Méthodologie de développement de systèmes multi-agents adaptatifs et conception de logiciels à fonctionnalité émergente

Gauthier Picard

THÈSE

présentée devant

l'Université Paul Sabatier de Toulouse III

U.F.R. MATHÉMATIQUES, INFORMATIQUE ET GESTION

pour obtenir le titre de

DOCTEUR DE L'UNIVERSITÉ PAUL SABATIER

Mention INFORMATIQUE

par

GAUTHIER PICARD

École doctorale : Informatique et Télécommunication
Laboratoire d'accueil : Institut de Recherche en Informatique de Toulouse
Équipe d'accueil : Systèmes Multi-Agents Coopératifs

Méthodologie de développement de systèmes multi-agents adaptatifs et conception de logiciels à fonctionnalité émergente

---

soutenue le 10 décembre 2004 devant la commission d'examen :

Claudette CAYROL – *professeur à l'université de Toulouse III* (présidente du jury)

Philippe MATHIEU – *professeur à l'université de Lille I* (rapporteur)
Michel OCCELLO – *professeur à l'université de Grenoble II* (rapporteur)

Carole BERNON – *maître de conférences à l'université de Toulouse III* (examinatrice)
Marie-Pierre GLEIZES – *maître de conférences à l'université de Toulouse III* (directrice de thèse)

Thierry MILLAN – *maître de conférences à l'université de Toulouse III* (invité)

Gauthier Picard

MÉTHODOLOGIE DE DÉVELOPPEMENT DE SYSTÈMES MULTI-AGENTS ADAPTATIFS ET CONCEPTION DE LOGICIELS À FONCTIONNALITÉ ÉMERGENTE

Directrice de thèse :
Marie-Pierre Gleizes, maître de conférences
Université Paul Sabatier

Résumé

Les environnements des applications d'aujourd'hui sont de plus en plus complexes et dynamiques, compte tenu du grand nombre et de la diversité des acteurs en jeu. Les fonctions de tels systèmes deviennent alors de plus en plus difficiles à définir, et leur spécification est souvent incomplète, même si les composantes restent pleinement identifiables et spécifiables. Si de nouvelles méthodes de conception et de modélisation ne sont pas mises au point, la gestion des projets deviendra de plus en plus contraignante, longue et coûteuse.

Nous proposons d'utiliser les systèmes multi-agents adaptatifs par auto-organisation coopérative pour palier ces problèmes de conception. La fonctionnalité de ces systèmes est une résultante émergeant des interactions coopératives entre agents. Toutefois, le développement de tels systèmes est resté confidentiel et réduit à un groupe autour de ses créateurs directs. Certes de nombreuses applications ont été conçues grâce à ces systèmes, mais jamais par des novices, non experts du domaine. Pour répondre à ce manque de visibilité et d'ouverture, le projet ADELFE - pour Atelier de DEveloppement de Logiciels à Fonctionnