et nous amène à faire des propositions différentes notamment avec l'intégration des déplacements dans la planification.

Après avoir positionné le contexte et les contraintes qui lui sont inhérentes, je vais maintenant détailler notre approche basée sur les interactions avant de présenter le modèle d'agent cognitif que nous proposons pour la réalisation de nos simulations.

3.3 Des agents définis par leurs compétences et des simulations orientées interaction

Avant de détailler nos travaux dans les sections suivantes, je vais en présenter les grandes lignes qui reposent essentiellement sur la notion d'*interaction*.

L'environnement mis à part, tous les éléments de la simulation sont appelés *agents*. On y distingue les objets du monde simulé de ses acteurs. Les premiers sont appelés *agents passifs* ou *inanimés* et les seconds *agent actifs* ou *animés*.

Mais le point central de notre proposition est la notion d'interactions. Celles-ci constituent la base de la représentation de connaissance dans la mesure où elles définissent les lois qui régissent le monde simulé. Les interactions décrivent en effet les actions qui peuvent être accomplies et leurs effets.

Ainsi, les propriétés principales des agents sont des listes d'interactions. Celles-ci sont appelées *peut-subir* et *peut-effectuer* (Cf. Figure 3.2), la seconde n'existant que pour les agents animés. Ces propriétés ont pour valeur des listes d'interactions. La première correspond aux interactions dont l'agent peut être la cible, la seconde à celles dont il peut être l'acteur.

```
agent        := agent-passif | agent-actif
agent-passif := { ( "nom", symbole),
                    ("peut-subir", {interaction* }),
                    propriété*}
agent-actif  := agent-passif ∪
                    {("peut-effectuer", {interaction*}),
                    ("buts", but*),
                    ("mémoire", environnement),
                    ("moteur", moteur)}
propriété    := (nom_propriété, Valeur)
```

Figure 3.2: Définition d'un agent

Les interactions que peut effectuer un agent animé représentent en fait les capacités ou compétences de cet agent. C'est bien la la valeur de sa propriété *peut-effectuer* qui détermine ce que l'agent peut accomplir et donc quel(s) rôle(s) il peut jouer dans la simulation. On retrouve ici une approche similaire à celle que nous avons développée dans le cadre du projet MAGIQUE présenté au chapitre précédent.

Ce sont les interactions qui sont à la base de la dynamique de nos simulations. Le principe général est la mise en correspondance d'un agent *acteur* et d'un agent *cible*, le premier exécutant l'interaction sur le second (cf. Figure 3.3). Ce principe peut être décrit ainsi :

$$a \in Actif, t \in Agent, \text{si } \exists i \in a.\textit{peut-effectuer}() \cap t.\textit{peut-subir}(), \text{alors } a.executer(i,t)$$

L'interaction à exécuter est en fait déterminée par le *moteur de comportements* de l'agent a afin qu'elle lui permette de réaliser son *but*. L'agent a doit alors rechercher la cible t qui convient effectivement.

3.4 Les interactions

Les interactions définissent les lois qui régissent le monde simulé. La connaissance y est exprimée de manière déclarative. De ce fait, le plus souvent, une interaction n'est pas attachée à une simulation en particulier dans la mesure où elle représente une connaissance "universelle". Cela constitue une contrainte au niveau de la représentation des interactions en obligeant le moteur

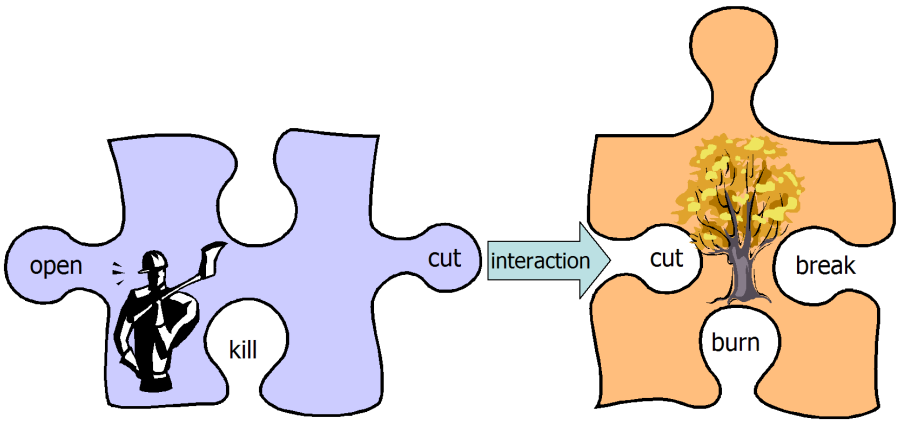

Figure 3.3: Les agents sont décrits en terme d'interactions qu'ils peuvent subir ou effectuer, ici un agent actif *bucheron* peut effectuer les interactions *open* et *cut* et subir *kill*, alors qu'un agent passif *arbre* peut subir *cut*, *break* et *burn*. Le principe central de la simulation est l'assocation d'un acteur effectuant une interaction et d'une cible la subissant, comme l'interaction *cut* ici.

de raisonnements qui les manipule à être générique. Mais de ce fait, la plupart de ces interactions peuvent être réutilisées dans différentes simulations et la généricité imposée au moteur des agents permet de satisfaire l'un des objectifs évoqués précédemment.

```
acteur          := l'agent qui réalise l'interaction
cible           := l'agent qui subit l'interaction

interaction     := (nom, garde, condition, action)
nom             := symbole
garde           := d op nombre
d               := la distance entre l'acteur et la cible
op              := = | > | <
condition       := test_propriete | primitive(args)
test_propriete  := { acteur | cible }.nom_propriété op Valeur
action          := affect_propriété | primitive
affect_propriété:= { acteur | cible }.nom_propriété = Valeur
primitive       := { acteur | cible }.nom_primitive(args)
nom_primitive   := Symbole
nom_propriété   := Symbole
```

Figure 3.4: Définition d'une interaction

Nous avons déjà évoqué il y a quelques lignes les agents *acteur* et *cible* qui caractérisent une interaction. Outre son nom, la description d'une interaction se fait classiquement en trois parties (cf. Figure 3.4) :

la condition Elle permet de tester le contexte d'exécution courant. Cette partie consiste essentiellement à tester les valeurs de propriétés de la cible et de l'acteur.

la garde Il s'agit ici d'exprimer des conditions plus générales d'applicabilité de l'interaction. En particulier, les aspects liés à l'aspect situé des simulations est exprimé dans cette garde et non pas dans la partie *condition*. Le plus souvent il s'agit d'exprimer que l'acteur doit être suffisamment proche de la cible pour que l'interaction puisse être effectuée. C'est pour cette raison que la *garde* est séparée de la *condition*. Dans un contexte non situé, on ne trouverait que cette dernière partie alors que les conditions de la *garde* sont à l'origine des déplacements dans le plan. Grâce à cette garde, il n'est pas nécessaire d'exprimer explicitement les déplacements dans l'interaction. L'expression déclarative des contraintes spatiales entrainera automatiquement la planification de ces déplacements si ils s'avèrent nécessaires.

l'action Elle décrit les conséquences de l'exécution de l'interaction : changement des valeurs de propriétés de l'acteur ou de la cible, effet de bord sur l'environnement, etc.

Illustrons cette description avec l'exemple de l'interaction *ouvrir* (cf. Figure 3.5). La *condition* exprime ici que la cible ne doit pas déjà être ouverte pour pouvoir subir *ouvrir*. Si elle semble triviale, cette connaissance est néanmoins nécessaire si l'on veut éviter des aberrations telles qu'un agent cherchant désespérément à ouvrir une porte ouverte... L'*action* exprime quant à elle le changement d'état de la cible une fois que l'interaction a été exécutée. Enfin, la garde exprime que l'acteur doit être à côté de la cible pour pouvoir exécuter celle-ci. Cette connaissance évite que nos agents deviennent télékynésiste... Elle n'aurait pas d'importance dans un contexte non située.

$$\textit{open:} \begin{cases} \texttt{condition} = \text{“}\texttt{target.opened} = \texttt{false}\text{”} \\ \texttt{guard} = \text{“}\texttt{d} < 1\text{”} \\ \texttt{action} = \text{“}\texttt{target.opened} = \texttt{true}\text{”} \end{cases}$$

Figure 3.5: L'interaction *ouvrir*. L'expression de la connaissance est déclarative, en particulier la nature de la cible n'est pas explicite.

Comme on peut le constater à la lecture du code de cette interaction, aucune référence n'est faite à un type particulier de cible. Celle-ci peut aussi bien être une porte qu'une boîte de conserve ou une fenêtre ou tout autre agent “ouvrable”. C'est essentiellement ce découplage entre le code de l'interaction et les agents auxquels elle s'applique qui favorise la réutilisabilité de ces éléments de connaissance entre différentes simulations.

Cependant cette généricité est mise à mal lorsque l'on considère que certaines cibles peuvent imposer des conditions particulières pour une même interaction, ou plutôt pour un même concept exprimé par une interaction. Pour poursuivre l'exemple précédent, considérons le cas de portes qui peuvent être cadenassées. Celles-ci peuvent bien sûr toujours subir l'interaction *ouvrir* mais pour pouvoir exécuter cette interaction, il est nécessaire qu'une condition supplémentaire soit vérifiée : la porte ne doit pas être cadenassée. Cependant, conceptuellement, le plan d'un acteur devant franchir cette porte sera fondamentalement le même : il doit *ouvrir* la porte. Ce n'est que le traitement de cette ouverture qui changera avec la nature de la porte selon qu'elle est cadenassable ou pas. Afin de garder le découplage entre les interactions et les agents ainsi que la généricité du moteur, créer une nouvelle interaction comme *ouvrir-si-cadenassable* ne paraît pas pertinent, ni judicieux. C'est pourquoi notre proposition consiste en la possibilité pour les cibles de spécifier par extension des conditions spécifiques. Lors de la construction du plan de l'acteur, la

cible “annonce” à l’acteur les contraintes supplémentaires. Pour l’acteur il n’y a toujours qu’une seule interaction générique. Par exemple, dans le cas d’une *porte-cadenassable* l’interaction *open* voit ses conditions enrichies de la condition *target.isLocked = false*. Ainsi lorsqu’un acteur a pour objectif d’*ouvrir une porte*. Il récupère la condition qui apparaît à la figure 3.5 pour une porte “simple”, mais récupère la condition *target.isOpen = false et target.isLocked = false* dans le cas d’une *porte-cadenassable*. Cette condition amènera le cas échéant l’agent acteur à chercher à décadenasser cette porte avant de l’ouvrir.

Ce mécanisme offre aux concepteurs de la simulation la possibilité d’ajouter de nouvelles cibles qui en spécifient une existante, prenant ainsi en compte certaines particularités du monde simulé sans remettre en cause l’existant. Cette extension ne nécessite en effet pas non plus que soient modifiés les acteurs éventuels, ou plutôt leur moteur de comportements. Cette flexibilité facilite la conception des simulations et favorise la réutilisation des interactions et par conséquence des agents d’une simulation à une autre.

L’article (Mathieu, Picault, and Routier 2003) présentait cette notion qui était également le thème du mémoire de DEA de Patrick Tessier (Tessier 2002).

3.5 Les agents

Comme je l’ai écrit précédemment nous distinguons dans nos simulations les agents *inanimés*, qui correspondent aux objets des mondes simulés, des agent *animés* qui sont les acteurs de ces mondes. Dans les deux cas, ils sont définis par leurs caractéristiques (nom, couleur, énergie, etc.) et les interactions qu’ils peuvent subir. Ce qui les distinguent c’est que les acteurs animés sont en plus définis par les interactions qu’ils peuvent effectuer et disposent d’un moteur de comportements qui leur permet d’accomplir leurs objectifs. Le cycle de fonctionnement de l’agent est le suivant :

1. perception de son environnement, c’est la phase de prise d’information dans un environnement situé ;
2. mise à jour de la base de connaissance ;
3. choix d’une action exécutable, cela implique le calcul d’un plan et la sélection d’une action proposée par ce plan ;
4. exécution de l’action choisie dans l’environnement.

Différents éléments (cf. Figure 3.6) composent donc ce moteur de comportements.

Le *module de perception* permet à l’agent de percevoir son environnement. Ces informations sont transmises au *module de mise à jour* dont la fonction est de prendre en compte l’impact de ces informations. Notamment c’est ce module qui met à jour la base de connaissances de l’agent. Celle-ci comprend les interactions que l’agent peut effectuer, mais surtout sa *mémoire*. Cette mémoire correspond à l’image qu’a l’agent de son environnement. Il s’agit donc d’une version dégradée, car incomplète et non nécessairement à jour, de cet environnement, y compris des autres agents qu’il contient. C’est sur cette mémoire que se base le *moteur de planification* pour établir les plans d’action destinés à lui permettre d’accomplir ses objectifs. Le moteur assure que ces plans sont corrects en accord avec la mémoire de l’agent. Cependant, ces plans peuvent être remis en cause par les informations apportées par le module de perception. Dans ce cas une

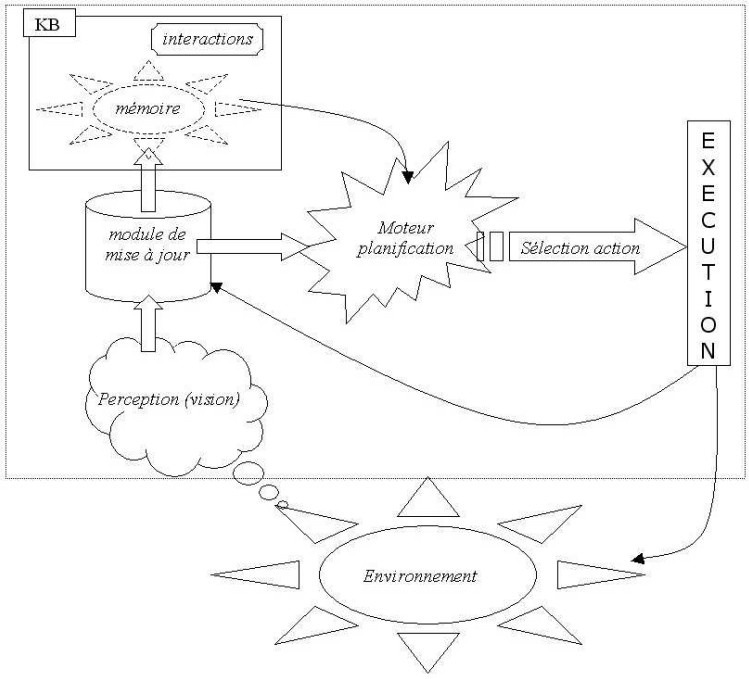

Figure 3.6: Les différents éléments du moteur comportemental.

replanification partielle des plans est réalisée allégeant le travail de planification. Un *mécanisme de sélection d'action* propose ensuite la prochaine action à exécuter par l'agent. Ce choix dépend de priorités de l'agent et du contexte dans lequel il évolue. Si nos agents sont proactifs, cela ne les empêchent en effet pas de manifester des comportements réactifs. L'agent essaie ensuite d'exécuter l'action proposée dans l'environnement. Cela peut s'avérer impossible si la mémoire de l'agent ne correspondait pas à l'état réel de l'environnement. Dans ce cas, le module de mise à jour prendra en compte la nouvelle information et la répercutera.

Le moteur de planification fonctionne de manière assez classique selon un mécanisme type chaînage arrière sur les interactions. Les difficultés principales naissent de la nécessité de prendre en compte l'aspect situé des simulations c'est-à-dire d'inclure les déplacements dans le plan. J'ai déjà évoqué cette problèmatique précédemment lors de la discussion sur les plans abstraits et concrets. Un problème supplémentaire à prendre en compte est la nécessité pour l'acteur de trouver les cibles qui lui permettront d'accomplir les actions que lui commande son plan. Ainsi, grâce aux interactions qu'il peut effectuer, il peut connaître l'existence a priori de telle cible sans savoir nécessairement où elles se trouvent dans l'environnement. A nouveau on voit apparaître une spécificité due au type des simulations visées et notamment à leur aspect situé.

Ces travaux ont été publiés dans (Devigne, Mathieu, and Routier 2005b).

3.6 L'environnement de conception de simulations

Toujours soucieux de mettre en application les concepts que nous proposons, et cela semble encore plus indispensable dans cette thématique, nous avons développé un environnement de conception ("IDE") et d'exécution de simulations (cf.Figure 3.7).

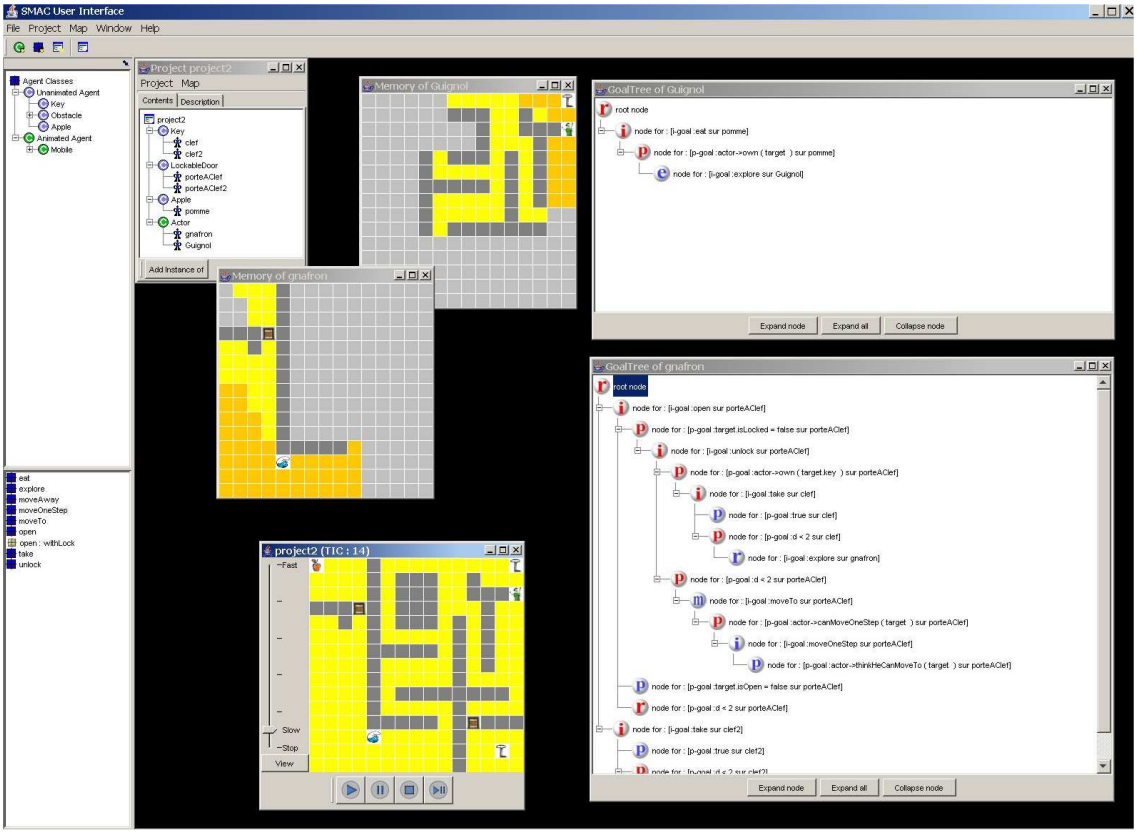

Figure 3.7: Environnement de conception d'applications de CoCoA : on y voit sur la gauche les classes d'agent (en haut) et interactions (en bas) déjà créées. La zone centrale montre une simulation en cours d'exécution. L'environnement y apparaît ainsi que les deux fenêtres montrant la mémoire de chacun des agents actifs avec sa vision partielle de cet environnement. Les arbres de raisonnement des agents sont également visibles.

Celui-ci permet aux concepteurs de simulations de créer des bibliothèques d'interactions et d'agents qui pourront être réutilisés de simulations en simulations. Pour les agents il s'agit plus de patrons (pour ne pas dire classes) partiellement initialisés, ceux-ci peuvent alors être "instanciés" pour créer les agents qui prennent effectivement part aux simulations. Il est également possible de concevoir l'environnement géographique de la simulation et d'y placer les agents auxquels on affecte des buts.

Une fois créée une telle simulation peut être exécutée. Il est alors possible pour chacun des agents animés de suivre la progression de son raisonnement. Ainsi, la mémoire de l'agent peut être observée et son plan d'action visualisé sous forme arborescente. Le concepteur de la simulation peut également intervenir pour modifier telle ou telle valeur dans l'environnement afin

d'étudier l'impact de cette modification sur le comportement des agents. Cet environnement constitue donc également un contexte de tests pour les simulations et les comportements créés.

Il existe d'autres plate-formes de réalisation de simulations centrées individus. On peut citer au niveau international SWARM (Swarm 1994-2005), REPAST (Repast 2003) et au niveau national MADKIT (Ferber, Gutknecht, and Michel 2000) et CORMAS (Bousquet, Bakam, Proton, and Page 1998). Cependant leurs objectifs ou approches diffèrent de la notre. En particulier, le plus souvent, ces plate-formes ne proposent pas (ou n'imposent pas) un modèle d'agents, c'est par exemple le cas de SWARM ou MADKIT. Elles fournissent plutôt un ensemble d'outils ou de primitives facilitant la mise en place de simulations mais laissent l'utilisateur définir son modèle d'agent. Leur approche est en ce sens proche de celle que nous avons adoptée dans le projet MAGIQUE. Ensuite CORMAS, par exemple, a des objectifs différents. Cet environnement a été conçu pour modéliser la gestion des ressources renouvelables et décrire les coordinations entre individus ou entre groupes exploitant des ressources communes. Ainsi notre plate-forme, et plus généralement notre projet, se différencie de ceux qui viennent d'être cités dans la mesure où il vise la modélisation de comportements et propose (et impose) pour cela un modèle d'agent.

3.7 Equipes d'agents

Les travaux décrits précédemment permettent la réalisation de simulations impliquant plusieurs agents évoluant en concurrence dans leur environnement, ces agents cohabitant sans coopérer. C'est pourquoi, de manière naturelle, nous avons été amenés à nous pencher sur la possibilité d'intégrer la notion d'équipe d'agents dans nos simulations. Ces travaux constituent l'un des axes de la thèse en cours de Damien Devigne que je co-encadre avec Philippe Mathieu.

Ayant toujours à l'esprit notre thématique principale d'application que sont les jeux vidéos, notre premier point de recherche a été de considérer des équipes avec chef. Cette configuration se retrouve en effet assez souvent dans les jeux vidéos. Cependant nous ne voulions pas modéliser des équipes au raisonnement totalement centralisé. En effet, les agents impliqués dans ces équipes correspondent à ceux décrits précédemment, c'est-à-dire à des agents proactifs, disposant de capacités de raisonnement. Ainsi il n'était pas question pour nous de réduire nos équipes à un chef qui décide de la moindre action de ses équipiers dont le rôle se réduirait à celui d'exécutants décérébrés.

De plus, nous souhaitions évidemment conserver la représentation des connaissances et notre modélisation à base d'interactions. Or dans ce cadre, nous disposons d'un moteur de comportements qui fonctionne de manière satisfaisante pour gérer les plans d'action individuel.

Notre problème a donc été : comment adapter notre moteur de planification pour produire des plans d'équipe dans lesquels les agents gardent une part d'autonomie.

La solution que nous avons trouvée découle principalement du principe que nous pourrions énoncer comme ceci : "Le chef a besoin de savoir ce que ses équipiers savent faire mais pas nécessairement de savoir comment ils le font". Ainsi le chef d'une équipe doit pouvoir affecter à ses agents des ordres d'assez haut niveau, charge à eux de les résoudre en fonction de leurs connaissances propres. Pour reprendre un exemple cher à Damien Devigne, dans le cadre de la construction d'une maison, un chef de chantier commandera à son électricien d'installer telle prise à tel endroit, mais ne lui dira pas qu'il doit dénuder les fils pour le faire. C'est en effet au chef de savoir où il faut installer les prises, mais comment celles-ci doivent être installées relève de la compétence de l'électricien. Le chef n'a même pas à avoir a priori cette connaissance.

Pour construire nos plans d'équipe, la réalisation de ce principe est obtenu en réutilisant exactement le moteur de planification individuel, mais en ne confiant au chef qu'une partie de la connaissance de ses équipiers. Cette connaissance correspond à la description en termes de fonctionnalités, et donc d'interactions, des rôles des équipiers. Ces interactions sont cependant un peu particulières, non pas dans leur structure puisqu'elles obéissent totalement au schéma présenté, mais dans leur contenu puisque leurs parties condition et garde sont masquées au chef, qui n'en voit donc que la partie action, c'est-à-dire le résultat. Cela revient au fait que le chef sait ce que l'interaction de son équipier peut amener (l'action) mais pas ce qu'il faut pour la réaliser (les conditions). Cette partie masquée est remplacée simplement par la valeur *vrai*, amenant naturellement le moteur de planification à arrêter son chaînage.

Ainsi lors de sa planification d'équipe, le plan du chef s'arrête sur des tâches correspondant à des compétences d'équipier. Le chef distribue alors les tâches aux équipiers qui les réalisent, ce qu'ils peuvent faire puisqu'ils disposent quant à eux de la connaissance dans son ensemble.

Cette solution permet dont la modélisation d'équipes dirigées par un chef qui est en charge d'établir les grandes lignes du plan de l'équipe et de distribuer les tâches à ses équipiers. Ceux-ci exploitent leurs capacités de raisonnement pour traiter de manière autonome ces objectifs. Ce travail a donné lieu aux publications (Devigne, Mathieu, and Routier 2005c, Devigne, Mathieu, and Routier 2005a). Il constitue la thématique principale de la thèse en cours de Damien Devigne (Devigne 2003-en cours) que je co-encadre.

Ce projet est encore en cours. Nous étudions notamment également la mise en place de comportements d'équipe sans chef. Au niveau du moteur comportemental, nous avons récemment travaillé sur le mécanisme de sélection d'action. Nous travaillons sur un modèle permettant à la fois d'introduire des comportements de type réactifs tenant compte de l'environnement immédiat sans perdre de vue les objectifs à plus long terme. Le mécanisme que nous cherchons à mettre en place permet également de simuler des caractères différents d'un agent à un autre. Ces travaux non encore publiés ont été commencés dans le cadre du Master Recherche de Julien Acroute (Acroute 2005).

Bibliographie

Acroute, J.. 2005. "Apport réactif aux comportements cognitifs de la plateforme de simulation CoCoA," Master's thesis, Master Recherche mention Informatique de l'Université de Lille 1, Co-dirigé avec le Professeur Philippe Mathieu.

Bousquet, F.; I. Bakam; H. Proton; and C. L. Page. 1998. "Cormas: common-pool resources and multi-agent Systems," in *Lecture Notes in Artificial Intelligence*, vol. 1416, 826–838.

Brooks, R. A.; and al.. 1999. "The COG Project: Building a Humanoid Robot," in *Computation for Metaphors, Analogy, and Agents*, vol. 1562 of *LNCS*, 52–87. Springer.

Deffuant, G.; N. Ferrand; S. Bernard; and D. Azembourg. 1999. "Modèles multi-agents de la diffusion de l'adoption de mesures agri-environnementales dans des réseaux sociaux d'agriculteurs : décision multicritère et abstraction décroissante des interactions sociales," in *Actes des JFI-ADSMA'99*.

Devigne, D.. 2003. “Simulation de comportements pour agents rationnels situés et étude du GraphPlan,” Master’s thesis, DEA d’Informatique de l’Université de Lille 1, Co-dirigé avec le Professeur Philippe Mathieu.

Devigne, D.. 2003-en cours. “Modélisation de comportement d’équipes en environnement Multi-Agents,” Ph.D. thesis, Université des Sciences et Technologies de Lille, Co-dirigée avec le Professeur Philippe Mathieu.

Devigne, D.; P. Mathieu; and J.-C. Routier. 2004. “Planning for Spatially Situated Agents,” in *Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT’04)*, ed. by I. Press, 385–388.

Devigne, D.; P. Mathieu; and J.-C. Routier. 2005a. “Gestion d’équipes et autonomie des agents,” in *actes des JFSMA2005*, ed. by Cépaduès.

Devigne, D.; P. Mathieu; and J.-C. Routier. 2005b. “Interaction-Based Approach For Game Agents,” in *Proceedings of ECMS/SCS/IEEE 19th European Conference on Modelling and Simulation. ECMS 2005*, ed. by Y. Merkuryev, R. Zobel, and E. Kerckhoffs, 7056714.

Devigne, D.; P. Mathieu; and J.-C. Routier. 2005c. “Teams of cognitive agents with leader: how to let them some autonomy,” in *Proceedings of IEEE Symposium on Computational Intelligence Games. CIG’05*, ed. by G. Kendall, and S. Lucas, 256–262.

Ferber, J.; O. Gutknecht; and F. Michel. 2000. “The MadKit Agent Platform Architecture,” Discussion paper, W3C.

Laird, J.; and J. Duchi. 2000. “Creating human-like synthetic characters with multiple skill levels: A case study using the soar quakebot,” .

Laird, J. E.; and M. van Lent. 2000. “Human-level AI’s Killer Application: Interactive Computer Games,” in *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence.*, 1171 – 1178. AAAI.

Laird, J. L. J.; and P. Rosenbloom. 2005. “A Gentle Introduction to Soar: 2005 Update,” unpublished, An update to an earlier paper (published in 1996).

Mathieu, P.; S. Picault; and J.-C. Routier. 2003. “Simulation de comportements pour agents rationnels situés,” in *Actes des Secondes Journées Francophones sur les Modèles Formels de l’Interaction, MFI’03*, ed. by A. Herzig, B. Chaib-draa, and P. Mathieu, 277–282.

Nareyek, A.. 1998. “Specification and development of reactive systems,” in *1998 AIPS Workshop*, 7–14, Menlo Park California. AAAI Press.

Nareyek, A.. 2000. “Intelligent Agents for Computer Games,” in *Computers and Games, Second International Conference, CG 2000. LNCS 2063.*, 414–422.

Nareyek, A.. 2004. “Artificial Intelligence in Computer Games - State of the Art and Future Directions,” *ACM Queue*, 1(10), 58–65.

Ponsen, M.; and P. Spronck. 2004. “Improving Adaptive Game AI with Evolutionary Learning.,” in *Proceedings of Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004)*, ed. by S. N. Quasim Mehdi, Norman Gough, and D. Al-Dabass, 389–396.

Querrec, R.; P. Reignier; and P. Chevaillier. 2001. “Humans and autonomous agents interactions in a virtual environment for fire-fighting training,” in *Proceedings of Virtual Reality International Conference (VIRC 2001)*, 57–64.

Repast. 2003. “The Recursive Porous Agent Simulation Toolkit,” `http://repast.sourceforge.net`.

Swarm. 1994-2005. “Swarm Software,” `http://www.swarm.org`.

Tessier, P.. 2002. “Planification et exécution dans un environnement non monotone,” Master’s thesis, DEA d’Informatique de l’Université de Lille 1, Co-dirigé avec Professeur Philippe Mathieu.

Tozour, P.. 2002. “The Perils of AI Scripting,” in *AI Game Programming Wisdom*, ed. by S. Rabin, 541–547. Charles River Media.

Partie II

Sélection d’articles

Chapitre 4

Dynamic Skills Learning : a Support to Agent Evolution

Philippe Mathieu, Jean-Christophe Routier et Yann Secq
Laboratoire d'Informatique Fondamentale de Lille
Cité Scientifique 59655 Villeneuve d'Ascq Cedex
{mathieu,routier,secq}@lifl.fr

Abstract.[1]

In this paper, we show that every agent can be built from an atomic agent through dynamic *skill* acquisition, a skill being a coherent set of abilities. At first, we propose a definition of an *atomic agent* and then we present the skill notion. In our work, since an agent is defined by the set of roles he can play according to the skills he has learned, we can consider that skills are the backbone of agents. We propose that the learning be dynamic and thus agents can effectively evolve during their "life". That means that the roles he plays can change. This approach promotes evolutivity, reusability and modularity.
These concepts have been applied to our own multi-agent system model, called MAGIQUE. It is based on hierarchies of agents (or agents recursively built from agents depending on the vision). This organisation provides an automatic delegation of the exploitation of skills between agents, that contributes to the simplicity and adaptability of MAS building.
An API that implements these ideas has been developed and of the corresponding API that. It provides an easy to use framework to build multi-agent applications where agents effectively dynamically evolve.

4.1 Introduction

The agenthood in an application is often hidden inside the programmer's mind (Shoham 1993, Travers 1996). This has contributed to the rise of various definitions of the *agent* term

[1] Article publié dans *Proceedings of AISB'01, York. ISBN 1 902956 17 0. pp 25-32 . March 2001.*

(Franklin and Grasser 1996), and it is not a difficult task to collect a great amount of different definitions. The intersection between all these definitions is often not empty but they differ, sometimes slightly, by some functionalities that are basic for each model.

Starting from this observation, we want to try to propose a basis above which every other definitions could be settled. Our purpose is to define an *atomic agent* which can evolve dynamically in different ways in order to match the different notions that are introduced.

Our central point is the notion of *skill*. It denotes a coherent set of abilities that can be "given" to an agent. The principle is to start from an atomic agent and then to *teach* him some skills in order to build the wanted agent. The "result" depends on the teached skills and therefore different kinds of agents can be reached. Moreover the teaching can also be done during the agent activity and thus an agent can evolve dynamically, that means that the role he plays changes.

From a programming point of view, this approach promotes reusability and modularity because once a skill has been developed, it can be used many times in different contexts. Indeed, in this vision, a skill can be considered as a software component (for one other approach building agents from components see (Horling and Lesser 1998) for example).

We claim that the different concepts for agents can be developped from that basis. As an illustration we have applied it to our own multi-agent model named MAGIQUE [2] (Bensaid and Mathieu 1995, Bensaid and Mathieu 1997). It is based on a hierarchic (or recursive) vision of a multi-agent systems. MAGIQUE is a multi-agent framework. We insist on the term *framework*: MAGIQUE is not a multi-agent system (MAS) but a support to build MAS. And as this, it allows the programmer to define his own policies for agent and MAS building since MAGIQUE gives to him the primitives to do it. In particular, MAGIQUE is based on the previously mentioned principle of building agent through skill acquisition. And then MAGIQUE provides the tools required to develop multi-agent applications where the agents dynamically evolve through skill acquisition. How the learning is done is the application part. Consequently, the ideas that issues from learning/acquisition works can be effectively put into concrete form with MAGIQUE.

Before we continue, a precision must be made. Throughout the following we are going to use the term "skill learning". This term could lead to confusion. Maybe the terms "skill acquisition" or "skill exchange" would be more fitted. We do not mean with the term "skill learning" that the agent learns from observation or some convergence algorithm. Rather, we mean that the agent has acquire a new skill that has been given to him by another agent (called the teacher, this can or not lose the skill when he gives it).

Thus, we present first a model, where the agents are dynamically built by dynamic skill acquisition and second an application framework, where multi-agent applications can be built according to this model. This provides the tools to perform the dynamic skill exchange and acquisition between agent. Let us note, that since it is a framework, the application part, such as the policies used to decide when a skill must be acquire or not, is left to the user.

The first section presents definitions: skills and agents. In the second one we give a brief description of our model MAGIQUE. The third should give an idea of the corresponding API that provides an easy to use framework to build applications where agents effectively dynamically evolve. Last we write some words about an application example where dynamic evolution of roles through skill is used.

[2] Magique stands for the french "Multi-AGent hiérarchIQUE" that (obviously) means... "hierarchical multi-agent".

4.2 Agent from Atomic Agent

4.2.1 Definitions

Skills

An agent is *one who acts*. In order to perform some activity, he must have the *skill* to do it. We define a skill as follows:

Définition 3 A *Skill* is a coherent set of abilities.

Then, a skill is a set of functionalities that can be exploited by an agent. We want everything being defined in term of skills, even the way an agent manages his skills. One could speak of meta-skills in this case, and even these should be able to dynamically evolve as explained before.

From a more pragmatic view, a skill should be seen as a software component whose public interface constitutes the abilities a capable agent can use.

The granularity and degree of complexity of a skill can not be definitively stated. The ability to parse an XML message or the ability to add two integers can each represent a skill although their complexities will probably be considered as being of different levels.

Moreover, whether you must group the four basic operations (addition, subtraction, multiplication and division) of integers in one skill or separate them in four skills, can not be definitively established. However it should be possible to reach a general agreement by saying that XML parsing and addition should be set in two different skills. That is what the "*coherent*" means in the previous definition.

Stated that to a skill should correspond only one ability (or conversely) could seem reasonable, but answer is not so easy, and in any case this will probably not withstand the confrontation with the reality of programmers... The problems that arise here are some common ones encountered in software engineering, and in OOP in particular, concerning (object) decomposition.

Agents

Now, the following definition should be able to receive a rather general agreement:

Définition 4 An *Agent* is an entity gifted of skills.

Any property commonly linked to the agent notion – such as proactivity, interactivity, intelligence, etc. – can indeed be expressed in term of *skills*. Therefore, it seems reasonable to say that all agent definitions, as those cited in (Franklin and Grasser 1996), can be obtained from this one: the differences between two given agent definitions come from the basic functionalities assigned to the agents, that is from their skills.

However, for that reason, this definition can also be taxed as being too nebulous since it allows many interpretations depending on the skills attached to the agent. Thus we are going to precise it by stating a minimal set of skills.

We assert that only two prerequisite skills are necessary and sufficient to define an *atomic agent* from which every other agent definition can be established. These skills are: first, one skill that allows the agent to acquire new skills, and second, one skill for communications (with other agents – who could be human or software agents).

These skills are indeed *necessary*. Without the "skill acquirement" skill, such an agent is just an empty shell unable to perform any task. Without the interaction skill, an agent is isolated from the "rest of the world" and therefore loses any interest. Moreover without communication an agent will not be able to learn new skill from others.

They are *sufficient* too since it suffices to an agent to use his interactive skill to get in touch with a gifted agent and then to use his acquirement skill to acquire some new talent. Then every aptitude can be given to an agent through learning from a "teacher".

Consequently we propose the following new agent definition:

Définition 5 An *atomic agent* is an entity gifted of two skills: one to interact and one to acquire new skills. An *agent* is an atomic agent who has acquired skills from communications.

We claim that every agent proposed by the various existing definitions match this definition. Thus an agent using KQML is an agent gifted of such a "KQML understanding" skill; another who can encrypt his messages is an agent gifted of a code/decode skill (whatever algorithm is used), etc. At a more conceptual level, the notions of role and group, that are the core of the Aalaadin model (Ferber and Gutknecht 1998), can also be translated in term of skills and this model could then be described in such terms (you must have skills to join or leave a group, to communicate inside a group, etc.).

Let us note that this is not the skills themselves that are important but rather their functionalities. Thus, we can imagine that the interactive skill used by an agent can change during his life cycle, because he learns a new one for example. The important point is not that the agent have a particular interactive skill, but that the agent always has the ability to communicate (and similarly for the "learning" skill).

With no intention to enter into a philosophical debate, we can meanwhile note that this definition applies to the "human agent" who, since he can communicate and is able learn new knowledge, can evolve step by step through interactions with others (human and environment) agents. And communication (through only the five senses at the beginning and, after some times of evolution, with the contribution of the language later) and learning abilities seems to be effectively its sole attributes at the beginning of his life, and they allows a complete and variable education. Moreover, people who suffers from communication or learning troubles have difficulties to join the community (the "human MAS").

4.2.2 Agent Education

Thus we can consider that all agents are at birth (or creation) similar (from a skill point of view): a shell with only the two above previously described skills.

Therefore the differences between agents issue from their education, i.e. the skills they have acquired during their "existence". These skills can either have been given during agent creation by the programmer, or have been dynamically learned through interactions with other agents[3].

[3]now if we consider the programmer as an agent, the first case is included in the second one

4.2.3 Advantages

This paradigm, dynamic construction of agent from skills, has several advantages from a programming point of view.

development becomes easier Building an agent is teaching him some skills. Then agent programming is "reduced" to skill programming, but once a skill have been developed it can be used in different contexts.

Skills can be seen as *software components*, with all the advantages linked to this notion : modularity, reusability, etc.

efficiency An agent can decide to delegate the achievement of some task to another one (if this one accepts it). But this has a cost (due to communication for example) and is dependent upon the "good will" of the other. That's why in some case, if he has to often perform the same task, an agent can "prefer" to learn a skill and thus remove the need to delegate its achievement.

On the other side, if an agent "feels" he is overwhelmed by requests from others to exploit one of his skills, he can choose to teach it to some other agent(s) (those could be agent he has created and teached in this specific goal), to lighten his burden.

robustness If for some reason an agent has to disappear from the MAS and he owns some critical skill, he can teach it to some other agent in the MAS and thus warrant the continuity of the whole MAS.

autonomy and evolutivity During his "life" a given skill of an agent can evolve and be improved, and new skills can be added. Then the agent increases his abilities and his autonomy.

dynamic evolution When a skill of an agent need to be changed (a priori "improved"), you do not have to achieve the classical (and boring) cycle: "stop it, change source, compile and restart". The new skill can be dynamically teached to the agent (who must forget the older one). This could be particularly important for a "long life" agent who is dedicated to some role that does not withstand any interruption.

size optimisation If you consider an agent with low memory (like on organisers or cellular phones), you can chose to load your agent with only the skills required at a given time (and unload others).

4.2.4 Role evolution

With this dynamic evolution, the important point is that it is in fact the roles the agent can play inside the MAS that change. So the MAS in its whole evolves and can adapt according to the current flow between agents.

With this possibility of dynamic evolution of agent, you can no more use the term of "class" of agents. Even if you start from a common basis for different agents, as they can (and probably will) receive a different education due to their "experience", they will soon differ and it will eventually be impossible to consider them as belonging to a same "class". This notion has definitively no more meaning in this context. This constitutes a strong difference between such an agent oriented programming and the object-oriented programming.

4.2.5 Conclusion

We have presented our vision of agents built from an atomic agent through a dynamic skill learning. We will now present how this principle can effectively be applied to describe a MAS model. This will be illustrated using our own model, called MAGIQUE.

4.3 MAGIQUE

We will describe briefly our proposition of organisation for multi-agent systems. It is called MAGIQUE and is based on the notion of hierarchies of agents (this could be seen as well as an agent recursively made of agents, see figure 4.1 (Mezrura, Occello, Demazeau, and Baeijs 1999)). More details could be found in (Mathieu and Routier 2000-2001, Mathieu and Routier 2001)[4].

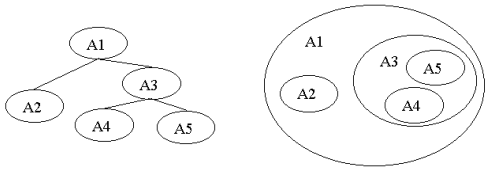

Figure 4.1: Two visions of the same MAS (or agent). Left: hierarchic vision. Right: "agent made of agents" vision.

The hierarchical organisation of agents allows a default automatic skill delegation mechanism that facilitates development of agents and of MAS. The used agents are the one described in section sec:atomic.

4.3.1 Hierarchies

A hierarchy is a tree whose root is labelled by an agent and whose childs, when any, are hierarchies too:

$$\text{MAS} = \text{Hierarchy}^*$$

$$\text{Hierarchy} = \begin{cases} \text{Agent} \\ \text{Agent} \\ \text{Hierarchy} \quad \cdots \quad \text{Hierarchy} \end{cases}$$

Leaf agents are called "*specialists*" and others "*supervisors*", these must be able (i.e. have the skill) to manage the "*team*" of agents (the sub-hierarchy) they are the root of).

A hierarchy characterises the basic structure for the routing of messages between agents. A hierarchical link denotes a communication channel between the implied

[4] and at `http://www.lifl.fr/MAGIQUE`

agents, and when two agents of a same structure are exchanging a message, by default this passes along the tree structure. This corresponds to some *vertical* communication.

With only those communications, the model would probably be too strict. That's why, Magique offers the possibility to create direct links (i.e. outside the hierarchy structure) between agents. We call them "*acquaintance links*".

The decision to create such links depends on the agent policy. However the intended goal is the following: after some times, if some request for a skill occurs frequently between two agents, the decision to dynamically create an acquaintance link for that skill can be taken. The interest is of course to promote the "natural" interactions between agents at the expense of the hierarchical ones. These links constitute the *horizontal* communications.

Then, in Magique, there is a default communication organisation that is the hierarchy. But this structure is doomed to evolve according to the dynamicity of the MAS in order to promote the most often used relations. Then after some times, the MAS should look more like a graph.

4.3.2 Mechanism for skill delegation

When an agent has a task to achieve, this requires the exploitation of some skills that the agent can directly know or not. In both cases the way he invokes the skills is the same. If the realisation of a skill must me delegate to another, this is done transparently for him. This delegation is performed thanks to the hierarchical structure. Here follows the principle of skill invocation:

- if the agent knows the skill, he uses it directly,
- if he does not, several cases can happen :
  - he has a particular acquaintance for this skill, he asks him to achieve the skill for him,
  - else, he is a supervisor and someone in his hierarchy knows the skill, then he forwards (recursively through the hierarchy) the realisation to the skilled agent,
  - else, he asks to its supervisor to find for him some gifted agent and this supervisor applied the same mechanism to do it.

One first advantage of this mechanism of skill achievement delegation is to increase the reliability of the MAS: the particular agent who will perform the skill has no importance for the "caller", therefore he can change between two invocations of the same skill (because the first had disappeared of the MAS or is overloaded, or ...) (cf. Figure 4.2).

Another advantage appears at the programming stage. Since the search of a skilled agent is automatically achieved by the hierarchy, when a request for a skill is programmed, there is no need to specify a particular agent. Consequently the same agent can be used in different contexts (i.e. different multi-agent applications) so long as an able agent (no matter which particular one) is present. A consequence is, that when designing a MAS, the important point is not the agents themselves but their skills.

4.3.3 MAGIQUE and skills

The agent used in MAGIQUE corresponds to the one presented in the section 4.2. They are built from an atomic agent by dynamic skill learning. Then the basic MAGIQUE agent must

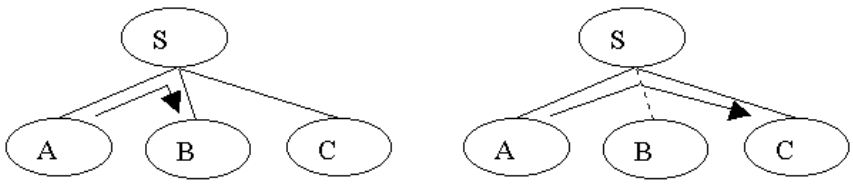

Figure 4.2: Agent A needs some skill , he invokes it and the request is forwarded via the supervisor S: **a.** S sends it to B – **b.** the link with B has been removed, delegation is automatically forwarded to C without A being aware of it.

have the few additional skills that are particular to MAGIQUE. Skills for the hierarchy management, others for the dynamicity (connecting, creating and killing agents), are the main ones (see figure 4.3).

All the other skills are applications skills, it is the MAS designer task to create them and to teach them to the agents. But the MAS can be built in such a way that the learning, and the agent evolution and specialisation, are automatic. The MAGIQUE API offers the tools to do that.

4.3.4 Dynamicity in MAS

In MAGIQUE an important point is that MAS can evolve dynamically. This dynamicity acts at different levels.

individual an agent can acquire or forget skills. The advantages of this aspect have been discussed previously. In Magique, this corresponds to an effective exchange of skills between agents that can be remote, and the agents have the initiative of the exchange.

communication acquaintance links can be created when favourite relations between two agents occur. This creation allows to cut some repeated communications along the tree structure as communication become direct between the implied agents. The decision of creating such a link is a prerogative of the agents.

With the skill delegation principle, this contributes to a non deterministic behaviour of the MAS: two successive “execution” of the same multi-agent applications can lead to two different communication structure and consequently the skills will not necessarily be achieved by the same agents.

organisational agents can be created or removed dynamically to adapt the MAS to some constraints. Two examples among many:

- an agent is overloaded by requests about one of his skills, he can decide to create a team of agents with that particular skill and then he can delegate some requests to these agents.
- an agent can leave temporarily the MAS and recover his place later, messages for him must be stored. This can be essential for agent located on mobile computer (or phone or ...).

4.4 API

MAGIQUE has been put into concrete form through an API[5]. It consists in an application framework above JAVA to develop agents and multi-agents systems over heterogeneous distributed network. Agents are indeed easily built from skills through teaching of these skills from an atomic agent.

In this API, written in JAVA, primitives are provided that allow an **effective** exchange of skills between running agents. The teaching of a skill between two agents can be done with no a priori condition concerning the code: when an agent teaches a skill to a *remote* other agent, the bytecode corresponding to the skill is **really** passed from the teacher to the learner. So even, if the "class" was not initially known by the platform of the learner, everything will work fine.

Let us note, that since the API is a framework, the application part is left to the user. Thus the user must chose and create the policies used to decide when a skill must be acquire or when a new acquaintance must be created, or etc.

4.4.1 Agent creation

Since it exploits the skill learning, the API offers an easy framework to develop agents. Indeed, building an agent is reduced to something like a script consisting in skill "plugging" starting from atomic agent. Then in Magique the source code for creating an agent looks like:

```
import fr.lifl.magique.*;
...
 Platform p = new Platform();
 // hollow agent creation
 Agent myAgent = p.createAgent("myName");
 // agent acquires skills (= component)
 myAgent.addSkill(new SkillOne());
 myAgent.addSkill(new SkillTwo(...));
 // join a hierarchy (= MAS)
 myAgent.connectToBoss("bossName...");
...
```

As you can note in the source code, Magique uses the notion of platform and each agent must belong to such a platform. This is used to facilitate message routing between agents, but the platform offers the support for "physically" exchanging skills too. In Magique, two agents can *effectively* exchange skills with no need to make hypothesis about the location of classes: dynamic skill code loading and exchange is automatically performed (even remotely and, in this case, even if network is broken).

As an example, the agents used in MAGIQUE are built from the atomic agent through the skill acquisition/education described in figure 4.3.

[5] It can be downloaded at `http://www.lifl.fr/MAGIQUE`

```
public class Agent extends AtomicAgent {
...
   protected void initBasicSkills() throws SkillAlreadyAcquiredException {
      addSkill(new fr.lifl.magique.skill.system.DisplaySkill());
      addSkill(new fr.lifl.magique.skill.system.AddSkillSkill(this));
      addSkill(new fr.lifl.magique.skill.system.LearnSkill(this));
      addSkill(new fr.lifl.magique.skill.system.ConnectionSkill(this));
      addSkill(new fr.lifl.magique.skill.magique.BossTeamSkill(this));
      addSkill(new fr.lifl.magique.skill.magique.ConnectionToBossSkill(this));
      addSkill(new fr.lifl.magique.skill.magique.KillSkill(this));
   }
}
```

Figure 4.3: Education of MAGIQUE agent from an tomic agent (directly extract from MAGIQUE API source code)

4.4.2 Skill creation

In the API, building a skill is writing a class whose instances are the software components teached to the agents. The `public` methods of this component can then directly be used by the agent.

```
import fr.lifl.magique.*;
import fr.lifl.magique.skill.*;
...
public class ASkill implements Skill {
 public ASkill() {...}

 //agent will be able to use <ability>
 public void ability(...) {
 }
}
```

4.4.3 Skill invokation and delegation

The invokation of a skill is very simple to program too. Let us recall that in Magique, it is not necessary to explicitly know an able agent, the MAS insures that if one exists, the realisation of the skill will be automatically forwarded.

Where in OOP you should write:

```
object.ability(arg...);
```

to make a call to a given method `ability`, you must write:

```
perform("ability",arg...);
```

This has the effect that a skill named "`ability`" will be "called" (no need to know by who[6]).

[6]Of course, Magique offers also possible to precise a recipient to an invokation request if needed.

The *perform* primitive is dedicated to the invokation of skills with no required answer. There exist mainly three other primitives: `ask` when an asynchronous answer is required, `askNow` for an immediate answer and `concurrentAsk` for a concurrent invokation of a skill.

4.4.4 Dynamic skill acquisition

The basic MAGIQUE agent knows the skill required to acquire dynamically new skills.

Thus, if you want an agent to acquire dynamically a new skill, whose name is `"skill"`, it suffices to use `addSkill` as stated before :

```
addSkill("skill",args);
```

Now, if you want this same skill being teached by an agent names `"teacher@..."`, you need the `learnSkill` skill :

```
perform("learnSkill",
        new Object[]{"skill","teacher@...",
                     args});
```

Once this done, the agent will be able to use the new skill. This can be done even if the teacher and the learner agent are on two different remote machines. No hypothesis need to be made about bytecode, , location. If needed, the bytecode for the skill will be forwarded to the platform of the learner.

4.4.5 Graphical Environment

In order to facilitate the creation of hierarchies, to distribute agents over the net and to allow addition of skills to agents, we have built a graphical development tool (see figure 4.4). Once the skill classes have been written, this tool allows to automatically generate agents, to build a MAS with them and to distribute the MAS over a network.

Once the agents have been distributed, this environment provides an interface to track the behaviour of each agent.

A shell tool allows to interact with the agents during their “life”, in order to learn them a new skill for example.

More details and some examples are available on the web site :

`http://www.lifl.fr/SMAC`

4.4.6 Conclusion

As you can see from this quick overview, once you know the JAVA language, there is no difficulty to apprehend and use the Magique API: the method calls are replaced by skill invocations according to a slight syntactical change explained above.

Of course the methodological difficulties due to the multi-agent aspect and the skill modularity are another problem...

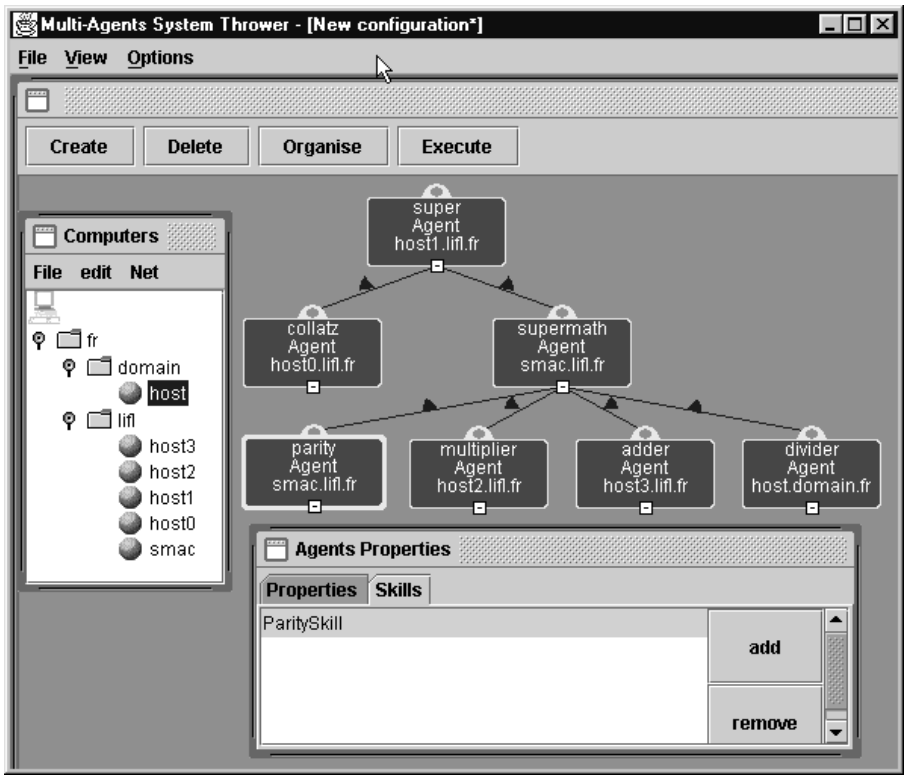

Figure 4.4: Graphical environment for building and distributing agents.

4.5 An application to dynamic skill learning

An application of the skill exchange can be a dynamic evolution of the role of the agents: you can make evolve or reduce the abilities of an agent and then change its role in the application.

We have developed a small groupware multi-agent application consisting in a distributed conference (see (Mathieu and Routier 2001)[7]). In this application, mainly two roles can be identified for the agents: the speaker and the listeners. The speaker is identified since he is the only one who own a remote control at a time. Of course, he can give this remote control to another agent and then he becomes a plain listener, and the receiver becomes the speaker. Therefore the roles of the agents can evolve dynamically.

The remote control is of course a skill since it gives special ability to its owner. Then when a speaker gives the control to a listener, what he does in fact is to teach the "control skill" to the listener and also forget it (see 4.5). And this is **effectively** done, the former speaker agent has really lost the ability to use the control. And this is done at the low bytecode level too: even between two remote agents, the skill bytecode is really exchanged and learned or lost.

In this case, we see that dynamic skill learning is used to manage something like the rights over an application. Roles evolve dynamically according to the skills learned or forgotten by the agents. The MAGIQUE API provides an easy to use framework to design applications with such behaviour.

[7] or/and have a look at `http://www.lifl.fr/MAGIQUE/examples/diapoExt.html`

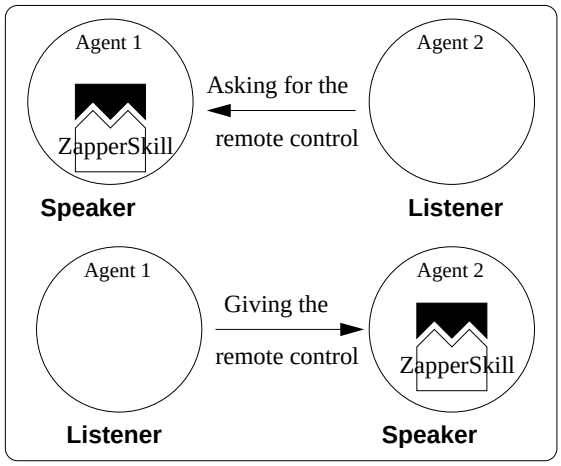

Figure 4.5: Dynamic evolution of roles through skill exchange

4.6 Conclusion

Here, we have proposed a vision of agents built from an atomic frame using dynamic skill acquisition. Skills are coherent sets of abilities and can be seen as software components. From a programmer point of view, an advantage is that modularity and reusability are promoted. Then an agent evolves during his "life". That means that he can play different roles depending on the skills he knows at that time.

This is an efficient tool to develop agent applications. The next step is to provide a methodology strongly based on the skill notion whose formalisation must be deepened. Starting from studies like (Wooldridge, Jennings, and Kinny 1999), a fundamental consideration of the nature of interaction must be undertaken. Interactions can indeed be tackled in terms of skills and roles. We are working on a formalism for representing them and we think it will lead to an automatic generation of MAS and agents. Dynamic learning should help that.

An API has been developed to validate these ideas. It allows to effectively build agents by dynamic "skill plugging" and to distribute them, no hypothesis about where the source of a skill is need to be done as soon as the learner knows it. The API, the graphical environment, some small illustrating examples and a brief tutorial can be downloaded at:

`http://www.lifl.fr/SMAC`

Bibliographie

Bensaid, N.; and P. Mathieu. 1995. "Un modèle d'architecture multi-agents entièrement écrit en Prolog," in *IV Journées Francophones de Programmation Logique, JFPL'95*, 381–385, Dijon-France. teknea, Toulouse-France.

Bensaid, N.; and P. Mathieu. 1997. "A Hybrid and Hierarchical Multi-Agent Architecture Model," in *Proceedings of PAAM'97*, 145–155.

Ferber, J.; and O. Gutknecht. 1998. “A meta-model for the analysis and design of organizations in multi-agent systems,” in *Proceedings of ICMAS’98*.

Franklin, S.; and A. Grasser. 1996. “Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agent,” in *Proceedings of the 3rd International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag.

Horling, B.; and V. Lesser. 1998. “A Reusable Component Architecture for Agent Construction,” Discussion paper, UMass Computer Science.

Mathieu, P.; and J.-C. Routier. 2000-2001. “Tutoriel de Magique,” Equipe SMAC - LIFL.

Mathieu, P.; and J.-C. Routier. 2001. “Une contribution du multi-agent aux applications de travail coopératif,” *TSI Hermès Science Publication. Réseaux et Systèmes Répartis. Calculateurs Parallèles.*, 13. Numéro spécial télé-applications, 207–226.

Mezrura, C.; M. Occello; Y. Demazeau; and C. Baeijs. 1999. “Récursivité dans les systèmes multi-agents : vers un modèle opérationnel,” in *JFIADSMA’99*, 41–52. Hermès.

Shoham, Y.. 1993. “Agent-Oriented Programming,” *Artificial Intelligence*, 60, 51–92.

Travers, M.. 1996. “Programming with Agents: New metaphors for thinking about computation,” Ph.D. thesis, MIT.

Wooldridge, M.; N. R. Jennings; and D. Kinny. 1999. “A Methodology for Agent-Oriented Analysis and Design,” .

Chapitre 5

Principles for Dynamic Multi-Agent Organizations

Philippe Mathieu, Jean-Christophe Routier et Yann Secq
Laboratoire d'Informatique Fondamentale de Lille
Cité Scientifique 59655 Villeneuve d'Ascq Cedex
{mathieu,routier,secq}@lifl.fr

Abstract.[1]

Many models of organizations for multi-agent systems have been proposed so far. However the complexity implied by the design of social organizations in a given multi-agent system is often not mentioned. Too little has been said about rules that must be applied to build the architecture of acquaintances between agents. Moreover, tools for managing the dynamic evolution of organizations are seldom provided in current framework propositions.

In this paper we discuss self-adaptation of organizations in multi-agent systems according to the dynamic of interactions between agents. Starting from a default organization, the architecture of acquaintances evolves autonomously depending on messages flow in order to improve the global behaviour of the system. We propose three principles that can be applied to adapt the organization: "have a good address book", "share knowledge", "recruit new able collaborators".

These principles have been applied in our multi-agent platform called MAGIQUE.

5.1 Introduction

Multi-agent systems can be seen as societies of interacting agents. This notion of interaction, which allows agent to find each other and then to exchange information, is a central point for the design of multi-agent applications. Some methodologies have been proposed, and they always identify the need that agents have to *get in touch* with other agents, but they seldom

[1] Article publié dans *Proceedings of PRIMA 2002, Tokyo, LNAI 2413. ISBN 3-540-44056-7. pp 109-122. August 2002.*

provide guidelines to design the acquaintances structure. The GAIA(Wooldridge, Jennings, and Kinny 2000) methodology, for instance, identifies this stage as the *acquaintance model*, which is defined as :

> *An agent acquaintance model is simply a graph, with nodes in the graph corresponding to agent types and arcs in the graph corresponding to communication pathways. Agent acquaintance models are directed graphs, and so an arc $a \rightarrow b$ indicates that a will send messages to b, but not necessarily that b will send messages to a . An acquaintance model may be derived in a straightforward way from the roles, protocols, and agent models.*

We see that this definition just defines what we could call the *natural* notion of acquaintance. The notion of organization is even not clearly identified. In another work (Kinny, Georgeff, and Rao 1996), it is stated that :

> *an Interaction Model describes the responsibilities of an agent class, the services it provides, associated interactions, and control relationship between agent classes.*

Again, this is just a way to express the fact that agents interact and so need to have some communication paths to exchange information. Other methodologies (Brazier, Dunin-Keplicz, Jennings, and Treur 1997, Kendall, Malkoun, and Jiang 1995), often state the same kind of concepts but seldom identify that the acquaintance structure is a first-class citizen of MAS.

It is true that some works highlight the importance of the notion of organization in multi-agent systems : the Contract-Net Protocol (Smith 1979) is based on a market-type system, the Aalaadin (Ferber and Gutknecht 1999) model relies on the idea that agents are identified by the roles they hold within some groups, the Magique (Routier, Mathieu, and Secq 2001) model proposes a hierarchical structure and, lastly, the holonic approach (Gerber, Siekmann, and Vierke 1999). Unfortunately, these works seldom reify this notion.

Moreover building an organization to optimize agent interactions is not straightforward : how should we spread funtionalities among agents, how is it possible to reduce the cost of communication, and overall how can the system deal with agents that freely leave or join it? Lastly, how can organizations deal with the ever-changing flow of agent interactions?

This paper postulates that this complexity should not be exclusively addressed by the multi-agent system designer. The infrastrcuture of organizations infrastructures should provide default behaviours to dynamically optimize communication flow, in order to lower the number of messages that are exchanged, or to improve the quality of service. Too little works have been done in this direction (Decker, Sycara, and Williamson 1997).

Thus, we propose three rather simple and natural principles inspired from social organizations, that can be applied to adapt multi-agent organizations. These principles lead to a modification of the acquaintance organization and to a dynamic modification of the skills of the agents or even to the creation of new agents. This implies a modification of the distribution of the roles and of the dependences network between agents. Firstly the acquaintance structure tends to map to the dependence structure with the creation of new direct communication channels, secondly the distribution of the skills among the agents in the system is dynamically changed. Moreover we want this ability to self adapt to be a primitive feature of the system and not to be chargeable to the multi-agent system designer or dependent upon the agent model.

In first section, we describe the needs to have an adaptive organization. We first present static organizations and their limitations, then we study how social organizations deal with these problems before we apply their solutions to multi-agent systems. The second section illustrates the dynamic organizations through some simple experiments performed with the MAGIQUE framework which will briefly be introduced.

5.2 Adapting the Architecture of the Organization

Before we consider how to adapt the organization of a multi-agent system, some problems with predetermined static structures must be considered. We will then propose some general strategies to tackle these problems.

5.2.1 Some problems with static organizations.

One of the first problems, and probably the basic one, is to determine how acquaintances are created? That is, how an agent can have information about the existence of another able agent. One solution of course, is that this can be predetermined and established by the multi-agent system designer, but this is not an enough satisfactory answer. Firstly, how should this designer proceed to choose the most fitted acquaintance architecture, which methodology must be applied, if there exists any really convenient? And secondly, what about systems where new agents appear, or what happens when the "able acquaintance" is removed from system, or becomes unavailable, because of a network failure for example?

A second problem is more connected with the distribution of the skills over the agents and is related with performance issues similar to load balancing. How can the sys tem be organizedin such a way that no agent becomes a critical overloaded resource(Gerber 1998)? This implies that even if an organizational structure has been chosen, this is not enough. You need to choose how the skills are distributed among the agents. It is, of course, difficult if not impossible, to give universal rules to do this. But when it is possible to avoid it, it would be better if one agent does not become a bottleneck in the system because he is the only one able to provide a too often required service. In this situation you probably prefer the service to be provided by several agents. Of course, this is not always appropriate, in the case of some certification service for example. But when it is, how could it be predetermined? It is not necessarily obvious which service will be critical (in term of overloading) and, even if you give such a service to several agents, how can we ensure that one of the service provider agents will not be overused and others ignored.

Lastly, we will consider a situation where we consider the "client of service" point of view rather than the service provider one. One agent may have to often use some given service for which he must make requests to an able agent. In this case, even if the service provider agent is not overburdened, the client agent will probably be penalized by too many requests, at least because of the communications. It would have been better, when designing the system, to qualify this agent with the service, or to allow the agent to dynamically acquire it.

Aware of these problems, a multi-agent system designer will take them into account and try to anticipate them and he will attend to limit them. He could succeed in that, but what happens in the context of dynamic multi-agent systems, where agents can freely join or leave the system ? This implies that some services will become available at some time and unavailable at others.

Agents must adapt themselves to such a dynamic environment. The designer can not predetermine these situations. Therefore the only thing he can do is to prepare his agents in such a way that they can adapt autonomously to the changes that occur within their environment. In consequence, general strategies must be given? We will discuss some of them in the following.

5.2.2 How does social organizations manage these problems?

The problems we have raised in the previous section are not peculiar to multi-agent systems but are general to social organizations, where members can be persons or companies.

In every social structure, the problem of finding the "right person for the job" appears. Often this "right person" is not known *a priori* and it is necessary to use known acquaintances to find who he/she/it is. But, of course, this may be a source of problems. You do not necessarily want to use some middleman that can know what you want from the "right person" and then make use of this information for his personal advantage. Moreover, this can have a cost since the middleman can ask for a payment only for having helped you to get in touch with the right person. Therefore after a time, when you have obtained the information you needed, you can try to reach the right person directly. This implies that you communicate directly with the person you effectively depend on.

The problem of overloaded resources exists too. The more able a person or a society is, the more it is probable that she/he/it will be overburdened (in fact this is often considered as a symptom of competence). And then the delay before you benefit from its service increases. In this case, the resource too often consulted must find a way to speed up its answer. Otherwise, in the case of a company for example, clients will be seeking an equivalent service somewhere else.

If you consider the client's point of view, making too frequent requests to some critical resource is a major drawback which has a cost. Either a time cost because clients must wait for the availability of the resource, or a money cost because clients pay for the service. Therefore, when it is possible, clients try to circumvent this dependence.

In these three cases, the problem of cost or efficiency appears. In social organizations, there is a trend to aim at better efficiency. This trend can be natural – we all have tendency to apply the law of least effort –, or economical by trying to reduce cost – unless the intent is to increase profit? –.

We have identified three principles that can be used to improve the global behaviour and that implies a dynamical organization of the social structure :

1. having a good address book,
2. sharing knowledge (or selling it...),
3. recruiting new able collaborators.

The first principle deals with the first problem mentioned earlier. It may seem that this principle could have been called "remove the middleman", however this must be moderated. Indeed, creating new (social) links has a cost and it is not always appropriate to circumvent the middleman. He may know his job, and his offer for a given service can change because he has had found of a more able provider. In such a case the use of the middleman would have been beneficial. In consequence, "having a good address book" does not always mean removing the middleman, but rather knowing when to use him and when not.

The second and third principles are rather means to tackle second and third problems and more generally to improve efficiency by reducing the time necessary for a service request to be treated. When a service company is overused, in order not to lose client, it will probably recruit able collaborators. In the same way, when the company needs a new skill, it can recruit new collaborators with the required competence. Or, consider a craftsman with too many orders; he will take one or more apprentices and train them. This is a combination of the two principles, even if it is more of the "sharing knowledge" since the intent is that, after its training, the apprentice becomes a new resource. Of course, again, recruiting or teaching/learning knowledge has a cost and can not be applied every time.

5.2.3 The three principles applied to multi-agent systems

These three principles can be applied to achieve a self organization of the social structure in multi-agent systems. By applying them, we want an evolution of the acquaintance structure and the distribution of skills in order to reduce, firstly, the number of messages exchanged in the system and, secondly, the time necessary for a service request to be treated.

According to these principles, we start from a predetermined organization, where the agents have default acquaintances and where skills (or services) are more or less arbitrarily distributed among the agents. The idea is to have an evolution of the structure of acquaintances where the dependence links are favoured at the expense of predefined ones.

Of course the major benefit should be for the designer of the multi-agent system who can prepare his system as he sees most suitable and then rely on these principles to adapt the efficiency of his system. Here are some examples, where these principles can be of considerable benefit:

- Applying the first principle, the dependence network tends to coincide with the acquaintance network.
- By learning new skills, and agent increases its autonomy.
- If an agent makes requests for a given service, the agent who answers may not be the same one between two requests[2]. This contributes towards increasing the reliability of the multi-agent system. Indeed, even if a skilled agent is removed, another could be found even if the designer had not explicitly anticipated it, or better, without need for the designer to anticipate it.

  We can imagine for example that the acquaintance architecture adapts to match the network performance architecture. Two agents a_1 and a_2 can provide the same service required by a client agent a_c. Depending on the localization of a_1 or a_2 in the network or, between any predefined acquaintances for a_c and one of the a_i, a_c will request only to the provider whose answer is the fastest (of course without using a systematic general broadcast).
- If an agent performs the same task, he can "prefer" to learn a skill and thus remove the need to delegate in order to perform the task.

  On the other side, if an agent is overwhelmed by requests from other agents who want to exploit one of his skills, he can choose to teach this skill to some other agent(s) to multiply the offer and then lighten his burden.

[2]Since the acquaintances are dynamically computed (according to some predefined rules of course).

- If for some reason an agent has to disappear from the multi-agent system and he owns some critical skill, he can teach it to some other agent and thus guarantee the continuity of the whole multi-agent system.

  This has some similarities with the work in (Kumar, Cohen, and Levesque 1999) where the *Adapative Agent Architecture* is proposed: the author use dynamic reorganization with middle-agent (or broker) to improve robustness and promote fault-tolerance. However our goal concerning self-organization is more general since we provide dynamic self organization in order to improve the interactions between agents according to their natural effective dependences (even if this sometimes means to remove this dependence when skills are exchanged, as we will see later).
- When the designer wants to improve how a service is treated in his system, he can dynamically add a new agent with the new version of the skill and make him teach it the other older-version-skilled agents to upgrade them.

To be able to practically apply these strategies, some abilites must be provided by the framework, agents should be able to :

- dynamically create new acquaintance links in order to self adapt the organization ((Ghanea-Hercock 2000)) to match the dependence links. However they must first have a way to find the "right agent". Therefore a default message routing and default acquaintances must be provided for at least reaching the "right agent" through middle-agents.
- learn new skills from other agents (and therefore agents must be able to teach each other) (see (Clement 2000)). A mechanism must be provided that supports it and the distribution aspect must be taken into account.
- create new agents, and by using the learning/teaching ability, these agents could be tuned to what is needed.

Of course, agents will use these abilities autonomously and therefore behavioural strategies, for deciding when to apply them, must be created. There is the need to challenge some of the decisions from time to time. For example when a direct acquaintance link has been created, because at some time it was the most suitable, this may no longer be the case later and then a new adaptation is necessary. Thus, these strategies should integrate some mechanisms to call into question direct acquaintances that have been created.

5.3 Experiments

To experiment these principles, we need a framework that provides code mobility in order to apply the dynamic acquisition of skills. Thus, we used our multi-agent framework called MAGIQUE[3] (Bensaid and Mathieu 1997, Routier, Mathieu, and Secq 2001). We will briefly introduce this framework and then experiment dynamic organizations of multi-agent systems with it.

[3]Magique stands for the french "Multi-AGent hiérarchIQUE" which obviously means "hierarchical multi-agent".

5.3.1 MAGIQUE

MAGIQUE proposes both an organizational model (Bensaid and Mathieu 1997), based on a default hierarchical organization, and a mimimal agent model (Routier, Mathieu, and Secq 2001), which is based on an incremental building of agents. MAGIQUE is dedicated to the *implementation* of multi-agent systems. MAGIQUE is not a multi-agent applications but a support for such applications. Thus, it does not directly provide high level features like knowledge base management, planners, etc.

Dynamicity is a keypoint in MAGIQUE and the three principles of self-organization we have presented need this dynamicity in order to be implemented. The requirements made in section 5.2.3 are satisfied. We will insist on features that promote these aspects, other details will not be taken into consideration here.

The agent model: building agents by making them skilled.

The agent model is based on an incremental building of agents from an elementary (or atomic) agent through dynamical skill acquisition. A skill is a "coherent set of abilities". We use this term rather than service[4], but you can consider both as synonyms here. From a programmer oriented view, a skill can be seen as a software component that groups a coherent set of functionalities. The skills can then be built independently from any agent and re-used in different contexts.

We assert that only two prerequisite skills are necessary and sufficient to the *atomic agent* to evolve and reach any desired agent: one to interact and another to acquire new skills (details can be found in (Routier, Mathieu, and Secq 2001)).

These skills are indeed *necessary*. Without the "skill acquisition" skill, such an agent is just an empty shell unable to perform any task. Without the interaction skill, an agent is isolated from the "rest of the world" and therefore loses any interest. Moreover without communication an agent will not be able to learn new skills from others.

They are *sufficient* since it suffices to an agent to use his interactive skill to get in touch with a gifted agent and then to use his acquirement skill to learn some new talent. Then every ability can be given to an agent through learning from a "teacher". Let us precise that the exchanged skills are "stateless", it is the functional ability that is exchanged not some kind of experience.

Thus we can consider that all agents are at birth (or creation) similar (from a skill point of view): an empty shell with only the two above-mentioned skills.

We claim that every model of agent proposed by the various existing definitions (Franklin and Grasser 1996) matches this definition. Indeed, it "suffices" to build the skills that provide the basic abilities of the model and to teach them to an atomic agent. Thus, we do not want to use a particular agent model, the "high level intelligent" abilities can be chosen and evolve at will.

Therefore, differences between agents issue from their "education", i.e. the skills they have acquired during their "existence". These skills can either have been given during agent creation by the programmer, or have been dynamically learned through interactions with other agents (now if we consider the programmer as an agent, the first case is included in the second one). This approach does not introduce any limitation to the abilities of an agent. Teaching skills to an agent is giving him the ability to play a particular role within the multi-agent system he belongs to.

[4] We keep *service* for "the result of the exploitation of a skill".

For our purpose here, this ability to dynamically learn and teach skills is useful for the dynamic organization of the multi-agent system, in particular to make use of the second and third principles.

The organizational model.

Throughout the following the agents are like these described in the previous section and are supposed to be co-operative agent and not self-interested one.

In MAGIQUE there exists a basic default organizational structure which is a hierarchy. It offers the opportunity to have a default automatic mechanism to find a skill provider.

The hierarchy characterizes the basic structure of acquaintances in the multi-agent system and provides a default support for the routing of messages between agents. A hierarchical link denotes a communication channel between the implied agents. When two agents within the same structure are exchanging a message, by default it goes through the tree structure.

With only hierarchical communications, the organization would be too rigid and thus MAGIQUE offers the possibility to create direct links (i.e. outside the hierarchy structure) between agents. We call them *dependence links* (by opposition of the default *hierarchical links*). The decision to create such links depends on some agent policy. However the intended goal is the following: after some times, if some request for a skill occurs frequently between two agents, the agent can take the decision to dynamically create a dependence link for that skill. In fact, to be more precise, we must say that an acquaintance link corresponding to a dependence link is created. The aim is of course to promote the "natural" interactions between agents at the expense of the default hierarchical ones.

With the default acquaintance structure, an automatic mechanism for the delegation of requests between agents is provided without the use of a general costly broadcast. When an agent wants to exploit some skill it does not matter if he knows it or not. In both cases the way he invokes skills is the same. If the realization of a skill must be delegated to another, this is done automatically for him, even if he does not have a particular acquaintance for it. The principle of the skill provider search is the following:

- the agent knows the skill, he uses it directly
- if he does not, several cases can occur
  - if he has a particular acquaintance for this skill, this acquaintance is used to achieve the skill (ie. to provide service) for him,
  - else, he has a team and someone in his sub-hierarchy knows the skill, then he forwards (recursively through the sub-hierarchy) the realisation to the skilled agent,
  - else, he asks his supervisor to find for him some competent agent and his supervisor applies the same delegation scheme.

In this mechanism, some agents play the role of middle-agents as defined in (Decker, Sycara, and Williamson 1997): "*Agents (...) that are* neither *requesters* nor *providers*". But let us precise, that this is just a temporary state: these agents are not dedicated to be exclusively middle-agents and can send requests or provide services at other moments.

One first advantage of this mechanism of skill achievement delegation is to increase the reliability of the multi-agent system: the particular agent who will perform the skill has no

importance for the "caller", therefore he can change between two invocations of the same skill (because the first has disappeared from the multi-agent system or is overloaded, or ...).

Another advantage appears at the programming stage. Since the search of a skilled agent is automatically achieved by the organization, when a request for a skill is coded, there is no need to specify a particular agent. Consequently the same agent can be used in different contexts (i.e. different multi-agent applications) so long as an able agent (no matter which particular one) is present. A consequence is, that when designing a multi-agent system, the important point is not necessarily the agents themselves but their skills (ie. their roles).

Obviously the evolutive default organizational structure with its automatic skill provider search offers the tools to apply the above-mentioned principles.

The API

These models have been put into concrete form as a JAVA API. It allows the development of multi-agent systems distributed over heterogeneous network. Agents are developed from incremental skill plugging (and dynamically if needed) and multi-agent system are hierarchically organized. As described above, some tools to promote dynamicity in the multi-agent system are provided: direct communication links can be created, new skills can be learned or exchanged between agents (with no prior hypothesis about where the bytecode is located, when needed it is exchanged between agents). This API can be downloaded at `http://www.lifl.fr/SMAC` topic MAGIQUE. We have used it to developped a co-operative work application (Mathieu and Routier 2002) or a framework for distributed calculus (Mathieu, Routier, and Secq 2002c).

5.3.2 Three experiments for three principles.

In this section we will present brief experiments that put into concrete form the principles of dynamic organization that have been described. These experiments have been completed with MAGIQUE[5].

The first consists in creating the acquaintances that suit the best to the natural flow of messages in the multi-agent system. In the second, the distribution of skills in the system is dynamically changed. While in the third new collaborators are created by an agent who wants to get rid of the need to treat too many requests for a given service.

First experiment: adapting the acquaintances organization

This is a simple example where one agent, *SU*, is a service user and the required service can be provided by two other agents of the multi-agent system, *SP1* and *SP2*. At the beginning, the multi-agent system is organized into a hierarchy and our three agents are located somewhere in the hierarchy but are not directly connected (cf. Figure 5.1). We do not show other agents in the multi-agent system since they do not interfere here. We have chosen to have *SP1* and *SP2* connected to the same root agent but this is of no importance or influence. These agents are distributed over a network of workstations.

Agent *SU* sends at regular intervals requests for a service σ. Once the service has been performed a payment request is sent back to *SU*, thus we have a way to measure the time which has

[5]The source codes of these experiments can be downloaded at `http://www.lifl.fr/MAGIQUE/dynamicity` and experiments can then be reproduced.

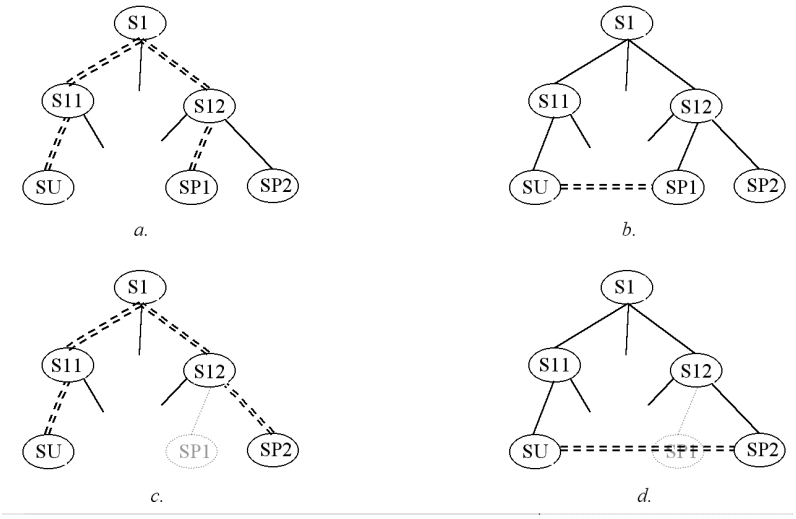

Figure 5.1: *Dynamic organization of acquaintances to match dependence links. a.* Beginning: multi-agent system is hierarchically organized, service requests (see double dash lines) use the default hierarchical organization and *SP1* is reached. *b.* Self-organization: direct communication link that corresponds to a concrete dependence with *SP1* is created. *c. SP1* disappears: service requests use the default organization and *SP2* is reached. *d.* Self-organization: direct communication link with *SP2* is created.

elapsed between the initial service request and the completion of the service (the same criterium is used in (Decker, Sycara, and Williamson 1997)).

At the beginning since *SU* does not know any skilled agent, the requests is routed using the default hierarchical organization. According to the automatic skill provider search, *SP1* is reached (see Figure 5.1-*a.*).

After some requests, since the same *SP1* provides the service to *SU*, *SU* decides to create a direct communication link with *SP1* in order to favour the dependence. The decision is taken according to some criteria that can be customized while the agent is designed (in this case a simple threshold decision process has been used). The direct link is now used (see Figure 5.1-*b.*) and as consequences:

- the number of messages sent in the multi-agent system is reduced,
- the agents *S1*, *S11*, *S12* are less “stressed” and can use their time to perform other tasks than routing messages and being used as middle-agents,
- thirdly the delay before the service is finished is reduced.

Now, assume that agent *SP1* is removed from the multi-agent system. Then the default hierarchical organization is again used, and agent *SP2* is now reached (see Figure 5.1-*c.*). The direct benefit for the multi-agent system is fault tolerance. Although an able agent disappears, the organization provides a way to find another able agent. This is automatically done for the service user, he performs the service requests in the same way as before.

Lastly, after some times the multi-agent system adapts again, and an acquaintance link between *SU* and *SP2* is created (see Figure 5.1-*d.*).

The table at figure 5.2 gives, for the 4 periods, the average time between the moment a service σ request is sent and the moment the payment is achieved. The first line corresponds to a multi-agent system where only agents *SU*, *SP1* and *SP2* are working. In the second line, agents have been added to simulate load on *S*, *S1* and *S2* and to generate extra network traffic. This is a more "realistic" situation. This explains differences between numbers in the first and third columns for the two rows.

Agents *SU*, *SP1* and *SP2* have been distributed over a network, and *SP2* was located in a different domain from the two others, this explains the slight difference between the results in columns two and four.

When the direct communication link is created, the middle-agents *SI*, *SI1* and *SI2* are no more used. We can see the performance enhancement of it in the differences between columns a and b.

Fig 5.1-*a*.	Fig 5.1-*b*.	Fig 5.1-*c*.	Fig 5.1-*d*.
174.25	135.8	144.3	118.2
341.37	147.1	325.1	119.6

Figure 5.2: Average durations in milliseconds before service achievement

Second experiment: adapt the skill distribution

This experiment is similar to the previous one. One agent, *SU*, is a service user and the required service can be provided by another agent *SP*. In the beginning, the multi-agent system is organized into a hierarchy and the two agents are located somewhere in the hierarchy (cf. Figure 5.3).

The scenario is the following: agent *SU* sends at regular time requests for a service σ. Once the service has been performed a payment request is sent back to *SU*.

At the beginning since *SU* does not know any skilled agent, the requests are routed using the default hierarchical organization. According to the automatic skill provider search, *SP* is reached (see Figure 5.3-*a.*).

But after some times, according to some predefined policy of his own, *SU* decides to acquire from *SP* the skill that is required to achieve the service σ. Since *SP* agrees, the skill σ is exchanged between agents (see Figure 5.3-*b.*). No hypothesis has to be made about the location of bytecode for σ, it is physically exchanged between agents[6] if needed.

Of course, once *SU* has learned (or acquired, to avoid confusion with the "learn" term) the skill, he is no longer dependant on *SP* and service σ is satisfied faster (see Figure 5.3-*c.*). Moreover, *SP* is freeded from the need to "help" *SU*. *SU* has increased his automony.

Now, if *SU* is disconnected from the system (see Figure 5.3-*d.*), he can still perform service σ (or similarly if it is *SP* that leaves the system).

[6]More precisely, exchange is performed by the platforms that host the agents, since it is the principle of the implementation of the MAGIQUE API.

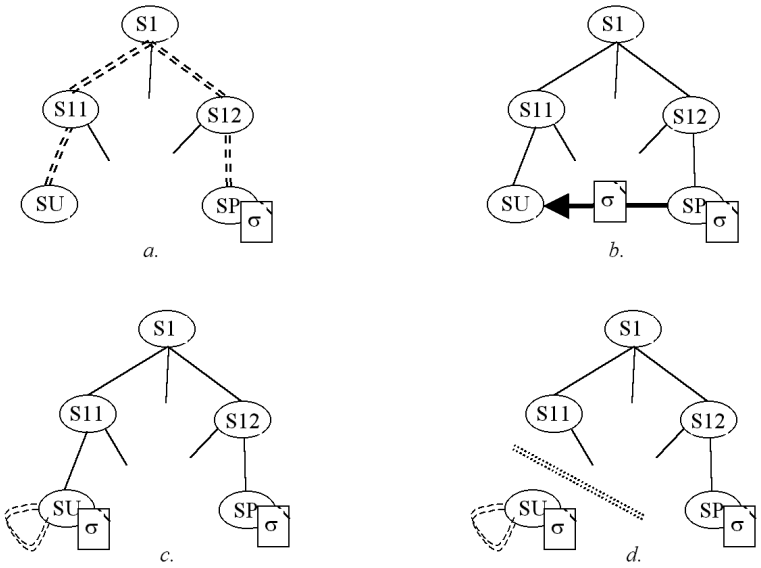

Figure 5.3: *Dynamic acquisition of skill. a.* Beginning: multi-agent system is hierarchically organized, service requests (see double dash lines) use the default hierarchical organization and *SP* is reached. *b.* Exchange: skill σ is "learned" by *SU* from *SP*. *c. SU* uses its "own" σ to achieve what he needs to. *d. SU* can even be disconnect from the remainder of the system.

Giving figures like the previous experiment is nor really meaningful. Before *SU* has acquired/learned the service, the time before service is carried out depends on how much *SP* and the hierarchy are loaded. After the skill acquisition, the time elapsed to perform the service for *SU* is reduced to the time needed to invoke it locally and disconnection of agent *SP* or *SU* is of no consequence on the achievement of σ.

Third experiment: create a pool of apprentices

In this experiment, an agent *SU* makes requests to a service σ. This service can be provided by an agent *SP*. But *SP* is also the agent which provides some π service. We assume this π service to be highly requested by some π-user agents (see Figure 5.4-*a.*).

Therefore, *SP* is overwhelmed with requests to its π-skill and *SU*, who does not use π, suffers from that. To avoid this situation, *SP* creates a pool of agents to support him. He teaches these agents the skill required to perform π and each time he receives a request for π, he dispatches it to one of his apprentices (see Figure 5.4-*b.*). The consequence is of course, that *SP* can spend more time satisfying other requests and in particular requests to σ. Thus, the global efficiency of the system is improved.

In this experiment, 8 π-users are used. They send n requests and simultaneously *SU* makes m requests for σ. Before the pool of apprentices is created (that is, when *SP* is alone to satisfy all requests), the $n.\pi$ et $m.\sigma$ requests are all achieved after 52 seconds. When *SP* creates a pool of 3 agents, for the same $n.\pi$ and $m.\sigma$ requests, we obtain a time of 30.7 seconds.

Of course, all these experimentats are just *proofs of concept*, and in particular figures are given only as examples.

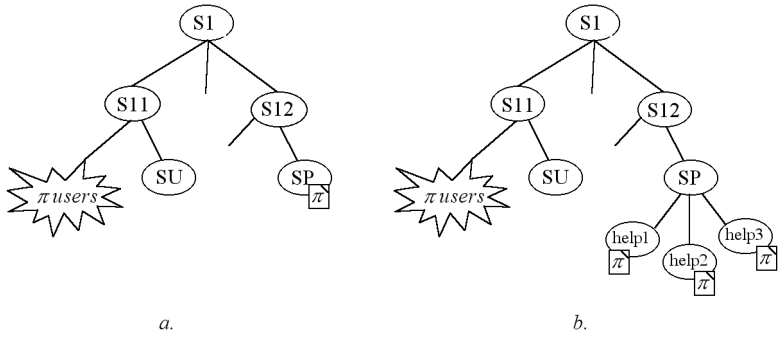

Figure 5.4: *Create pool of apprentices*. *a. SP* must satisfy requests from π service users and from *SU*, he is overwhelmed by requests for π. *b. SP* has created 3 apprentice agents and taught them the π skill, he distributes requests for π to these apprentices and thus lightens his burden.

5.4 Conclusion

Static organizations have drawbacks. In order to be efficient, there is a need to be reactive and to adapt the organization to the reality of the exchanges. Our theme in this paper is that the needs are the same for multi-agent systems. It is too difficult (and probably even impossible) for a multi-agent system designer (and moreover for a team of designer) to foresee the flow of messages within his system. It should be possible to rely upon generic strategies to manage dynamicity of exchanges.

We have proposed some principles to adapt the organization in order to reduce the number of messages in the multi-agent system and to improve the delay before a request is satisfied:

- creation of new specific acquaintance relations to remove the middle-agents,
- exchange of skills between agents to increase autonomy,
- creation of new agents to reduce overloading.

A consequence of the application of these principles is a modification of the dependence network. Agents can apply these principles autonomously depending on some decision of their own. And the taken decision should be challenged after some times, to ensure that the current acquaintance is still the best choice. Our position is that such abilities must be provided as basics in a multi-agent framework.

Future works on this notion of dynamic organizations should be given a more formal framework, particularly by working on and defining an ontology that describes its semantic. Then, we could have agents that belong to several organizations, relying on different kinds of organizational models. But they would be able to handle the dynamicity within these organizations.

Bibliographie

Bensaid, N.; and P. Mathieu. 1997. “A Hybrid and Hierarchical Multi-Agent Architecture Model,” in *Proceedings of PAAM’97*, 145–155.

Brazier, F. M. T.; B. M. Dunin-Keplicz; N. R. Jennings; and J. Treur. 1997. "DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework," *Int Journal of Cooperative Information Systems*, 6(1), 67–94.

Clement, R.. 2000. "To Buy or to Contract Out: Self-Extending Agents in Multi-Agent Systems," in *SOMAS: A Workshop on Self Organisation in Multi Agent Systems*.

Decker, K.; K. Sycara; and M. Williamson. 1997. "Middle-Agents for the Internet," in *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Nagoya, Japan.

Ferber, J.; and O. Gutknecht. 1999. "Operational Semantics of a Role-based Agent Architecture," in *Proceedings of ATAL'99*.

Franklin, S.; and A. Grasser. 1996. "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agent," in *Proceedings of the 3rd International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag.

Gerber, C.. 1998. "Bottleneck Analysis as a Heursitic for Self-Adaptation in Multi-Agent Societies," Discussion paper, DFKI GmbH.

Gerber, C.; J. Siekmann; and G. Vierke. 1999. "Holonic Multi-Agent Systems," Discussion paper, DFKI GmbH.

Ghanea-Hercock, R.. 2000. "Spontaneous Group Formation in Multi-Agent Systems," in *SOMAS: A Workshop on Self Organisation in Multi Agent Systems*.

Kendall, E. A.; M. T. Malkoun; and C. H. Jiang. 1995. "A Methodology for Developing Agent Based Systems," in *First Australian Workshop on Distributed Artificial Intelligence*, ed. by C. Zhang, and D. Lukose, Canberra, Australia.

Kinny, D.; M. Georgeff; and A. Rao. 1996. "A Methodology and Modelling Technique for Systems of BDI Agents," Discussion paper, Australian AI Institute.

Kumar, S.; P. R. Cohen; and H. J. Levesque. 1999. "The Adaptive Agent Architecture: Achieving Fault-Tolerance UsingPersistent Broker Teams," Discussion Paper CSE-99-016-CHCC.

Smith, R. G.. 1979. "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," in *Proceedings of the 1st ICDCS*, 186–192. IEEE Computer Society.

Wooldridge, M.; N. Jennings; and D. Kinny. 2000. "The Gaia Methodology for Agent-Oriented Analysis and Design," *Journal of Autonomous Agents and Multi-Agent Systems*.

Chapitre 6

RIO : Roles, Interactions and Organizations

Philippe Mathieu, Jean-Christophe Routier et Yann Secq
Laboratoire d'Informatique Fondamentale de Lille
Cité Scientifique 59655 Villeneuve d'Ascq Cedex
{mathieu,routier,secq}@lifl.fr

Abstract.[1]

The notions of role and organization have often been emphasized in several agent oriented methodologies. Sadly, the notion of interaction has seldom been reified in these methodologies. We define here a model of runnable specification of interaction protocols. Then, we propose a methodology for the design of open multi-agent systems based on an engineering of interaction protocols. These interaction protocols are described in term of conversation between micro-roles characterized by their skills, then micro-roles are gathered in composite roles. Then, composite roles are used to build abstract agents. Lastly, these latter can be distributed on running agents of a multi-agent system.

6.1 Introduction

The idea of an agent based software engineering has appeared roughly ten years ago, with the paper from Shoham entitled *Agent Oriented Programming*(Shoham 1993). Since these days, several methodologies have been proposed to help developers in their analysis and design (Kendall, Malkoun, and Jiang 1995, Brazier, Dunin-Keplicz, Jennings, and Treur 1997). For that, the concepts of role, interaction and organization are often proposed to facilitate the decomposition and the description of distributed systems. However, we think that suggested methodologies do not clearly identify the various levels of abstraction making it possible to break up a system

[1] Article publié dans *Proceedings of the 3rd International/Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003. Prague. LNAI 2691. pp 147-157. June 2003. ISBN 3-540-40450-3.*

and especially they generally do not propose pragmatic concepts or principles facilitating the realization of such systems. Thus, our proposal relies on a model of minimal generic agent and a model of executable specification of interactions. The agent model is the infrastructure allowing the deployment and the management of interactions, while the specification of the interactions describes a global sight of the conversations between the agents of the system.
In the first part of this article, we briefly present two methodologies which were proposed for the use of multi-agent systems for the design of complex distributed systems, then we put them in relation with interaction oriented approaches. In the second part, we propose a model of minimal generic agent and a formalism for the specification of interaction protocols between micro-roles. The latter are assembled in composite roles which are then atributed to the agents of the system. This specification is made executable by the generation of Colored Petri Nets for each micro-role. This executable specification and the use of a generic model of agent enable us to propose the RIO methodology facilitating the design, the realization and the effective deployment of multi-agent systems.

6.2 Agent methodologies and interaction languages

Several agent oriented methodologies have been proposed like AALAADIN(Ferber and Gutknecht 1999) or GAIA (Wooldridge, Jennings, and Kinny 2000). It is significant to notice that these two methodologies do not make any assumption on agent models and concentrate mainly on the decomposition in term of roles of a complex system. This point is fundamental, in particular because of the multiplicity of available agent and multi-agent systems models. This multiplicity makes the task of the developer difficult: which agent model should be used? Which organizational model should be chosen? Indeed, each platform imposes too often both its own agent model and its organizational model. These methodologies are interesting on many points, but remains too general to ease the transition from the design stage to its concrete realization. Moreover, the various levels of communication are not clarified in the description of interaction protocols. Indeed, works on agent communication languages (ACL)identify three levels that constitutes a conversation: the semantic, the intention (these two are expressed through languages like KIF or SL, and KQML or FIPA-ACL), and the interaction level. However, even by considering heterogeneous platforms sharing the same ontology, it remains difficult to have guarantees on the respect of interaction protocols. This is the reason why works have been undertaken to formalize this aspect with several objectives: to describe the sequence of the messages, to have certain guarantees on the course of a conversation and to ease interoperability between heterogeneous platforms.

Interaction languages. To illustrate these approaches based on a formalization of the interactions, we studied three of them: APRIL(McCabe and Clark 1995), AGENTALK(Kazuhiro Kuwabara 1995) and COOL(Barbuceanu and Fox 1995). APRIL is a symbolic language designed to handle concurrent processes, that eases the creation of interaction protocols. In APRIL, the developer must design a set of *handlers* which treats each message matghing a given pattern. According to the same principles, AGENTALK adds the possibility to create easily new protocols by specialization of existing protocols, by relying on a subclassing mechanism. Another fundamental contribution of AGENTTALK is the explicit description of the protocol: the conversation is represented by a set of states and a set of transition rules. This same principle was employed in COOL, which proposes to model a conversation using an finite state automata. In COOL, the

4	Applicative skills	Database access, graphical user interface ...
3	Agent model related skills	inference engine, behavioral engine, ...
2	Agenthood skills	Knowledge base, conversation management, organizations management
1	Minimal system skills	Communication and skill management

Table 6.1: The four layer of our abstract agent model

need for the introduction of *conventions* between agents to support coordination is proposed. This concept of *convention* must be brought closer to the works of Shoham on *social rules* (Shoham and Tennenholtz 1995) and their contributions on the global performance of the system. Thus, the introduction of this level of interaction management while rigidifying in a certain way the possible interactions between agents, brings guarantees on coordination and allows the reification of these interactions. The table below, inspired by work of Singh(Singh 1996), illustrates the various levels of abstractions within a multi-agent system:

Applicative skills	Business knowledge
Agent models and system skills	Agent oriented design
Conversation management	Interaction oriented design
Message transport	Agent platform (i.e. agents container)

To conclude, we would like to cite a definition suggested by Singh(Singh 1996) of the interaction oriented approach, which characterizes our approach: *We introduce interaction-oriented programming (IOP) as an approach to orchestrate the interactions among agents. IOP is more tractable and practical than general agent programming, especially in settings such as open information environments, where the internal details of autonomously developed agents are not available*. It is the point of view that we adopt, by proposing a pragmatic method for the design and realization of multi-agent systems, relying on the concept of *executable* specification of interaction protocols.

6.3 Interaction oriented design

The heart of our proposal is a formal model to describe interaction protocols, and a transformation mechanism to generate the code that is necessary to the management of these protocols. In order for running agents to be able to exploit these new interactions, we rely on a minimal generic agent model, which authorizes the incremental construction of agent per skills addition. Thus, we will initially present this generic agent model, then we will study the model of specification of interaction protocols and the associated transformation mechanism.

A minimal generic agent model. The basis of our model is on the one hand the interactive creation of agent, and on the other hand a search on the fundamental functionalities of agenthood.

We are not interested in the description of the individual behavior of agents, but rather in the identification of functions that are sufficient and necessary to an agent. Indeed, the management of interactions, the knowledge management or the management of organizations, are not related to the agent model, but are intrinsic characteristics with the concept of agent. In our model, an agent is a container which can host skills. A skill is a coherent set of functionalities accessible through a neutral interface. This concept of skill is to be brought closer to the concept of software component in object oriented technologies. Thus, an agent consists of a set of skills which carries out various parts of its behavior. We identified four layers which are characterized by the various levels of abstraction of functionalities that are proposed (table 6.1).
The first level corresponds to "system" skills, i.e. the minimal functionalities allowing to bootstrap an agent: the communication (emission/reception of messages) and the management of skills (dynamic acquisition/withdrawal of skills)(Routier, Mathieu, and Secq 2001). The second level identifies *agent* skills: the knowledge base, media of interaction between skills and the place of knowledge representation, the management of interaction protocols (cf. following section) and the management of organizations (cf. last section). The third level is related to skills that define the agent model (reactive, BDI...), while the last level represents purely applicatives skills. Thus, the first and the second level characterize our generic minimal agent model. This model is generic with respect to the agent models that can be used, and minimal in the sense that it is not possible to withdraw one of the functionalities without losing a fundamental aspect of agenthood.
A skill is made of two parts: its *interface* and its *implementation*. The interface specifies the incoming and outgoing messages, while the implementation carries out the processing of these messages. This separation uncouples the specification from its realization, and thus makes it possible to have several implementations for a given interface. The interface of a skill is defined by a set of message patterns which it accepts and produces. These messages must be discriminated, it is thus necessary to type them.

$$\texttt{interface := ((}m_{in}\texttt{)+, (}m_{out}\texttt{)*)* where }m_x\texttt{ = message pattern}$$

The typing of message patterns can take several forms: a strong typing, which has the advantage of totally specifying the interfaces, while a weak typing offers more flexibility with regard to the interface evolution. Thus, if the content of messages are expressed in KIF or DAML+OIL, a strong typing will consist of an entire message checking, while a weak typing will only check it partially.

A model of executable specification of interaction protocol. Many works have been done to specify interaction protocols. Recently, AgentUML(Odell, Parunak, and Bauer 2000) was defined like an extension of UML, to specify the conversations between agents, in particular by specializing sequence diagrams in UML. However, these specifications require the interpretation of developers, which must then translate them in their own system. Works of Labrou and Finin(R. Scott Cost and Peng 1999) explore the use of Colored Petri Nets(Jensen 1992) (CPN) to model conversations between agents. In (Mazouzi, Seghrouchni, and Haddad 2002), the same approach is used, but the concept of Recursive Colored Petri Nets is introduced to support conversations composition. Our work follows the same principles: to represent interactions in a global way, and to use a recognized and established formalism. However, contrary to preceding works, our goal is to produce an executable *specification*. i.e., a description of the interaction protocol which can be then directly integrated in a running system. Moreover, CPN are unquestionably adapted to the modeling of concurrent processes, and provide an interesting graphic formalism,

but they are unfortunately not really user friendly. This is why it appears preferable to us to define a language adapted to the modeling of interaction protocols, and to use a projection mechanism that translates this language into CPN.

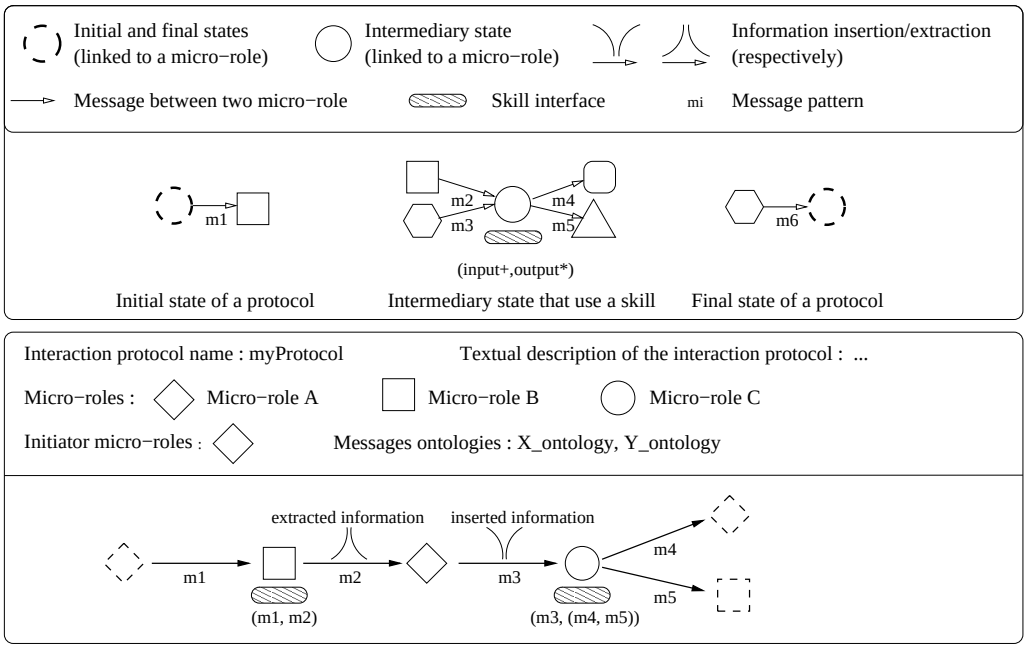

Figure 6.1: Definition of the syntactic elements that constitute interaction protocols

The interaction protocol specification model. The purpose of this model is to ease the specification, the checking and the deployment of interaction protocols within multi-agent systems. On all these stages, the designer has to define the specification, the other stages being automated. For that, we define a formalism representing the global view of an interaction protocol, and a projection mechanism which transforms this global view into a set of local views dedicated to each role. We will initially describe the specification of the global view, before presenting the projection mechanism which generate local views that agents use while a protocol is running. The interaction protocols are regarded here as social laws within the meaning of Shoham(Shoham and Tennenholtz 1995), that means that agents lose part of their autonomy (conversational rules are static), but the system gains in determinism and in reliability.

Our model relies on the concept of skill, micro-role and a graph that represents the state of the conversation. An interaction protocol formally specifies the course of a conversation (regarded as a social law) between various entities, i.e. the nature of exchanged messages, the flow of these messages and skills that entities must implement for each stage of the conversation. These entities correspond to micro-roles, and are characterized by their name and their skills. One uses a graph to represent the course of the conversation: the nodes represent micro-roles which can be associated to a skill interface, and the arcs with a sending of message between two micro-roles (typed by a message pattern).

An interaction protocol is thus defined by the following elements (figure 6.1): the name of the interaction protocol, a textual description of the goal of this protocol, the list of the micro-

roles involved in the interaction and the geometrical symbol which is associated to them, ontologies of exchanged messages, a list of information necessary in input and produced at output, an interaction graph gathering information like the temporal course of the conversation, the synchronization and the nature of the exchanged messages, and the skills that are used by micro-roles.

What it is significant to understand is that the designer has a global view of the interaction: the flow of the messages, their nature (the type of these messages corresponds to the annotations attached to the arcs), needed skills and information used or produced. In addition, all information necessary to the management of the interaction protocol is centralized here and can be used to carry out the generation of the code required to manage this interaction for each micro-role. The designer has thus only to define the interaction protocol by using a graphical tool, the projection mechanism takes care of the generation of descriptions for each micro-role, and the generic agent model can then use these descriptions.

The projection mechanism. The preceding section described the formalism representing interaction protocols, we will now explain the transformation making it possible to obtain a *runnable* specification. The specification of interaction protocols gives to the designer a global view of the interaction. Our objective is to generate for each micro-role a local view starting from this global view, this one could then be distributed dynamically to the agents of the system. The projection mechanism transforms the specification into a set of automata. More precisely, an automata is created for each micro-role. This automata manages the course of the protocol: coherence of the protocol (messages scheduling), messages types, side effects (skill invocation). The implementation of this mechanism is carried out by the generation of Colored Petri Nets. Indeed, we use the color of tokens to represent messages patterns, in addition we have a library facilitating the interactions between generated networks and the agent skills. On the basis of the interaction graph, we create a description of Colored Petri Net for each micro-role, and we transform this textual description to a Java class. This class is then integrated within a skill, which is used by conversation manager skill (level 2 in table 6.1). The interest of this approach is that the designer graphically specifies the global view of the interaction, the projection mechanism generates the skill needed to the management of this interaction. Moreover, thanks to the dynamic skill acquisition, it is possible to add new interaction protocols to running agents of the system.

Knowledge and organization management. The knowledge management and the management of organizations are also mandatory functionnalities of agent, and they should not be enclosed within the agent model. The knowledge management gathers at the same time their representation, the information storage, the means of reaching and of handling them. The implementation of these functionalities is strongly dependent on the used agent model (third level of table 6.1). The concept of organization is necessary to structure interactions that intervene between entities of the system. This concept brings some significant benefits: a means to logically organize the agents, a communication network per defect and a media to locate agents, roles or skills. Moreover, its reification provides a door in the system, making it possible to visualize and to improve interactions between agents(Mathieu, Routier, and Secq 2002).

6.4 RIO : towards an interaction based methodology

In this section, we will present the methodology that we are developing, and which relies on the previously presented concept of *runnable* specification . This methodology falls under the line of GAIA(Wooldridge, Jennings, and Kinny 2000), and thus aims the same applicability. However, GAIA remains too general to easily be able to go from the system design stage to its realization. The purpose of our proposal is to facilitate this transition. The RIO methodology relies on four stages, the two first represent reusable specifications, while the two last are singular with the application (figure 6.2). Moreover, we will not speak about *application*, but about an agent society. Indeed, the RIO methodology proposes an incremental and interactive construction of multi-agent systems. By analogy with our minimal generic agent model, where an agent is a container which can receive skills, we see a multi-agent system like a container that has to be enriched by interactions. We will detail this approach by studying the four stages of our methodology.

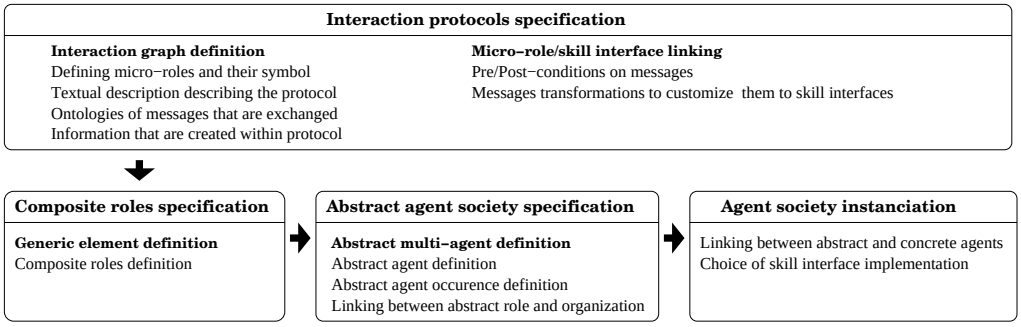

Figure 6.2: The stages of the RIO methodology

Interaction protocols specification. The first stage consists in identifying the involved interactions and roles. Then, it is necessary to determine the granularity of these interactions. Indeed, for reasons of re-use of existing protocols, it is significant to find a balance between protocols using too many roles, these protocols becoming thus too specific, and protocols where there are only two roles, in this case the view of the interaction is no more global. The specification of the interaction protocols can then be done in three ways : either *ex-nihilo*, by specialization, or by composition. Creation *ex-nihilo* consists in specifying the interaction protocol by detailing its cartouche (figure 6.1). Specialization makes it possible to annotate an existing cartouche. Thus, it is possible to specify the cartouche of an interaction protocol such as FIPA CONTRACTNET, its specialization will consist in changing micro-roles names to adapt them to the application, to refine message patterns, and if required to modify insertions/extractions of information. Finally, the composition consists in assembling existing protocols by specifying associations of micro-roles and information transfers. At the end of this stage, the designer has a set of interaction protocols. He can then pass to the description of composite roles, which will allow the aggregation of micro-roles that are involved in complementary interactions.

Composite roles specification. This second stage specifies role models. These models are abstract reusable descriptions. The composite roles correspond to a logical gathering of micro-roles. These patterns define *abstract* roles, which gather a set of consistent interaction protocols. For example, a composite role SUPPLIES MANAGEMENT will gather the micro-role BUYER within the

PROVIDERS SEEKING interaction protocol and the micro-role STOREKEEPER of the interaction SUPPLIES DELIVERY. Indeed, a role is generally composed of a set of tasks which can be, or which must be carried out by the agent playing this role. Each one of these tasks can be broken up and be designed as a set of interactions with other roles. The concept of composite role is thus used to give a logical coherence between the micro-role representing the many facets of a role.

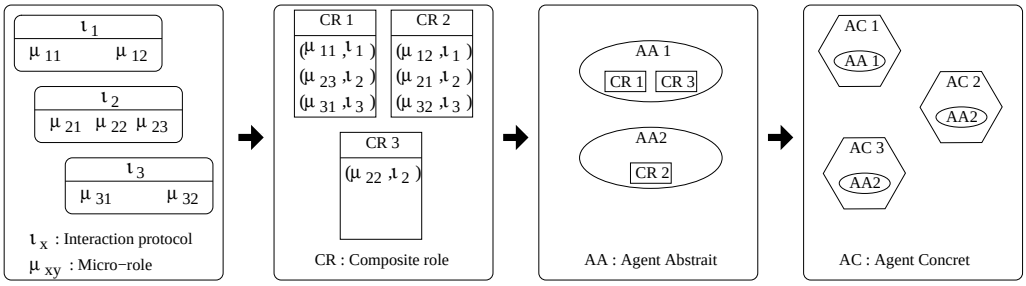

Figure 6.3: Synthetic illustration of RIO stages

Agent societies specification. This third stage can be regarded as a specification of an abstract agent society, i.e. a description of abstract agents and their occurrence, as well as the link between composite roles and organizations. Once the set of composite roles is created, it is possible to define the abstract agents (patterns of agent, or *agent template*), which are defined by a set of composite roles. These abstract agents describe applicative agent models. These models are specific, because they introduce strong dependencies between composite roles. For example, the OFFICE STATIONERY DELIVERY composite role could be associated with the TRAVELLING EXPENSES MANAGEMENT composite role to characterize a LABORATORY SECRETARY abstract agent (fig 6.4). Once abstract agents are defined, it is necessary to specify their occurrence in the system. It means that each abstract agent has an associated cardinality constraint that specifies the number of *instances* that could be created in the system (exactly N agents, 1 or more, *, or [m..n]). The second part of this stage consists in specifying for each abstract agent, and even for the composite roles of these agents, which organization should be used to find their acquaintances. Indeed, when agents are running, they have to initiate interaction protocols, but in order to do that they initially have to find their interlocutors. The organization is used as a media for this search. This association makes it possible to use various organizations for each interaction protocol.

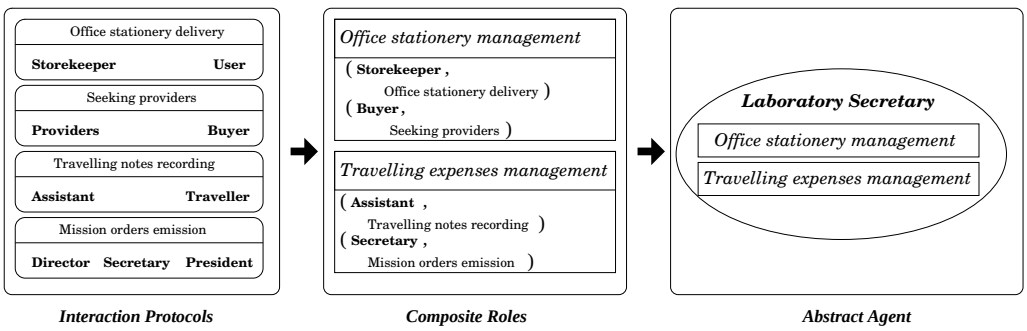

Figure 6.4: The Secretary example

Instantiating an agent society in a multi-agent system. This last stage specifies deployment rules of the abstract roles on the running agents of a system. We have a complete specification of the agent society, that can be mapped on the concrete agents of the multi-agent system. For that, it is necessary to indicate the assignments from abstract agents to concrete ones. Then, the connection between a skill interface and its implementation is carried out. The designer indeed must, according to criteria that are specific to the hosting platform or applicative, bind the implementation with skill interfaces. It is during deployment that the generic agent model is justified as a support to dynamic acquisition of new skills related with the interaction. Indeed, the interaction, once transformed by the projection mechanism, is represented for each micro-role by a Colored Petri Net and its associated skills. All these information are sent to the agent, which adds applicative skills and delegates the CPN to the conversation manager. When an agent receives a message, the conversation manager checks if this message is part of a conversation in progress (thanks to the conversation identifier included in the message), if it is the case, it delegates the message processing to the concerned CPN, if not he seeks the message pattern matching the received message and instantiates the associated CPN. If it does not find any, the message will have to be treated by the agent model.

6.5 Conclusion

We have presented in this article a methodology falling under the line of GAIA, but relying on the concepts of the interaction oriented programming. The basis of the RIO methodology is the engineering of interaction protocols, and more precisely the engineering of *runnable* specifications. For that purpose, we use a tool facilitating the graphical design of interaction protocols, and a projection mechanism that generates the code corresponding to the vision that each participant has of the interaction. By using these specifications, it is possible to create abstractions characterizing the various roles and agents of a multi-agent system: the composite roles, which gather a set of micro-roles, and abstract agents, which gather a set of composite roles. An implementation of this approach is under development, and we use the following technologies: Coloured Petri Nets, DAML+OIL for messages ontologies and knowledge representation, OSGi as component model, and the Java language for the multi-agent platform.

Bibliographie

Barbuceanu, M.; and M. S. Fox. 1995. “Cool: A language for describing coordination in multi-agent systems,” in *Proceedings of the First International Conference oil Multi-Agent Systems (ICMAS-95)*, 17–24, San Francisco, CA.

Brazier, F. M. T.; B. M. Dunin-Keplicz; N. R. Jennings; and J. Treur. 1997. “DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework,” *Int Journal of Cooperative Information Systems*, 6(1), 67–94.

Ferber, J.; and O. Gutknecht. 1999. “Operational Semantics of a Role-based Agent Architecture,” in *Proceedings of ATAL’99*.

Jensen, K.. 1992. “Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use, Vol. 1: Basic Concepts,” in *EATCS Monographs on Theoretical Computer Science*, 1–234. Springer-Verlag: Berlin, Germany.

Kazuhiro Kuwabara, Toru Ishida, N. O.. 1995. “AgenTalk : Describing Multiagent Coordination Protocols with Inheritance,” in *Proc. 7th International Conference on Tools with Artificial Intelligence (ICTAI’95)*, pp. 460–465.

Kendall, E. A.; M. T. Malkoun; and C. H. Jiang. 1995. “A Methodology for Developing Agent Based Systems,” in *First Australian Workshop on Distributed Artificial Intelligence*, ed. by C. Zhang, and D. Lukose, Canberra, Australia.

Mathieu, P.; J. Routier; and Y. Secq. 2002. “Principles for Dynamic Multi-Agent Organisations,” in *Proceedings of Fifth Pacific Rim International Workshop on Multi-Agents (PRIMA2002)*.

Mazouzi, H.; A. E. F. Seghrouchni; and S. Haddad. 2002. “Open protocol design for complex interactions in multi-agent systems,” in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, 517–526. ACM Press.

McCabe, F. G.; and K. L. Clark. 1995. “April – agent process interaction language,” in *Intelligent Agents: Theories, Architectures, and Languages (LNAI volume 890)*, ed. by M. Wooldridge, and N. R. Jennings, 324–340. Springer-Verlag: Heidelberg, Germany.

Odell, J.; H. Parunak; and B. Bauer. 2000. “Extending UML for Agents,” .

R. Scott Cost, Ye Chen, T. F. Y. L.; and Y. Peng. 1999. “Modeling agent conversations with Colored Petri Nets,” in *Third Conference on Autonomous Agents (Agents-99), Workshop on Agent Conversation Policies*, Seattle. ACM Press.

Routier, J.; P. Mathieu; and Y. Secq. 2001. “Dynamic Skill Learning: A Support to Agent Evolution,” in *Proceedings of the AISB’01 Symposium on Adaptive Agents and Multi-Agent Systems*, 25–32.

Shoham, Y.. 1993. “Agent-Oriented Programming,” *Artificial Intelligence*, 60, 51–92.

Shoham, Y.; and M. Tennenholtz. 1995. “On Social Laws for Artificial Agent Societies: Off-Line Design,” *Artificial Intelligence*, 73(1-2), 231–252.

Singh, M. P.. 1996. “Toward Interaction-Oriented Programming,” Discussion Paper TR-96-15.

Wooldridge, M.; N. Jennings; and D. Kinny. 2000. “The GAIA Methodology for Agent-Oriented Analysis and Design,” *Journal of Autonomous Agents and Multi-Agent Systems*.

Chapitre 7

Interaction-Based Approach for Game Agents

Damien Devigne, Philippe Mathieu et Jean-Christophe Routier
Laboratoire d'Informatique Fondamentale de Lille
Cité Scientifique 59655 Villeneuve d'Ascq Cedex
{devigne,mathieu,routier}@lifl.fr

Abstract.[1]

Most often, agents in simulations are based on reactive models. Such systems do not plan actions for the agents and are too limited to express complex realistic behaviour. We propose here an agent model for spatially situated simulations, like computer games are. The involved agents are cognitive (or deliberative) ones: they are able to build plans and to adapt them according to the dynamics of the simulation. Our main goal is to obtain believable behaviours for the agents in simulations.

Thus we propose a generic model for cognitive situated agent in simulations of which video games are a typical example. The proposed ideas promote re-usability from one simulations to another and then favour good software design.

The two main problems can easily (and without surprise) be identified: *how to represent knowledge ?* and *how to build plan using this knowledge*. To solve the first we propose to describe the laws that manage the simulated world in term of interactions that can be performed by some agents and suffered by others. Concerning the second problem, we propose a planning algorithm that is based on the interactions and take into account the facts that agents are situated and that plans must be executed in a situated environment that is in permanent evolution. Thus the plans are actually incrementally built through partial replanning.

[1] Article publié dans *Proceedings of ECMS/SCS/IEEE 19th European Conference on Modelling and Simulation. ECMS 2005. pp. 705-714. Riga. 2005.*

7.1 Introduction

The simulation of rational or believable behaviour is, quite from the beginning of Computer Science, one of the major objectives of this field, especially in the purpose of Artificial Intelligence, ie. to reproduce the intellectual abilities of the human being. This issue has been initially addressed from a logical and linguistic viewpoint, which raises huge difficulties. In addition, it appeared rapidly that a large number of AI *applications* did not require a human-like intelligence level.

Now this is not still true. New research fields need a human-like level AI, not in order to *solve complex problems*, but rather to *develop harmonious interactions* with human partners: for instance, social robotics (Brooks and al. 1999), the use of virtual reality in teaching or training, or the large domain of video games (Nareyek 2004, Magerko, Laird, Assanie, Kerfoot, and Stokes 2004) are illustrative examples.

According to J. Laird, the latter constitutes a "Killer Application" for human-level AI (Laird and van Lent 2000). The characters involved in video games have indeed to be perceived as autonomous entities with increasing realistic behaviours. They have to be *convincing*, thus their behaviour must comply with the rational expectations of their partner or opponent human players. They also need to adapt to new situations, acquire additional abilities throughout the game, etc. In addition, team strategies are also often useful. In order to develop such kind of interactions, the agents have to make the human observer thinks that, in order to achieve their goals, they behave in an "intelligent", "rational" way, ie. like the human would have behaved. Our research aims at this goal : modelling believable characters for simulations in general and games in particular. Let us precise at this point that we do not consider here the problem of the simulation of "emotions" (Allbeck and Badler 2003), but consider "simulation" in the sense of "simulation of sequence of actions".

In the case of video games, theses "cognitive" constraints meet additional "economical" ones: the time needed for developing the game. This depends to a large extent on the reusability of previous works. In the case of character's AI, it is often difficult to reuse from one game to another or even, inside a game, from one character to another. This is mainly due to the almost systematic use of scripts whose drawbacks have been many times underlined (Tozour 2002), and that are only partially solved with dynamic scripting (Spronck, Sprinkhuizen-Kuyper, and Postma 2004).

More generally, this domain of modelling believable characters combines difficulties that can be encountered in classical AI (knowledge representation), in distributed AI (coordination of agents having most of the time different individual goals), and in Software Engineering (reusability of conceptual and software tools).

Some propositions have been done concerning agents and games (Nareyek 2000), and most of them concern reactive agents (Niederberger and Gross 2003). But reactive agents, while effective in several cases, offer limited behaviours. Indeed their behaviours are "short term directed" and not "goal oriented". Their ability to perform some tasks depends on the immediate surroundings and does not result of wilful acts. Our proposition aims at offering cognitive (or deliberative), driven by goals, proactive agents. Let us precise that we do not consider the interesting problem of behaviour's learning (Ponsen and Spronck 2004).

We promote a generic approach that assumes that a single formalism can be used to design realistic (ie. believable) behaviours in an artificial world in general and in games in particular. Thus from one simulation to another the cognitive behavioural engine stays the same even if the context changes and the behavioural components can be (partially) reused. The main principle is

to base the dynamics of the simulations (and the knowledge representation too) on interactions between agents: some agents can perform interactions and other agents can suffer them. Our main goal is then to provide a uniform and generic frame for simulations. Our target is multi-agent spatially situated simulations like most of computer games are. More precisely role-playing games are a privileged target for our work. Agents are situated in an environment provided by an euclidean space. This space has a "geography" (a "map") and notions like "position", "neighbourhood", "move", "distance",... have a meaning. The agents have, at each moment, a partial perception of their environment. This environment is dynamic and concurrent, and thus non monotonic (insofar once an agent knows some data, this knowledge can become wrong - or irrelevant - after some times). The agent's knowledge about the environment is incomplete and can be wrong. The abilities of the agents can differ. Agents may have cognitive abilities. Some of them have objectives (or goals) that direct their actions in the environment. To achieve its goals each cognitive agent has a behavioural engine. This engine chooses at every moment an action to do. This action must allow the agent to fulfil its goals "at best" and rationally. The "rational" notion of a behaviour is rather subjective and is actually evaluated by a jury that is external to the simulation. Thus, we will consider as rational a behaviour if the decision to perform an action could have reasonably been taken by a human which would have had the same information than the agent.

First section concerns knowledge representation. We first present the environment that models the geography of the simulated world, second the agent model is described. These two points are not sufficient, we must precise how these agents can have an influence on the environment, that is, what the laws that rule the world are. This knowledge representation is crucial since it is used by the agent's behavioural engine in order to act in the environment. We use what we call interactions to achieve this. The following section is dedicated to the agent's behavioural cognitive engine. We present the structure of this engine and more precisely the planning and re-planning algorithm.

7.2 Knowledge Representation

Simulations consist in *agent*s that evolve in an *environment* and *interact* with the environment and other agents, according to the laws that rule the environment. Therefore, it is essential to describe these different core notions. We will first present the environment that is the basis of the situated side of the simulations. Second, we define the agents involved in our simulations, they are divided in passive (closer to "things") and active agents that are responsible of the dynamics of the simulation. These agents can suffer or perform interactions that are described in the third part. They represent the atomic knowledge beans used by the deliberative agent to act.

7.2.1 Environment

The environment describes the geography of the simulation. It provides the support to situate the agents and then to control the possibility of the realisation of some of their actions, when the notion of neighbourhood has an importance for example. The environment gives a meaning to the notion of *move* for an agent, although it is simple, this notion is full of importance since it impacts a lot on the dynamics, at least the visible one, of the simulation and it makes our concerns different from the pure planning problems. It is the environment too, that is in charge to determine which information can be perceived by an agent.

We represent the environment by a graph where vertices are *places* and edges denote *path* from one place to another.

```
environment  :=  {place*,path*}
path         :=  (place_origin,
                  place_destination,
                  condition)
condition    :=  boolean expression
```

A place is a geographical elementary area. The granularity of a place depends on the simulation, the only constraint is that inside a place there is no restriction neither for moves, nor for perception (restrictions due to the other agents, like collision problems, excepted). A place can represent a room, a town or any other part of the environment, and inside a place the position of an agent can be managed discretely or continuously depending on needs.

A path denotes an oriented transition between two places. It is defined by the places that it links, and a condition that must be satisfied if an agent wants to use this path. The edge is oriented and the condition to go from some place a to a place b is not necessarily the same than the one to go from b to a.

This formalism allows to describe, for example, that a door between two rooms must be opened if we want to go from one room to the other, or that an agent must be able to swim to cross a river between two fields. In this last case, our approach allows, depending on needs, to choose to model or not the river with a place. It depends on whether the crossing of the river has a meaning in the simulation (see figures 7.1 and 7.2).

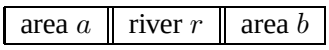

Figure 7.1: River is modelled. The paths are: (a,r,*"agent can swim"*), (r,a,*true*), (b,r,*"agent can swim"*), (r,b,*true*)

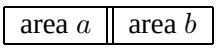

Figure 7.2: River is not modelled. The paths are:(a,b,*"agent can swim"*),(b,a,*"agent can swim"*).

The environment is the place where the agents are situated. It plays the role of a reference for the agents (then in this context the environment can not itself be an agent). Each agent is located in a place and can not be in a path. If the path must be put into concrete form, this must be done using a place, like we have seen it with the river example. Then a place is mainly characterised by the set of the agents that belongs to it.

The relative position of the agents inside a place (when this has a meaning) will be managed by the place itself, and is a parameter of the simulation.

7.2.2 Agents

The agents involved in the simulations we are interested in, are situated in a place of the environment. It is the environment that is in charge of the creation or removal of an agent in the simulation, even if the decision of these creations or removals is the result of the behaviours of the present agents.

We call *agent*, every entity that has some relevance in a simulation, that is, that can have an influence over the simulation. Among these agents, we distinguish two special classes: the *passive agents* and the *(pro-)active agents*. We use the terms of *inanimate* and *animate* agents too. It is for the latter that the notion of behaviour as a meaning.

In a rather natural and classical approach, agents are defined by a set of properties, a property being a pair (*name*, *value*). However, we will refine this definition (see Figure 7.3) and precise some particular properties imposed to our agents. We spend no time on the *name* property which allows to have a symbolic reference of the agent, but we rather insist on what characterize the agents: their abilities expressed by interactions.

Our agents (passive or active) are, at first, characterised by the actions (in the following we rather use the term "*interaction*" which denotes the way an action is coded) they can suffer. A *tree* agent could be cut, a *door* agent could be opened or painted, a *sheep* agent could be sheared, etc. We name *can-suffer* this property, the associated value is the list of interactions that the agent can suffer (that is for which he can be a target). The interactions are presented in the next section.

We must now study the particular case of the active agent. It is easy to guess that these agents have the possibility to interact with their environment, that is with the other agents (seen through their "passive" facet). These abilities are expressed by a collection of interactions they can perform, and defined in a property; we name *can-perform* this property.

However, this property remains a declaration of abilities. In order for an active agent to have an impact over the simulation, he must be provided with a behaviour engine that takes at every moment the decision of the action undertaken by the agent, and then of the used interaction. This decision depends on the context. This engine is influenced/directed by the existence of goals for the agent. The section 7.3 is dedicated to the presentation of this engine.

Confusion must not be made between "active" or "animate" agent and the modelling of "living" entity. Thus, if in a simulation there is a machine which produces regularly some objects o, this must be modelled by an active agent whose goal would be the production of agents corresponding to o and whose behaviour would be the satisfaction of this goal.

```
agent          :=  passive-agent | active-agent
passive-agent  :=  {  ( "name", Symbol),
                      ("can-suffer", {interaction* }),
                      property*}
active-agent   :=  passive-agent ∪
                    { ("can-perform", {interaction*}),
                    ("goals", goal*),
                    ("memory", (degraded) environment),
                    ("engine", engine)}
```

Figure 7.3: Definition of an agent

We can point out another particular property: the memory of the active agent. It represents the knowledge base for all the information gathered by the agent concerning the environment: the topology of the environment, the other agents (their position and state), etc. This memory is a degraded environment insofar as it corresponds to the data the agent knows about the environment. This knowledge can be incomplete, for instance the agent does not know that others exist. It can

even be wrong, for instance because the agent is not necessarily aware when other agents act and modify the state of some entities.

To come back on what we said at the beginning of this paragraph, an element must be considered as being represented by an agent in a simulation (ie. has an influence over the simulation), if and only if either it is active and the list of the interactions it can perform is not empty, and there exists at least one possible target for one of these interactions, or it is passive and the list of the interactions it can suffer from is not empty, and at least one active agent can perform one of these interactions.

It results from this definition that the interactions play an essential role in our simulations. Agents are different because they perform or suffer different interactions. Moreover the interactions define the "laws" of the simulated environment, and then play a central role in knowledge representation. We now define this notion.

7.2.3 Interactions

Interactions are the backbone of our simulation model, we could even speak of *interaction oriented simulations*. These interactions are the basis of the knowledge representation in the simulations. They define the laws of the modelled world, that is the actions that can be performed in the simulations. They are central since the agent's engine uses them to build the agent's behaviour.

These interactions are the units of knowledge that describe the laws of the simulated world. They represent a declarative knowledge. A consequence is that an interaction must not, very special case excepted, be attached to one simulation but must represent a rather universal knowledge. This constitutes a difficulty in regards with the representation of these interactions, but allows to reuse them from one simulation to another one.

We characterize an interaction by an *actor* and a *target*. The *actor* is instantiated by an active agent who can perform this interaction and the *target* is any agent who can suffer from this interaction.

An interaction is defined in a rather classical way as presented in figure 7.4. The *name* is a unique identifier. The other three parts are:

- the *condition*, it tests the current context of execution of the interaction and consists mainly of tests on values of target or actor properties.
- the *guard*, it checks general conditions for the interaction applicability, typically it defines that to be fired an interaction requires that the distance between the target and the actor must be less than some given value.
  The guard is separated from the condition since it corresponds to the knowledge due to the geographically situated feature of the simulations. In a non situated context, one would have only the condition and action parts. The guards are at the origin of the moves in the plan, and this is these moves that are indeed specific to situated problems. Thus, we do not express explicitly in an interaction that the agent has a move to do in order to fire it, we want the agent to plan it when required by a guard.
- the *action*, it describes the consequence of the interaction, it can be a change in the state of the actor and/or of the target (ie. a change of the value of a property), and/or the activation of an environment action (like the creation of an agent).

```
actor                the agent who performs the interaction
target               the agent who suffers the interaction

interaction          (name, condition, guard, action)
name                 Symbol
guard                d op Integer
d                    distance between actor and target
op                   = | > | < | ≤ | ≥
condition            test_property | predicate_primitive(args)
test_property        { actor | target }.property_name op Value
predicate_primitive  primitive
action               affect_property | primitive
affect_property      { actor | target }.property_name = Value
primitive            { actor | target }.primitive_name(args)
primitive_name       Symbol
property_name        Symbol
```

Figure 7.4: Definition of an interaction

Some interactions does not naturally obey to this schema of interaction between a target and an actor. This is the case, for example, for the "*sleep*" action. However, in this case it suffices to consider that the actor and the target are the same agent: the actor decides to make the target (himself) sleep, and thus he changes the state of the target. Such action can then be represented with the same interaction model.

More generally the consequence of an action is a change in the state of the target or actor. Thus to *open* an object (door, chest, window, etc.) makes it changing from *opened* state to *closed* one. The precise essence of the target is of no importance here, this knowledge must then be represented in a "universal" way by the interaction:

$$\textit{open:} \quad \begin{cases} \texttt{condition} & = & \text{“}\texttt{target.opened} = \texttt{false}\text{”} \\ \texttt{guard} & = & \text{“}\texttt{distance}(\texttt{actor}, \texttt{target}) < 1\text{”} \\ \texttt{action} & = & \text{“}\texttt{target.opened} = \texttt{true}\text{”} \end{cases}$$

Such an action can be used by an engine to generate a plan such that: "*to push a button in the next room I must* open *this obstacle*" (or more precisely the knowledge would be *the obstacle must be opened*, and when this is not satisfied the given plan is produced). Whether this obstacle is a door or a window or anything else that is openable, the plan remains valid.

A problem arises when considering more "specific" agents. For instance, let us consider the case where the obstacle is a *lockable* door. To push the button, the above plan is still valid with respect to the knowledge that must be used and then with respect to the behaviour engine. The difference exists only in the condition for the execution of the action. This lockable door requires that, in its particular context, something like *target.lock = false* must be satisfy too. Then, from an abstract point of view the plan is still valid, but the *open* interaction must be understood as "make the door change from *closed* to *opened* state *when it is unlocked*". The problem is then how the same abstract plan (ie. "open the door to push the button") can receive different solutions (just "open" or "unlock and then open") depending on the target (whether it is lockable or not). To

have two different interactions, one named *open-when-lockable* (or anything else) and the other named *open* is not relevant. As a first consequence this leads to a multiplication of interactions and implies that the active agents must be finely tuned. Moreover, the agent engine must take into account the different possible cases, although they conceptually represent the same action (*open* here). Thus it is more than probable that we would fall again into one of the major pitfall of the script approach for designing agent's behaviour.

Therefore, our proposition consists in the possibility to specify at the target level (ie. the agents having the considered interaction as a "*can-suffer*" one), the specific process. One can notice that only the nature of the target requires a change in the manipulation, not in the plan. During an interaction between such a target and an actor, the target "tells" to the actor the particular knowledge to be used while interacting with this target. For the actor (active agent), there is still only one generic interaction. Thus, the *can-suffer* property of a *lockable-door* agent contains the *open* interaction, with a specialization of the condition like: *target.locked = true*. Thus, when an actor tries to interact, using *open*, with this door, he gets the full condition "*target.opened = false and target.locked = false*". This leads the actor to (try to) unlock the door before it can open it. This can be seen as a kind of inheritance for interaction. It offers to the game designer the possibility to add new targets that specifies an existing one in order to take into account some particularity of the simulated world (for instance a *lockable door* that specifies a *door*). And this specification does not require the possible actors to be modified, at least their engine must not be changed. This flexibility eases the design of simulations and the ability to reuse interactions and agents from one simulation to another.

7.3 The Agent Engine

This section details the engine of the active agents. We first present the structure of the "mind" of the agent and the dependences between the different elements and second the planning and behaviour engine.

7.3.1 The Agent Structure

The decision cycle applied by the agent is presented in the figure 7.5. Continuously the agent perceives its environment. The acquired information are forwarded to an update module that can influence the memory and the currently established plan. An action is then chosen and the agent tries to execute it in the environment, and so on. To apply this cycle the agent is provided with a "mind".

The "mind" of our active agents is made of several modules: a knowledge or beliefs base, a new information management module, a planning engine, an action selection module and an execution module. The articulation between these parts is illustrated in Figure 7.6.

The Knowledge Base.

The knowledge of the agent can be divided in two. On one side, the knowledge about the actions the agent can do and on the other side the knowledge about the environment he belongs to.

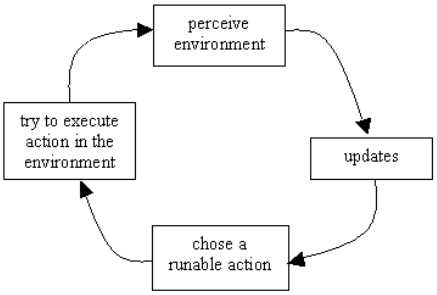

Figure 7.5: Agent decision cycle.

The first is defined by the set of all the interactions that the agent can perform. These interactions represent the absolute knowledge of the agent about an environment independently of any given particular context. They are the basis of the behaviour engine to build plans.

The second corresponds to a base of beliefs and is called the *memory* of the agent. It is a contextual knowledge. It evolves according to the information perceived by the agent. It consists in the knowledge concerning the geography of the environment and in the information about the other agents. The memory is like a degraded environment and corresponds to the perception that the agent has of the environment. In the memory, some of the information can be marked *unknown*. Every known information is timestamped, this helps to estimate a confidence in the data: the older an information concerning the position of a mobile agent is, the less confident it is. The information in the memory are used by the behaviour engine to determine the one among the *can-perform* interactions that must be applied in order to achieve goals. These information are beliefs and not absolute knowledge, consequently when the agent tries to perform an action for which it believes all the conditions are satisfied, it is necessary to check if it is indeed the case in the environment.

Perception.

An innate and absolute knowledge of the environment in which the agent evolves will not produce realistic behaviour. Then it is necessary to provide the agent with a way to perceive new information while he is acting. Actually we content on a simple “visual” perception. The agent perceives the information of the environment that are inside its field of view (whose shape and radius can be changed at will). The perceived information are forwarded to the new information management module that is in charge to manage their influence. Since the perception module is only in charge of the perception and not of the treatment of the new information, it is easy to extend it to new kind of perceptions such as sound.

The update module.

As we previously say, the new information management module is in charge of the new perceived information. It is a kind of short term memory. It operates on two levels: first the memory in order to update the beliefs, and second the planning engine in order to adapt, if needed, the current computed plan through a partial re-planning. This is detailed in the following.

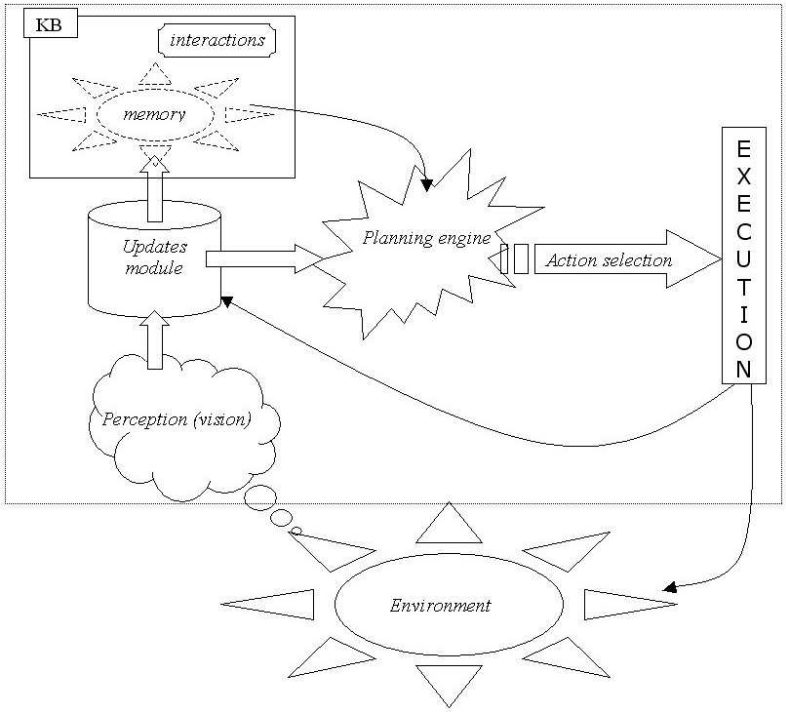

Figure 7.6: Different elements of agent mind.

Planning, selection, execution.

These points will be more detailed in the next section. The engine is in charge of the resolution of the objectives of the agent. It uses the *can-perform* interactions of the knowledge base to build a plan of actions according to the memory. The built plan is valid according to the memory but can be wrong in the environment, this is checked at the execution step. This plan determines at every moment which actions the agent can undertake, an action selection strategy is then applied to choose the next effectively fired action. This strategy can be changed from one agent to the other to obtain different behaviours and then different individuality, even if the planning engine is the same. Once the action is chosen, the agent tries to execute it. Either the beliefs of the agent were right and the action is effectively performed in the environment, or they were wrong and the action can not be done and the agent must update its knowledge.

7.3.2 The Planning

The planning algorithm we use is a kind of backward chaining. To fulfil its goal, among all the interactions that it can perform, the (active) agent searches those that can help to achieve it, and then selects one. If the conditions of this action are satisfied (according to the agent memory), the (inter)action can be fired and the plan is done. Otherwise, the non satisfied conditions become new goals that need to be planned.

The goals.

There exist two kinds of goals. First the *interaction-goals*, they correspond to an (inter)action that the agent wants to execute. The target of this interaction can be less or more precisely given: from a named agent to any agent that can suffer the interaction, as shown in the next table:

goal	*type of target*
`eat(apple_12)`	a given precise apple
`eat(an apple)`	any apple
`eat(*)`	any eatable (ie. "who can-suffer from eat") agent

Second, the *condition-goals*, they correspond to a condition that the agent wants to bring to true. For instance:

`actor.energy > 100` "*having actor energy to be greater than 100*"

Planning tree.

In a rather classical way, the plan produced by the backward chaining can be viewed as a tree. The nodes are made of the different goals and subgoals encountered during the resolution. Some are interaction-goals, others are condition-goals. Thus this tree is an AND-OR tree. AND-nodes correspond to condition-nodes (for condition-goals) and OR-goals to interaction-nodes (for interaction-goals).

Condition-nodes and interaction-nodes: A condition-node has sons only if its condition is not satisfied. These sons are interaction-nodes built from the interactions whose action part offers a way to satisfy the condition (or to approach this satisfaction, for example by increasing the energy for the above given condition-goal example). The tree leaves are the satisfied condition-nodes (ie. whose conditions are satisfied).

The interaction-node's sons are built from the conditions that can be found in the condition and guard parts of the interaction: from these, condition-nodes are built. These sons are always built. An interaction-goal is said to be satisfied when all its sons are satisfied, the associated interaction is then declared runnable.

These correspond to the general cases, however since the simulations take place in situated environment, moves must be taken into account. They must receive particular considerations as discussed in (Devigne, Mathieu, and Routier 2004), this leads to introduce *move-nodes*

Move-nodes and exploration-nodes: *To move* or *to explore* the environment correspond for the agent to execution of interactions. The associated nodes must then be present in the planning tree as particular cases of interaction-nodes.

The exploration case can be reduced to the move case. To explore the agent must indeed make move towards a chosen location. The existence of the exploration-nodes are justified by the need to choose the targeted position before making the move. The agent must then apply its own exploration strategy to make its choice. Thus in the following we will only concentrate on the move case.

```
actor = the agent that builds the plan

this = interaction-node
expand()
  for each condition c in this.condition and this.guard
    newNode = createConditionNode(c)
    this.sons.add(newNode)
    newNode.expand()

this = condition-node
expand()
  If this.condition.isSatisfied()
    then finished
    else
              // gets the action that allows to solve condition
      BA = this.condition.getBackwardAction()
              // the list of can-perform candidates for BA
      LI = actor.getCanPerform(BA)
      for each I in LI
        if I is "moveTo"
          then this.sons.add(createMoveNode())
        else
              // list of known agents that can suffer I
          lAgents = actor.getKnownAgents(I)
          for each a in lAgents
            if actor.execute(I,a) satisfied this.condition
              newNode = createInteractionNode(I,a)
              this.sons.add(newNode)
              newNode.expand()
            end if
        end if
        if this.sons.isEmpty()
          then this.sons.add(create-exploration-node)

this = move-node
expand()
  path = actor.computePath()
  If path = null            // no path found
    then this.sons.add(createConditionNode("false")
    else
              // list of conditions≠"true" on paths
      conditionsList = path.getPathsConditions()
      If conditionsList.isEmpty()
        then this.sons.add(createConditionNode("true")
        else for each condition c in conditionsList
          newNode = createConditionNode(c)
          this.sons.add(newNode)
          newNode.expand()
```

Figure 7.7: Node's expansion algorithm.

One problem is: what are the condition-nodes sons of a move-node? This problem amounts to ask what are the conditions that must be satisfied to make a move possible. To a move corresponds a computed path that is a sequence of elementary paths presented in the "Environment" section. With these elementary paths come conditions. A move is possible if these conditions are satisfied. With these conditions we create condition-nodes that become the sons of the considered move-node.

A classical backward chaining.

The planning tree is built according to the algorithm presented (in broad lines) in Figure 7.7. Every calculus are based on the memory (ie. beliefs base) of the agent. It is in particular the case when the agents checks a condition or computes a path for a move. Therefore, the computed plan is valid according to the agent memory, but can be wrong once it faces up to the reality of the environment.

For the exploration-nodes, the principle is roughly the same once the exploration strategy has provided a place to reach.

Every details are of course not presented in this algorithm. In particular if the same (sub)goal occurs more than one time during the planning, the corresponding node is not expanded twice, it is shared by its fathers. The tree is then an oriented graph.

But, actually, our algorithm builds the plan in an incremental way as we will see in section on replanning. Indeed, according to perceived information, the plan is adapted: the plan's tree is locally modified and not fully rebuilt.

A small example To illustrate the different points described in the previous paragraph we will consider a very small and simple example (for instance, we do not consider the *open* interaction). We consider a world with two places/rooms separated by a door d (see figure 7.8), the path between these two rooms has the condition "d.*locked=false*". Four interactions define the laws: *unlock*, *take*, *move*, *push* (see Table 7.1). In the world are an active agent a that can perform these 4 interactions, and three passive agents, the door d that can suffer *open*, a key k that can suffer from *take* (and can be used to unlock d) and a button b that can suffer from *push*. The goal of the agent is to push on b. The figure 7.8 presents the planning in two different situations.

As one can see, different plans are obtained depending on the context. Moreover in this case, this is because the agents are situated in the environment that two different plans exist. Indeed, it is because the actor must move near the button in order to push it that the state of the door is of importance. It results that the notions of neighbourhood and distance have a direct influence on the produced plan.

A partial replanning.

Active agents evolve in a dynamic environment. They establish a plan according to their knowledge, that can prove to be incorrect and then they can then be brought to adapt the computed plan according to new perceived information. These information can be of several types:

- *a new information*: the agent learns that a so far unknown information exists. It is the case when the agent sees a new place, meets another agent for the first time, gets a new goal, discovers a condition on a path of the graph, etc.

$$
\begin{array}{rl}
\textit{unlock:} & \left\{ \begin{array}{rcl} \textit{condition} & = & \text{“}\texttt{target.locked} = \texttt{true}\text{”} \\ & & \text{“}\texttt{actor.own}(\texttt{target.key})\text{”} \\ \textit{guard} & = & \text{“}\texttt{distance}(\texttt{actor}, \texttt{target}) < 1\text{”} \\ \textit{action} & = & \text{“}\texttt{target.locked} = \texttt{false}\text{”} \end{array} \right. \\
\textit{take:} & \left\{ \begin{array}{rcl} \textit{condition} & = & \texttt{true} \\ \textit{guard} & = & \text{“}\texttt{distance}(\texttt{actor}, \texttt{target}) < 1\text{”} \\ \textit{action} & = & \text{“}\texttt{actor.own}(\texttt{target}) = \texttt{true}\text{”} \end{array} \right. \\
\textit{push:} & \left\{ \begin{array}{rcl} \textit{condition} & = & \texttt{true} \\ \textit{guard} & = & \text{“}\texttt{distance}(\texttt{actor}, \texttt{target}) < 1\text{”} \\ \textit{action} & = & \text{“}\texttt{target.pushed} = \texttt{true}\text{”} \end{array} \right. \\
\textit{move:} & \left\{ \begin{array}{rcl} \textit{condition} & = & \texttt{conditionsfoundinpath} \\ \textit{guard} & = & \\ \textit{action} & = & \text{“}\texttt{distance}(\texttt{actor}, \texttt{target}) < 1\text{”} \end{array} \right.
\end{array}
$$

Table 7.1: Definitions of the interactions (adapted - but without distortion - to shorten the example)

- *a modification of an existing information*: it concerns mainly modifications about the state of known agents, a property value change or a position change. The position change information covers three situations:

known→known:	we thought agent at a position and we see it at another
known→unknown:	we thought agent at a viewed position and it is not there
unknown→known:	we did not know where the agent were and we now see it

To each of these situations is associated an event that describes what must be modified (changed or added) in the agent memory. The table 7.2 lists these events, one can easily guess what they are according to the previous paragraphs.

new information	NewGoal, NewPlace, NewPathCondition, NewAgent, NewInteraction
modification	AgentModified, AgentMovedKK, AgentMovedKU, AgentMovedUK

Table 7.2: List of events for new information

These events are transmitted to the update module who is in charge to take them into account and to consider their impact. First, the memory of the agent must be modified: either a new knowledge is added, or an existing data must be corrected. Second, because of the changes, it is possible that the currently established plan must be adapted. It is the new information management module that is in charge of forwarding these information to the planning engine.

However, a new information concerns only a portion (even none) of the planning tree. Therefore, it is neither reasonable, nor efficient, to rebuild a new plan for every new information. Indeed, even if it is established that, in theory, no efficiency gain can be guaranteed while using plan reuse rather than new plan generation (Nebel and Koehler 1993), in practice improvements

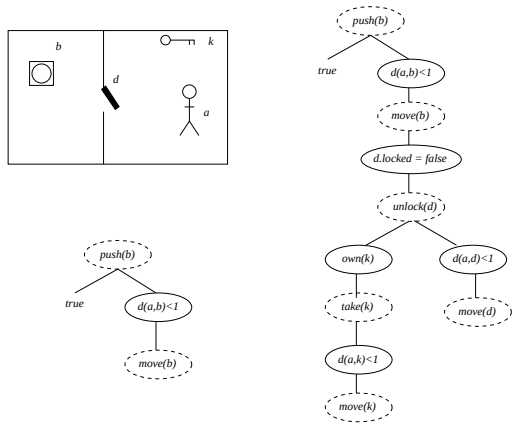

Figure 7.8: An active agent a is situated in an environment where are also 3 passive agents: a door d, a button b and a key k. The goal of a is to push on b. a must build a plan to achieve it. A plan can be drawn as a tree, nodes due interaction-goals are drawn with dashed lines and nodes due to condition-goals with solid lines. Depending on the execution context, different plans can be obtained. Left is the tree obtained when d is not locked and right is the case where d is locked. a must adapt its plan to the context.

can be expected. Indeed, our context of dynamic simulations corresponds to the case where the agent perceives very frequently slight changes of its knowledge base.

In particular, this is due to the fact that the agent engine uses uncertain information: the planning is based on the information that are in the memory. But, since simulations occur in open dynamic environments, then the built plans are correct with respect to the memory of the agent, but can be wrong once confronted with the real environment, at the execution step. Therefore only partial and local adaptations can be expected in most of cases. Our experiments confirm that.

Our approach is then to top-down propagate events from root to leaves in the planning tree. Each node checks if it is concerned by each event, first because it is the good type and second because additional conditions are satisfied. Checking these conditions is very fast, therefore propagation costs not too much time and in particular less than a replanning when events have no impact. Thus, only the appropriate nodes are updated (collapsed, re-unfolded, adding or removal of subnodes, etc.).

It would probably be a bit tedious to enumerate all the cases and conditions for which a node is affected by an event. Thus we give only two examples. For instance, a new information that affects the graph topology or agent positions can (but not systematically) have an impact on move-nodes. Indeed a new path can “appear” or at the opposite a computed path can become blocked, in these cases, the move-node should be re-expanded. In a similar way, a new met agent can affect a condition-nodes. For instance this can be the case if this agent can be the target of an interaction that helps to solve the condition (that is one of the interactions stored in the *LI* list seen in the algorithm presented in Figure 7.7). In this case a new interaction-subnode must be added and unfolded. Other cases are similar.

Using this principle, there is no need to recompute the plan at each step. It is indeed probable that only few new information occur each step, and this is not necessarily of importance for the agent. But this principle helps the agent to remains reactive too and to adapts as soon as

it is necessary and relevant. Moreover this partial and local tree modification leads the plan to be incrementally built, change after change.

7.4 Conclusion

The design of simulations of behaviours that will be perceived and evaluated as believable by a human external observer is not an easy problem. The solutions proposed by the reactive systems are not generic and reusable enough.

We endeavour to propose a general model for the simulation of behaviours. The dynamics of this model relies on the description of interactions that can be performed or suffered by some agents that are situated in a dynamic environment. An individual generic behavioural engine uses these interactions to propose a plan of actions to the deliberative agents.

A natural application of this work are computer games. Targeted games could be action and role-playing games, where one needs complex believable non-player characters to increase the quality of the simulated world and the interactions of the player with it.

Our current work concerns the application of this model to team of agents (Devigne, Mathieu, and Routier 2005). It is clear that it is a challenge for simulations, and games in particular, to be able to design groups of characters that act together to fulfil common objectives.

Bibliographie

Allbeck, J.; and N. Badler. 2003. “Representing and Parameterizing Agent Behaviors,” in *Life-like Characters: Tools, Affective Functions and Applications.*, ed. by H. Prendinger, and M. Ishizuka, 19–39.

Brooks, R. A.; and al.. 1999. “The COG Project: Building a Humanoid Robot,” in *Computation for Metaphors, Analogy, and Agents*, vol. 1562 of *LNCS*, 52–87. Springer.

Devigne, D.; P. Mathieu; and J.-C. Routier. 2004. “Planning for Spatially Situated Agents in Simulations,” in *Proceedings of the 2004 IEEE/WIC/ACM International Joint Conference IAT’04*, 385–388.

Devigne, D.; P. Mathieu; and J.-C. Routier. 2005. “Team of cognitive agents with leader : how to let them autonomy,” in *Proceedings of 2005 IEEE Symposium on Computational Intelligence and Games*.

Laird, J. E.; and M. van Lent. 2000. “Human-level AI’s Killer Application: Interactive Computer Games,” in *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence.*, 1171 – 1178. AAAI.

Magerko, B.; J. Laird; M. Assanie; A. Kerfoot; and D. Stokes. 2004. “AI Characters and Directors for Interactive Computer Games,” in *Proceedings of the 2004 Innovative Applications of Artificial Intelligence Conference*, ed. by A. Press.

Nareyek, A.. 2000. “Intelligent Agents for Computer Games,” in *Computers and Games, Second International Conference, CG 2000. LNCS 2063.*, 414–422.

Nareyek, A.. 2004. “Artificial Intelligence in Computer Games - State of the Art and Future Directions,” *ACM Queue*, 1(10), 58–65.

Nebel, B.; and J. Koehler. 1993. “Plan modification versus plan generation: A complexity-theoretic perspective,” in *Proceedings of of the 13th International Joint Conference on Artificial Intelligence (IJCAI)*, 1436–1441.

Niederberger, C.; and M. Gross. 2003. “Hierarchical and Heterogenous Reactive Agents for Real-Time Applications,” in *Proceedings of the Eurographics 2003, Computer Graphics Forum, Conference Issue*, ed. by P. Brunet, and D. Fellner, 323–331. Blackwell Publishing, UK.

Ponsen, M.; and P. Spronck. 2004. “Improving Adaptive Game AI with Evolutionary Learning.,” in *Proceedings of Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004)*, ed. by S. N. Quasim Mehdi, Norman Gough, and D. Al-Dabass, 389–396.

Spronck, P.; I. Sprinkhuizen-Kuyper; and E. Postma. 2004. “Enhancing the Performance of Dynamic Scripting in Computer Games,” in *Entertainment Computing - ICEC 2004*, ed. by M. Rauterberg, no. 3166 in Lecture Notes in Computer Science, 296–307. Springer-Verlag.

Tozour, P.. 2002. “The Perils of AI Scripting,” in *AI Game Programming Wisdom*, ed. by S. Rabin, 541–547. Charles River Media.

Chapitre 8

Teams of cognitive agents with leader: how to let them some autonomy.

Damien Devigne, Philippe Mathieu et Jean-Christophe Routier
Laboratoire d'Informatique Fondamentale de Lille
Cité Scientifique 59655 Villeneuve d'Ascq Cedex
{devigne,mathieu,routier}@lifl.fr

Abstract.[1]

Most often situated multi-agent simulations, of which platform-games are an example, uses reactive agents. This approach has limitations as soon as complex behaviours are desired. For these reasons we propose an approach using cognitive agents. They have knowledge, objectives and are able to build plans in order to achieve their goals and then execute them.

In this paper we particularly address the problem of teams of cognitive agents. We chose to build teams directed by a leader. One major problem is the building of the team plan and in particular one difficulty is to find the means in order to let autonomy to the team members. This can be done if the leader builds abstract plans. We present in this article a solution to this problem.

8.1 Introduction

Our work aims at producing agent-based simulations where the agents, situated in a geographical environment, behave "rationally". An application of such a work can be simulation platforms of which computer games are an instance, movies is another like "*The Lord of the Rings*" and the MASSIVE application illustrate it[2].

The main characteristics that we consider for these simulations are: first, the environment is defined by a geography, then the notions of positions or coordinates have a meaning, this is

[1] Article publié dans *Proceedings of IEEE Symposium on Computational Intelligence Games. CIG'05. pp. 256-262. Colchester. 2005.*

[2] see http://www.massivesoftware.com/news.html

a critical feature since relative positions must be considered before executing an action and consequently moves must be performed; second, the world is dynamic (agents can appear or disappear), and then heavily non monotonic (ie. values, once known, can change); third, agents are different: they can perform and suffer actions that are different from one to the other; and last, the agents are embodied: they are situated in the environment, and have a partial perception of it, from this it follows that the agent's knowledge is incomplete, moreover because of the non monotonic nature of the environment, this knowledge can be wrong.

According to J. Laird, computer games constitute the "killer application" for human-level AI (Laird and van Lent 2000). The characters involved in video games, like FPS or role-playing games, have indeed to be perceived as autonomous entities with increasing realistic behaviours. They have to be *convincing*, thus their behaviour must comply with the rational expectations of their partner or opponent human players. They also need to adapt to new situations, acquire additional abilities throughout the game, etc. In addition, team strategies are also often useful. Some research has been done concerning agents and games (Nareyek 2000), and most of them concern reactive agents (Nareyek 1998).

But reactive agents, while effective in several cases, offer limited behaviours. Indeed their behaviours are "short term directed" and not "goal oriented". Their ability to perform some tasks depends on the immediate surroundings and is not the result of wilful acts. In current commercial games, too often the character's behaviours are reactive ones, coded using scripts based on trigger/action sets. This approach has limitations (Tozour 2002). First the obtained behaviours are rather limited and it is difficult to get deliberate group behaviours unless they hard-coded them. This leads to a second major problem: the software design concern. It appears to be very difficult to reuse parts of AI from one game to another: scripts are too much tied to game design.

Our proposition aims at offering cognitive, driven by goals, proactive agents. To use cognitive agents allows to obtain more abstract reasoning. From one simulation/game to another the cognitive behavioural engine stays the same even if the context changes, and the behavioural components, the *interactions*, can be at least partially reused. Our approach uses declarative knowledge and thus favours the separation between the game logic and code. This promotes reusability and should ease the development.

In a first part we describe how we design simulated worlds or game environments: the geography, the laws that rule the world (the interactions) and the cognitive agents and their behavioural engine. Then we discuss teams and present our proposition to obtain team plans.

8.2 Simulations with cognitive agents

We define a simulation as follows:

$$\textit{Simulation} = E \times I \times A$$

where E is the topologic *environment*, I is the set of *interactions* that rule the simulated world and A is the set of *situated agents* involved in the simulation. In the following subsections we will quickly define these three points.

8.2.1 Environment

The environment describes the geography of the simulated "world". We represent the environment by a graph where nodes are *places* and vertices denote *path* from one place to another

(see Table 8.1). A place is an elementary geographical area. The granularity of a place depends on the simulation, the only constraint is that inside a place there is no restriction neither for moves, nor for perception (restrictions due to the other agents, like collision problems, are exempt). A place can represent a room, a town or any other part of the environment, and inside a place the position of an agent can be handled discretely or continuously depending on needs.

Environment	=	{*place*,path**}
path	=	(*place*$_{origin}$, *place*$_{dest}$,*condition)*

Table 8.1: Definition of environment

A path denotes an oriented transition between two places. It is defined by the places that it links, and a condition that must be satisfied if an agent wants to use this path (usually most of the conditions are simply *true*). The paths are oriented and the condition to go from some place a to a place b is not necessarily the same than the one to go from b to a. This formalism allows to describe, for example, that a door between two rooms must be opened if we want to go from one room to the other, or that an agent must be able to swim to cross a river between two fields. In this last case, our approach allows, depending on needs, to choose to represent or not the river with a place. It depends on whether ot not the crossing of the river has a meaning in the simulation (see figure 8.1).

Figure 8.1: Left: river is modelled. Paths are: (a,r,*"agent can swim"*), (r,a,*true*), (b,r,*"agent can swim"*), (r,b,*true*) Right: river is not modelled. The paths are: (a,b,*"agent can swim"*),(b,a,*"agent can swim"*).

8.2.2 Interactions

Knowledge representation is based on the notion of what we call "*interactions*"(Mathieu, Picault, and Routier 2003). Since the objective is to achieve simulations (like games are), it is necessary to represent the laws that rule the simulated world and to allow the agents to manipulate these as knowledge elements. We introduce our interactions in this goal.

Interactions are the backbone of our simulation model. They are at the basis of the knowledge representation in the simulation. Some agents (the actors) can perform interactions and others (possibly the same) can suffer from them (the targets).

An interaction is defined by a name and three parts:

- a *condition*, it tests the current context of execution of the interaction and consists mainly of tests on values of target or actor properties.
- a *guard*, it checks general conditions for the interaction applicability, typically it defines that to be fired an interaction requires that the distance between the target and the actor must be

less than some given value.
The guard is separated from the condition since it corresponds to the knowledge due to the geographically situated feature of the simulations. In a non situated context, one would have only the condition and action parts. The guard will be at the origin of the moves in the plan, moves that are indeed specific to situated problems. We do not express explicitly in an interaction that the agent has a move to do in order to fire it, we want the agent to plan it when required.

- an *action*, it describes the consequence of the interaction, it can be a change in the state of the actor and/or of the target (ie. a change of the value of a property), and/or the activation of an environment action (like the creation of an agent).

By example, to *open* an object (door, chest, window, etc.) makes it changing from *closed* state to *opened* one. The nature of the target is of no importance here (insofar as it can suffer *open*), this knowledge can then be represented in a "universal" way by the interaction (see below). In this sense, interactions are declarative knowledge: they describe an action and not how to solve/use it. Let us remark that there is no mention of moves to be performed to fire the interaction, the knowledge due to the situated property is mentioned in the guard. An interaction is a piece of *abstract* knowledge where the situated point of view is taken into account in the guard.

$$open: \begin{cases} \texttt{condition} & = & \text{“}\texttt{target.opened} = \texttt{false}\text{”} \\ \texttt{guard} & = & \text{“}\texttt{distance}(\texttt{actor}, \texttt{target}) < 1\text{”} \\ \texttt{action} & = & \text{“}\texttt{target.opened} = \texttt{true}\text{”} \end{cases}$$

One advantage in using such interactions is that this approach favours a good software engineering design. Since interactions are not tied to a particular agent nor to a simulated world, they can be reused from one to another. This is clearly the case with the above *open* interaction. Reusability is of course an important concern in software engineering design and in particular in AI game design where it is reputed to be not applied although wished.

To increase the generic nature of our interactions we propose a way to specialize them. This is not the object of this paper to detail it but let us say that the aim is to keep the declarative and abstract nature while taking into account the fact that to solve a given abstract action can require different conditions. To solve that we use something like inheritance of interactions. Using an example should help to present it shortly: again consider the *open* interaction, we said that it can be applied to different types of targets and used in a plan such that "*to fetch an apple in the next room I must* open *this door*". Now let us assume that this door is a lockable one (and is indeed locked). From an abstract point of view the plan is still valid, but the *open* interaction must be understood as "make the door change from *closed* to *opened* state *when it is unlocked*". The problem is then how the same abstract plan (open the door to fetch the apple) can receive different solutions (just "open" or "unlock and then open") depending on the target (whether it is lockable or not). Our solution is to allow to specialize interactions by adding extra conditions, thus you create another *open* interaction that "inherits" the previous one and to which you add the condition `target.isLocked=true`. This is this version that is given as *can-suffer* interaction to the lockable agents.

8.2.3 Agents

Our agents are embodied agents that are situated in their environment, they are influenced by it and more precisely by where they are in it.

We distinguish two kinds of agents: inanimate and animate agents (not to mistake them with mobile/non-mobile agents). Both are defined by properties and can suffer interactions but the latter can also perform interactions and have a behavioural engine (see Table 8.2). The animate agents are cognitive and proactive agents. They are responsible for the dynamics of the simulations. The interactions that an agent can perform correspond to its abilities.

Agents	=	*Animate* | *Inanimate*
Inanimate	=	{*Properties**, can-suffer}
Properties	=	*(name, value)*
can-suffer	=	*Interaction-name**
Animate	=	*Inanimate* $\cup$ can-perform
		$\cup$ *brain* $\cup$ *goals*
can-perform	=	*Interaction-name**
brain	=	*planning engine* $\cup$ *memory*
goals	=	*interaction-goal* | *condition-goal*

Table 8.2: Agent's definition

The cognitive agents The structure of the animate agent's "mind" is presented in Figure 8.2. Agents have a memory that can be seen like a "degraded environment". This one represents the knowledge base for all the information gathered by the agent concerning the environment: the topology of the environment, the other agents (their position and state). This information is used by the planning engine to determine the action that the agent must try to execute in the environment to fulfil its goal. A perception module is used to pick up information in the environment and to update the memory, this perception is local. Updates are performed by a separate module that has an influence on the planning engine in order to adapt the currently computed plan to the new perceived situation. This last module is a kind of "short term memory". From this it results that the knowledge of an agent is not complete, then an agent may have to search for unknown information, and can be wrong, but the agent is supposed to behave with respect to its knowledge. This is due to the dynamic and non monotonic nature of the environment. Knowing that its environment is non monotonic must be taken into account by the agent.

Goals Animate agents have goals. The satisfaction of these goals leads the agents to behave according to a computed plan. There exist two kind of goals. First, the *interaction-goals*, they correspond to an (inter)action that the agent has to execute. The target of this interaction can be less or more precisely given: from a named agent to any agent that can be the target of the interaction, as shown in the next table:

goal	type of target
`eat(apple_12)`	a given named apple
`eat(an apple)`	any apple
`eat(*)`	any eatable (ie. "who can-suffer eat") agent

Second, the *premiss-goals* (or *condition-goals*), they correspond to a condition that the agent wants to become true. For example:

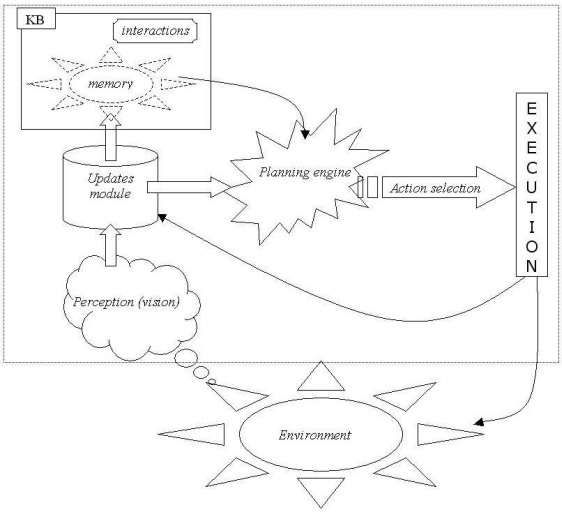

Figure 8.2: Different elements of agent's mind.

`actor.energy > 100`:
"having his energy being greater than 100"

Planning engine In order to achieve their behaviour the animate agents have an engine that computes plans in order to satisfy the goals given to them. The plan is produced by a backward chaining on the *can-perform* interactions of the agents. The plan building depends on the information stated in the memory (ie. the beliefs base) of the agent. In a rather classical way, the plan can be viewed as a tree (see Figure 8.3). The nodes are made of the different goals and subgoals encountered during the resolution. Some are *interaction-goals*, others are *premiss-goals*. Thus the tree is an alternation of condition and interaction nodes and corresponds to an AND-OR tree. AND-nodes correspond to condition-nodes (for condition-goals) and OR-nodes to interaction-nodes (for interaction-goals).

The condition and interaction nodes are classical case. The sons of a condition node are interaction-nodes built from the interactions whose action part offers a way to satisfy the condition (or help to satisfy it). The interaction-node's sons are built from the conditions that can be found in the condition and guard parts of the interaction: from these, condition-nodes are built. This is classical in backward chaining.

We want to underline a point that introduces differences in comparison with planning in non situated context. Indeed, since we consider embodied agents situated in a geographical environment and since we want to simulate their behaviour in such an environment, agents must perform moves. Typically an agent must move next to a target to interact with it. It is here that the guards that we have introduced in our interactions play their roles. To be allowed to fire the action part of the interaction, the agent must satisfy the conditions and the guards. But guards, since they imply moves that are a crucial side-effect in situated simulations, require specific consideration. We have discussed this problem in (Devigne, Mathieu, and Routier 2004) where we show in which ways the situated context has an influence on "planning while executing". Indeed, considering only conditions and actions leads to *abstract plans* that are valid independently of any situated

context: "to open a door only requires that it is closed and makes it opened". However, simulations in situated environment implies that in the *execution plan* moves must be done, and then "*to open a door*" requires also the actor being next to the door as expressed in the guard of the *open* interaction. This guard must be considered while building the plan. The backward chaining on the guard produces the moves and these moves can require specific planning in order to be achieved. The conditions that must be satisfied in order to be able to perform a move are the conditions that exist between the places in the computed path. Then the same *abstract plan* can lead to several *execution plans* each depending on the execution context where the agent is situated.

Let us just add that the plan is not rebuild at each step but partial replanning is done according to the new perceived information given by the updates module.

A small example To illustrate the different points described in the previous paragraph we will consider a very small and simple example. We consider a world with two places/rooms separated by a door d, the path between these two rooms has the condition "*d.locked=false*". Four interactions define the laws: *unlock*, *take*, *move*, *push* (see Table 8.3). In the world are an animate agent a that can perform these 4 interactions, and three inanimate agents, the door d that can suffer *open*, a key k that can suffer *take* (and can be used to unlock d) and a button b that can suffer *push*. The goal of the agent is to push on b. The figure 8.3 presents the planning in two different situations.

$$
\begin{array}{rl}
\textit{unlock:} & \left\{ \begin{array}{rcl} \textit{condition} & = & \text{“}\texttt{target.locked} = \texttt{true}\text{”} \\ \textit{guard} & = & \text{“}\texttt{distance}(\texttt{actor}, \texttt{target}) < 1\text{”} \\ \textit{action} & = & \text{“}\texttt{target.locked} = \texttt{false}\text{”} \end{array} \right. \\
\textit{take:} & \left\{ \begin{array}{rcl} \textit{condition} & = & \texttt{true} \\ \textit{guard} & = & \text{“}\texttt{distance}(\texttt{actor}, \texttt{target}) < 1\text{”} \\ \textit{action} & = & \text{“}\texttt{actor.own}(\texttt{target}) = \texttt{true}\text{”} \end{array} \right. \\
\textit{push:} & \left\{ \begin{array}{rcl} \textit{condition} & = & \texttt{true} \\ \textit{guard} & = & \text{“}\texttt{distance}(\texttt{actor}, \texttt{target}) < 1\text{”} \\ \textit{action} & = & \text{“}\texttt{target.pushed} = \texttt{true}\text{”} \end{array} \right. \\
\textit{move:} & \left\{ \begin{array}{rcl} \textit{condition} & = & \texttt{conditionsfoundinpath} \\ \textit{guard} & = & \\ \textit{action} & = & \text{“}\texttt{distance}(\texttt{actor}, \texttt{target}) < 1\text{”} \end{array} \right.
\end{array}
$$

Table 8.3: Definitions of the interactions (adapted - but without distortion - to shorten the example)

8.3 Teams

In the previous section we have briefly described our approach to model the simulated world and the agents. In particular we present how we obtain individual agent behaviour that allows agents to achieve tasks. However individual behaviours are not enough, sometimes tasks must be done by groups, or teams, of agents. In computer games the need of teams is important: teams of fighters in FSP games, groups of units in strategy games, teams of characters in role-player games, etc.

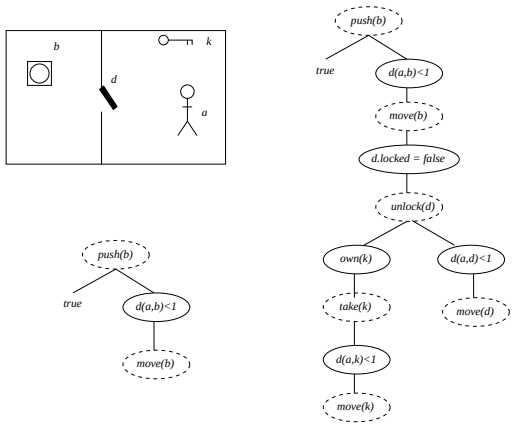

Figure 8.3: An animate agent a is situated in an environment where are also 3 inanimate agents, a door d, a button b and a key k. The goal of a is to push b. a must build a plan to achieve it. A plan can be drawn as a tree, nodes due interaction-goals are drawn with dashed lines and nodes due to condition-goals with solid lines. Depending on the execution context, different plans can be obtained. Left is the tree obtained when d is not locked and right is the case where d is locked. a must adapt its plan to the context.

Several situations can require the use of teams. First, getting a number of agents to do a job can speed up its achievement, this corresponds to task parallelization when several agents have the same abilities and perform similar tasks simultaneously, one agent could have done it alone but it would have taken more time. Second, in some cases, one agent is not sufficient to perform a task, and several must cooperate *simultaneously* to do it, by example this is the case when a heavy load must be carried and two or more agents are required in order to lift it, of course they must do this simultaneously. Third, complex tasks require a lot of different abilities and it is not often the case that one agent alone has all of them, then several "specialist agents" that together gather these abilities must cooperate to achieve the task. Of course, these three cases can mix. In this paper we mainly address this last case.

8.3.1 Teams of cognitive agents with leader

Making teams of agents work has been the subject of several approaches. Emergence is a solution to obtain a team behaviour (Capera, Georgé, Gleizes, and Glize 2003). But in this case we think that the notion of team work is only "apparent" since the team behaviour is a collateral effect of the sum of the individual behaviours and is not deliberate. We mean that the agents are not conscious that they work in a team and no team strategy is explicitly planned.

Our objective is to make our cognitive agents work in a team and being aware of it. To perform this we make some choices in this paper:

1. the team is assumed to be already created, that means that we are not concerned here with the problem of recruiting an able agent or constituting the team before doing the job.

2. the team is directed by a leader which is an agent that plays a particular role in the team. It is known at the beginning.

3. the leader is in charge of the team strategy and coordination, therefore our work does not consider the cases where agents negotiates to cooperate and to find an agreement on a plan. For example, plan merging (Kambhampati, Mali, and Srivastava 1998, Alami, Fleury, Herrb, Ingrand, and Qutub 1997) is not our interest here.

Having a leader that builds the team plan can be seen as a restriction since this implies that control is partially centralized. However one can see by oneself that this is often the case in real life: firemen in a squadron or workers in a building site obey to the orders of their leader. The point is that in such teams, even if the leader gives orders, team members still have their autonomy. They must behave according to the leader plan but have to use their knowledge and abilities to achieve these orders.

Indeed, an important feature is the granularity of the leader orders. A site foreman does not order a bricklayer "*take this red brick, bring it there and put it on the foundation, then take this second one, bring it there and put it next to the first one, then take this third one...*". His order is simply "*build this brick wall here*". How the wall is built is the responsibility and the competence field of the bricklayer. As we see here, the leader gives rather high level orders and is not concerned with details. Moreover these orders can be abstract insofar as they are not necessarily tied to a particular situated context: the foreman can use the plan of construction in his office to show the bricklayer the walls to be built, this one is in charge to do it according to the site constraints and situation. The worker is autonomous once the order has been given, probably he only must report when he succeeds or even informs his leader when a problem he cannot solve occurs.

Therefore, the leader is in charge to build the plan that solves the team goal but this plan does not describe the solution in full details.

In the following (see paragraph 8.3.3) we propose a solution to reproduce this behaviour: the leader has the knowledge about its team members abilities, it builds an abstract plan to solve the team goal and it distributes orders to the members. The members autonomously resolve their tasks and report to the leader.

8.3.2 Description of a team

Describing a team simply as a group of agents is not sufficient. Our proposition consists in describing the team structure independently of any concrete agent and then to instantiate it with the members.

Since we are interested in teams that gather several complementary specialists, we design the team structure in term of roles. A role corresponds to a set of abilities (ie. interactions) required to play it (see Table 8.4). We add a cardinality to each role, this allows to precise when several agents of the same type are required in a team. This information is for example useful to handle dynamic reorganization of the team, but we will no more use it in the following of this paper.

To instantiate a team consists in selecting existing able agents and to attribute them some role in the team. Of course to be able to play a role an agent must have all the interactions that describe it in its *can-perform* property. Then a team is given by a team-structure and a mapping from the role in the structure and the agents members of the team.

The knowledge concerning the team is given to the team leader. In this paper we are not interesting in how the leader recruits its team-mates.

TeamStructure	=	*(Role, Cardinality)**
Role	=	*Interaction**
Cardinality	=	*Natural..Natural*
Team	=	*TeamStructure* $\times$ *(Role,Agent)**

Table 8.4: Definition of team

8.3.3 Our proposition

Giving autonomy to the team members is an important point. First, as said before, this is more realistic and simulates what happens in real life. Second, this avoids to obtain stupid behaviours, in particular because we consider situated agents in dynamic environments like game worlds are. One must not forget that, as we say earlier, agent's knowledge, and by way of consequence the leader's knowledge, is not necessarily correct. Then, in the case where the leader builds the plan in every detail and gives very precise orders to the team members, those having no right to modify them, it is more than probable that members will be confronted with unexpected situations and will not be able (nor authorized actually) to handle them. As an example, such situations can be due to objects that are not where they are supposed to be. Then a precise order like "*go to a given precise location and take the brick*" can not be solved by a non autonomous agent if the brick has been moved. This is not the case with the more abstract order "*take the brick*" given to an autonomous agent that can decide and plan how to find the brick according to the environment it is confronted with. As a third advantage this provides an easy way to consider team of teams, we will discuss this later in the paper.

As it has been described earlier our agents are cognitive ones and are able to build plan according to their knowledge. Insofar as individual and team plans are of the same nature, there is no reason that the individual planning strategy could not be applied to team planning.

So, the problem that arises is: *How to adapt the individual planning to team work in order to let some team members autonomy*. Here follows our proposition.

As we have seen the plan can be viewed as a tree where root is the goal and leaves are actions to be executed in order to solve the goal. In a team plan, leaves are then the orders that the leader gives to the members. To be able to build this plan, the leader must have some knowledge concerning the members abilities, that is about their *can-perform* interactions.

Let us consider that the leader knows all the *can-perform* interactions of the members. Then exactly like in the individual planning, he could build a plan to solve the goal. But in this case he would build a full plan and team members will no more have any planning autonomy! So the solution is to not let the leader plans "until the end", but to limit the tree depth. But this can not be done arbitrarily. There is no reason to decide that the leader unfolds the tree until it has a depth of 3 rather than 5 or 10. The appropriate depth will depend on the goal and the members abilities, it will be different at every time. In order to compute the depth's limit the leader would have to compute the full plan before to cut it at a relevant depth. It is a nonsense to compute the full plan, then to forget it and ask the members to re-build it!

Therefore this approach is not correct. We must not forget that we want the plan built by the leader to be abstract. And then the leader does not need to have explicitly all the knowledge about the *can-perform* of its team members. Actually, the leader only has to know what high-level tasks the members are able to do. It even does not need to know by itself how to perform these tasks, like a foreman does not necessarily have to know how to build the wall, it suffices that he

knows that the bricklayer is able to do it. Indeed, the leader has to know what things must be done but not necessarily how to do them.

To achieve this we propose to hide some knowledge to the leader: the guards and conditions in the *can-perform* of the members are hidden to the leader. Thus the leader can know which of the member's interactions can be used in its plan since, knowing their action parts, it can use them during its backward chaining. However, since the leader knows no condition (nor guard) for these, it considers they are satisfied and stops the chaining. Then these interactions become necessarily leaves of the leader tree plan and can be distributed to the members as goals. Those members, having full knowledge, are able to make the appropriate plan.

Let us take an example. We have one agent named a_0 who can perform some interaction I_0 (see Table 8.5). Two other agents, named a_1 and a_2 can respectively perform interactions $\{I_1, I_3, I_4\}$ and $\{I_2, I_5\}$ (see Table 8.6).

a_0 will be the leader of the team. The team structure is made of two roles r_1 and r_2 that are defined respectively by interactions $\{I_1, I_3\}$ and $\{I_2\}$. This implies that the "high level" tasks the leader can ask to the members are to satisfy p_1, p_2 or p_3. The conditions and guards of these interactions are hidden to a_0 (see Table 8.5).

	leader.can-perform	team.can-perform		
name	I_0	I_1	I_2	I_3
conditions	p_1, p_2	–	–	–
guard	true	–	–	–
actions	p_0	p_1	p_2	p_3

Table 8.5: Leader's knowledge, conditions and guards are hidden for the interactions of the team roles. They are considered as *true*.

As we can see, agents a_1 and a_2 can play roles r_1 and r_2 respectively, they are chosen as team members.

name	I_1	I_2	I_3	I_4	I_5
conditions	p_3, p_4	p_5	true	true	true
guard	G_1	G_2	G_3	G_4	G_5
actions	p_1	p_2	p_3	p_4	p_5

Table 8.6: Definitions of *can-perform* interactions of team members.

Now, the team is given the goal p_0. The leader, a_0 builds the plan for it. According to its knowledge, the backward chaining leads to use interaction I_1[3] that requires p_1 and p_2 to be solved. These lead respectively to the use of I_1 and I_2 (in this plan I_3 does not interfere from the leader point of view). For the leader, I_1 and I_2 have their conditions and guards satisfied (since they are hidden and seem to have none), then it stops its planning here (see Figure 8.4).

Now the leader can distribute the tasks to its team members according to their role in the team, indeed the leader is not able to perform the task itself since I_1 and I_2 are not in its *can-perform*. Then it gives p_1 to a_1 as goal and p_2 to a_2. Now each agent autonomously solves its goal

[3] In the following we assume condition p_i not to be satisfied.

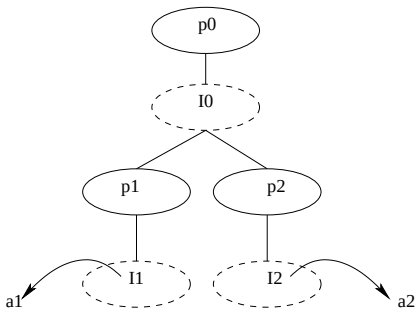

Figure 8.4: Tree representing the plan built by the leader. It is developed until the abstract team's interactions are reached, goals can then be given to the team members.

according to its knowledge and builds the appropriate plan (see Figure 8.5). Then it can determine which action to fire to solve its goal and can inform its leader when it succeeds or if it fails.

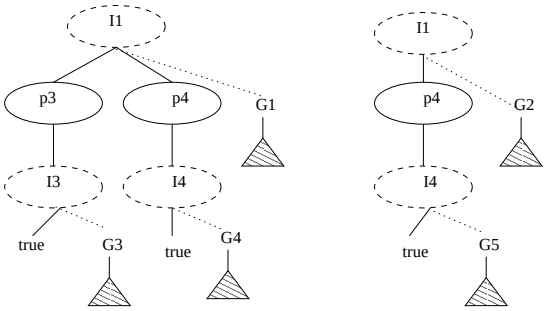

Figure 8.5: Trees representing the individual plans built autonomously by the 2 team members (guard's plan are not detailed, they depend on the context).

The importance of the autonomy of the agent is increased while considering the influence of the surrounding environment and specially with the situated aspect. This is expressed within the guards whose resolution has not been detailed in above trees. Indeed, as we have seen since guards express that the actor must be near the target in order to fire the interaction, they requires planning to be solved. But this is highly context dependant. And it would have been particularly irrelevant for the leader to plan it instead of the members.

8.3.4 Team of teams

With our above described approach it is easy to build "team of teams", or team built as a hierarchy of agents, where one "big leader" orders to subleaders that order to and so on, until "basic members". In fact it applies immediately to such cases without change!

Indeed, each level of the hierarchy corresponds to a level of decision with its type of orders. The higher in the hierarchy an agent is, the more high-level or abstract its orders are. For example, a works foreman can order a bricklayer leader to have walls built and the carpenter leader to have windows installed. Each of them orders to his team-mates to do the appropriate work.

With our approach the leader's planning stops with the abstract orders that can be given to team members and become a goal for them. If a team member agent is a leader itself, it applies the same procedure: starting from the goal that he has received, he builds a plan that stops when the abstract knowledge of its team (ie. interactions where conditions are hidden) is used. Then it can give orders to its team members. As we note nothing particular has to be done in order to consider hierarchies of teams: building the teams with their knowledge suffices.

8.4 Conclusion

In this paper we propose a mean to handle teams of cognitive situated agents directed by a leader. The presented solution let the team members some autonomy in the way they contribute to the team plan achievement. Indeed, the leader uses abstract knowledge on team's abilities to build an abstract plan and then distributes high-level orders to its team-mates.

Several problems have not been addressed in this paper and require complementary works, among them let us cite:

- how is the team built, that is how the leader recruits its team-mates?
- how to dynamically reorganize a team when an agent leave it (the cardinality information that are just mentioned in the paper can be used here)?
- how the information are exchanged inside the team?
- how to proceed in the case of teams with no leader?

They are, with others, the subjects of future works.

Bibliographie

Alami, R.; S. Fleury; M. Herrb; F. Ingrand; and S. Qutub. 1997. "Operating a Large Fleet of Mobile Robots using the Plan-Merging Paradigm," in *Proceedings of IEEE ICRA 97*.

Capera, D.; J.-P. Georgé; M.-P. Gleizes; and P. Glize. 2003. "The AMAS theory for complex problem solving based onself-organizing cooperative agents," in *Proceedings of WETICE'03*, 383–388.

Devigne, D.; P. Mathieu; and J.-C. Routier. 2004. "Planning for Spatially Situated Agents in Simulations," in *Proceedings of the 2004 IEEE/WIC/ACM International Joint Conference IAT'04*.

Kambhampati, S.; A. Mali; and B. Srivastava. 1998. "Hybrid planning for partially hierarchical domains," in *Proceedings of the 15th national/10th conference on Artificial Intelligence/Innovative applications of artificial intelligence*, 882–888. AAAI.

Laird, J. E.; and M. van Lent. 2000. "Human-level AI's Killer Application: Interactive Computer Games," .

Mathieu, P.; S. Picault; and J.-C. Routier. 2003. "Simulation de comportements pour agents rationnels situés," in *Actes des Secondes Journées Francophones MFI'03*, 277–282.

Nareyek, A.. 1998. “Specification and development of reactive systems,” in *1998 AIPS Workshop*, 7–14, Menlo Park California. AAAI Press.

Nareyek, A.. 2000. “Intelligent Agents for Computer Games,” in *Computers and Games, Second International Conference, CG 2000. LNCS 2063.*, 414–422.

Tozour, P.. 2002. “The Perils of AI Scripting,” in *AI Game Programming Wisdom*, 541–547.

Partie III

Annexe

Mes activités d'enseignant-chercheur

Jean-Christophe Routier né le 4 juin 1968, marié, 3 enfants

Diplômes

– DEA Intelligence Artificielle et Applications, Université de Caen, 1990, mention bien.

– Ingénieur de l'ENSI de Caen, option Intelligence Artificielle, 1990.

– Docteur en Informatique de l'Université des Sciences et Technologies de Lille, thèse dirigée au LIFL par Philippe Devienne et Patrick Lebègue et soutenue le 1er février 1994, mention très honorable.
Terminaison, calculabilité, pouvoir calculatoire d'une clause de Horn binaire

Situation Actuelle

Maître de conférences depuis septembre 1994 à l'UFR d'IEEA de l'Université des Sciences et Technologies de Lille.

Membre de l'équipe SMAC (Systèmes Multi-Agents et Coopération) du Laboratoire d'Informatique Fondamentale de Lille – CNRS UMR 8022

Dans cette section je vais brièvement présenter mon activité dans les différentes facettes de mon travail d'enseignant-chercheur. J'essaierai ainsi de démontrer mon implication dans les trois domaines :

- *Enseignement* où je décrirai les enseignements effectués ainsi que la partie gestion de formation
- *Administration* où je citerai les charges collectives auquel j'ai pris ou prends part
- *Recherche* où, dans la mesure où les premières parties de ce document constituent une synthèse de mes activités depuis mon doctorat, je citerai mon activité en terme d'encadrement, participation à des projets et animation.

Je finirai par une énumération de mes publications rangées par catégories.

Enseignement

Enseignements devant étudiants

Je ne vais pas énumérer ici tous les enseignements que j'ai effectués depuis ma nomination en 1994, ce serait fastidieux pour le lecteur. Pour les résumer, je peux dire que j'ai effectué mes interventions (cours magistraux, TD ou TP) dans toutes les années du cycle universitaire : Bac+1 à Bac+5. Les formations où je suis intervenu sont également diversifiées, professionnalisées ou non. Il s'agit principalement de : DEUG MIAS, IUP GMI (seconde et troisième année), DESS (surtout IAGL), Licence Informatique, DEA d'Informatique. Actuellement mes enseignements se concentrent sur les semestres S4-S5-S6 de la licence mention Informatique de Lille 1. L'essentiel de mes enseignements peut se ranger dans les rubriques "Programmation/Conception de Logiciel" (de l'initiation à l'approfondissement) et "Intelligence Artificielle". J'ai eu également l'occasion au cours de ces années d'encadrer, aux niveaux Bac+4 et Bac+5, de nombreux projets d'étudiants ainsi que des stages en entreprise.

Je peux également parler rapidement de quelques enseignements dont je suis (ou étais) responsable et que j'ai créés.
D'abord la mise en place au début de ma nomination des enseignements d'initiation à la programmation lors de la création du DEUG MIAS. Jusque là, il n'y avait pas d'enseignement d'Informatique en première année de DEUG à Lille 1. Ce travail a débouché sur la rédaction d'un ouvrage (cf. *Publications*).
Ensuite les cours de "Programmation Orientée objet" et "Conception Orientée Objet" que j'ai créés en 2001-2002 lorsque le paradigme de programmation objet a été choisi comme approche principale de la programmation en licence d'Informatique. Dans la suite de ces cours, je suis également responsable de l'Unité d'Enseignement "Projet Logiciel" de la licence mention Informatique.
J'ai également créé cette année avec Erci Wegrzynowski l'Unité d'Enseignement ELFE de la licence S5 dont l'objectif est de présenter les paradigmes logique et fonctionnel.

Administration de l'enseignement

Au cours de ces années j'ai eu l'occasion de participer à différents niveaux à la gestion de formations professionnalisées ou non.

– Président de jury d'une section de DEUG MIAS, 1995-1996.

– Responsable du DESS Intelligence Artificielle et Génie Logiciel (IAGL) de 1997 à 2002.

  - présidence de jury,
  - sélection/recrutement des étudiants,
  - gestion, coordination des enseignements et de l'équipe pédagogique,
  - gestion des projets et stages,
  - rédaction de la maquette d'habilitation de la formation (1998 et 2001)

– Membre du Groupe de Proposition de Formation "Mathématique-Informatique-Physique-Mécanique" à l'occasion de la mise en place du LMD à Lille 1 (2003-2004).

– Co-directeur des études des semestres S4-S5-S6 la mention Informatique de la licence Sciences et Technologies A de Lille 1 depuis 2004.

- organisation des enseignements, gestion de l'équipe pédagogique,
- relation avec les étudiants,
- rédaction de la maquette d'habilitation de la formation (2004 et 2005),
- gestion/animation des Comissions Pédagogiques Paritaires (CPP).

Diffusion de la connaissance

– rédaction et diffusion de documents de cours en ligne,

– rédaction d'un livre sur l'initiation à la programmation (cf. *Publications*).

Charges Collectives

Il s'agit ici des charges collectives non directement liées à l'enseignement car celles-ci ont déjà été présentées dans la rubrique précédente.

– Membre suppléant de la CSE 27ème section de l'USTL de 2000 à 2003.

– Membre titulaire de la CSE 27ème section de l'USTL depuis 2003

– Membre du conseil de l'UFR d'IEEA de l'USTL depuis 2002.

– Responsable de la gestion des services pour l'Informatique depuis 2003.

  Dans ce cadre j'ai développé un logiciel permettant un gestion plus rationnelle des services via une "interface web". Ce logiciel est également déployé dans la partie EEA de l'UFR cette année.

– Responsable et coordinateur pour l'Informatique des dossiers et opérations "Contenu Numérique en Ligne" de l'Université de Lille 1, en 2004 et 2005.
  Un des résultats de cette action est le portail de présentation de la mention Informatique de la licence `http://www.fil.univ-lille1.fr/Licence`.

Recherche

Les premières parties de ce mémoire constituant une synthèse de mes travaux de recherche depuis l'obtention de mon doctorat, je ne vais pas les résumer à nouveau ici. Je citerai simplement les éléments importants en terme d'animation, d'encadrement et publications.

Contrat

– Collaboration avec la société Cryo Interactive à travers un projet du programme PRIAMM (Programme pour l'Innovation dans l'Audiovisuel et le Multimédia), janvier 2000 à juin 2001.

Organisation de conférences

– Membre du comité d'organisation des Journées Francophones de l'Intelligence Artificielle Distribuée et des Systèmes Multi-Agents (JFIADSMA) 2002. Lille, 27-30 octobre 2002. Chargé de la gestion des inscriptions.

– Membre du comité d'organisation des Journées sur les Modèles Formels de l'Interaction (MFI) 2003. Lille, 20-22 mai 2003. Chargé de la gestion des inscriptions.

Participations à des groupes et projets de recherche

– membre des groupes ASA et MFI du GDR I3.

– membre d'AgentLink à travers l'équipe SMAC,

– participation aux CPER (Contrat de Plan Etat-Région) successifs dans lesquels l'équipe était ou est actuellement impliquée : d'abord le projet *Ganymède* puis ses successeurs *COLORS* ("Composants Logiciels Réutilisables et Sûrs"), *NIPO* ("Nouvelles Interactions Personnes-Organisations") et *Formasciences* sur les nouveaux usages, et actuellement *MIAOU* ("Modèles d'Interaction et Architectures Orientées Usages") et *MOSAIQUES* ("MOdèles et InfraStructures pour Applications ubIQUitairES").

Encadrement

Mémoire de DEA/Master Recherche

– Yann Secq, "*Notion de service et d'échange de services dans les systèmes multi-agents*". DEA d'Informatique de l'Université de Lille 1. Co-dirigé avec le Professeur Philippe Mathieu. 1999.

– Patrick Tessier, "*Planification et exécution dans un environnement non monotone*". DEA d'Informatique de l'Université de Lille 1. Co-dirigé avec Professeur Philippe Mathieu. 2002.

– Damien Devigne, "*Simulation de comportements pour agents rationnels situés et étude du GraphPlan*". DEA d'Informatique de l'Université de Lille 1. Co-dirigé avec le Professeur Philippe Mathieu. 2003.

– Julien Acroute, "*Apport réactif aux comportements cognitifs de la plateforme de simulation CoCoA*". Master Recherche mention Informatique de l'Université de Lille 1. Co-dirigé avec le Professeur Philippe Mathieu. 2005.

Thèse

– Yann Secq, "*RIO : Rôles, Interactions et Organisations, une méthodologie pour les systèmes multi-agents ouverts*". Co-dirigée avec le Professeur Philippe Mathieu. Soutenue le 2 décembre 2003.

– Damien Devigne, "*Modélisation de comportement d'équipes en environnement Multi-Agents situés*". Co-dirigée avec le Professeur Philippe Mathieu. 2003-en cours.

Publications

Livre

– "*Débuter la Programmation avec* SCHEME" Jean-Christophe Routier et Éric Wegrzynowski. International Thomson Publishing France. 349 pages. 1ère édition 1997. seconde édition mai 2003.

Revues internationales

– "*Smallest Horn clause programs*" Philippe Devienne, Patrick Lebègue, Anne Parrain, Jean-Christophe Routier et Jörg Würtz (DFKI, Saarbrücken), in Journal of Logic Programming, Vol. 27(3), pp. 227-267. 1996.

– "*Using agents to build a distributed calculus framework*" Philippe Mathieu, Jean-Christophe Routier et Yann Secq, in The Interdisciplinary Journal of Artificial Intelligence and the Simulation of Behaviour vol. 1 number 2, pp. 197-208. June 2002.

Revues francophones

– "*Une contribution du multi-agent aux applications de travail coopératif*" Philippe Mathieu et Jean-Christophe Routier in TSI Hermès Science Publication. Réseaux et Systèmes Répartis. Calculateurs Parallèles. Volume 13. Numéro Spécial Télé-Applications, pp. 207-226. 2001.

– "*Les agents intelligents*" Philippe Mathieu, Sébastien Picault et Jean-Christophe Routier. "Pour la Science" Numéro 332. Juin 2005, pp. 44-52.

Conférences internationales avec actes

– "*The halting problem of one binary Horn clause is undecidable*" Philippe Devienne, Patrick Lebègue et Jean-Christophe Routier. oin P. Enjalbert, A. Finkel and K.W. Wagner (Eds), Proceedings of STACS'93, Würzburg. LNCS 665, pp. 48–57. 1993.

– "*The emptiness problem of one binary Horn clause is undecidable*" Philippe Devienne, Patrick Lebègue et Jean-Christophe Routier. in Dale Miller (Ed), proceedings of 1993 International Symposium on Logic Porgramming, ILPS'93, Vancouver, pp. 250–265. 1994.

– "*One binary Horn clause is enough*" Philippe Devienne, Patrick Lebègue, Jean-Christophe Routier et Jörg Würtz (DFKI, Saarbrücken). in P. Enjalbert, E.W. Mayr and K.W. Wagner (Eds), proceedings of STACS'94, Caen. LNCS 775, pp. 21–32. 1994.

– "*Dynamic Skill Learning: A Support to Agent Evolution*" Philippe Mathieu, Jean-Christophe Routier et Yann Secq, in Proceedings of the AISB'01 Symposium on Adaptive Agents and Multi-Agent Systems, York, ISBN 1 902956 17 0, pp. 25–32. 2001.

– "*A Multi-Agent Approach to Co-operative Work*" Philippe Mathieu et Jean-Christophe Routier, in C. Kolski and J. Vanderdonckt (Eds), Proceedings of the Computer Aided Design of User Interfaces, CADUI'02, Valenciennes, pp. 367-380. 2002.

– "*RAGE: An agent framework for easy distributed computing*" Philippe Mathieu, Jean-Christophe Routier et Yann Secq, in Proceedings of the AISB'02 Symposium on Artificial Intelligence and Grid Computing, ISBN 1 902956 24 8, London, pp. 20-24. 2002.

– "*Principles for dynamic multi-agent organizations*" Philippe Mathieu, Jean-Christophe Routier et Yann Secq , in K. Kuwabara and J. Lee (Eds), proceedings of the Fifth Pacific Rim International Workshop on Multi-Agents, PRICAI2002-PRIMA2002, Tokyo, Spinger-Verlag LNAI vol. 2413, pp. 109-122. 2002.

– "RIO *: Roles, Interactions and Organizations*" Philippe Mathieu, Jean-Christophe Routier et Yann Secq. in V. Marik and J. Müller and M. Pechoucek (Eds), the Proceedings of the 3rd International/Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003, Prague. LNAI 2691, pp. 147-157. June 2003.

– "*Bridging the gap between semantic and pragmatic*" Philippe Mathieu, Jean-Christophe Routier et Yann Secq. in H.R. Arabnia (Ed), the Proceedings of The 2003 International Conference on Information and Knowledge Engineering, IKE'03, vol. I, Las Vegas, pp. 308-314. June 2003.

– "*Runnable specifications of interactions protocols for open multi-agent systems*" Philippe Mathieu, Jean-Christophe Routier et Yann Secq. in Nazli Goharian (Ed), the Proceedings of the 2003 International Conference on Information and Knowledge Engineering, IKE'03, vol. II, Las Vegas, pp. 431-437. June 2003.

– "*Towards a Pragmatic Methodology for Open Multi-agent Systems*" Philippe Mathieu, Jean-Christophe Routier et Yann Secq in N. Zhong, Z. W. Raś, S. Tsumoto and Einoshin Suzuki (eds) , the "Foundations of Intelligent Systems". 14th International Symposium on Methodologies for Intelligent Systems, ISMIS 2003, Maebashi (Japon). Springer-Verlag, LNAI 2871, pp. 206-210. 2003.

– "Planning for Spatially Situated Agents" Damien Devigne, Philippe Mathieu et Jean-Christophe Routier. in the Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'04), Beijing (Chine). IEEE Press, pp. 385-388. 2004.

– "Team of cognitive agents with a leader : how to let them acquire autonomy" Damien Devigne, Philippe Mathieu et Jean-Christophe Routier. in G. Kendall and S. Lucas (Eds), the Proceedings of IEEE Symposium on Computational Intelligence and Games, CIG 2005, York, pp. 256-262. 2005.

– "*Interaction-Based Approach For Game Agents*" Damien Devigne, Philippe Mathieu et Jean-Christophe Routier. in Y. Merkuryev and R. Zobel and E. Kerckhoffs (Eds), the Proceedings of ECMS/SCS/IEEE 19th European Conference on Modelling and Simulation. ECMS 2005, Riga, pp. 705-714. 2005.

Conférences francophones avec actes

– "*RIO: Rôles, Interactions et Organisations*" Philippe Mathieu, Jean-Christophe Routier et Yann Secq, in A. Herzig and B. Chaib-draa and P. Mathieu (Eds), actes des Secondes Journées Francophones sur les Modèles Formels de l'Interaction, MFI'03, Lille, pp. 179-188. Mai 2003.

– "*Simulation de comportements pour agents rationnels situés*" Philippe Mathieu, Sébastien Picault et Jean-Christophe Routier. in A. Herzig and B. Chaib-draa and P. Mathieu (Eds), actes de Modèles Formels des Interactions, MFI'03, Lille, pp. 277-282. mai 2003.

– "*Gestion d'équipes et autonomie des agents*" Damien Devigne, Philippe Mathieu et Jean-Christophe Routier, Calais, actes des JFSMA2005. 2005.

Autres publications

– "*Weighted Sytems of Equations revisited*" Philippe Devienne, Patrick Lebègue et Jean-Christophe Routier. in actes du Workshop on Static Aanalysis WSA'92, Bordeaux, pp. 163–173. 1992.

– "*The halting problem of one binary Horn clause is undecidable*" Philippe Devienne, Patrick Lebègue et Jean-Christophe Routier. "Workshop on termination" à l'occasion de JICSLP'92, Washington. 1992.

– "Tutoriel de l'API MAGIQUE", Jean-Christophe Routier `http://www.lifl.fr/SMAC` rubrique `MAGIQUE`. 2000.

– "*Un modèle de simulation agent basé sur les interactions*" Philippe Mathieu, Jean-Christophe Routier et Pascal Urro (Sté Cryo Interactive), in actes de Modèles Formels des Interactions, Toulouse, MFI'01. (poster) 2001.

– "*Dynamic Organization of Multi-Agent Systems*" Philippe Mathieu, Jean-Christophe Routier et Yann Secq, poster, in proceeedings of AAMAS'02, Bologne, pp. 451-452. (poster) july 2002.

– "*Ubiquitous Computing : vanishing the notion of application*" Philippe Mathieu, Jean-Christophe Routier et Yann Secq, poster, in proceedings of the Workshop UbiAgents'02 on Ubiquitous Agents on embedded, wearable, and mobile devices, Bologne, (poster) july 2002.

– "*Towards a pragmatic use of ontologies in multi-agent platforms*" Philippe Mathieu, Jean-Christophe Routier et Yann Secq in the "Ontology and Multi-Agent Systems Design" session (OMASD'03), in the Seventh International Conference on Knowledge-Based Intelligent Information & Engineering Systems, KES'03, Oxford, 2003.

– "*Gestion simple d'équipes d'agents cognitifs*" Damien Devigne, Philippe Mathieu et Jean-Christophe Routier. in the Proceedings of Majestic'2004, Calais. 2004.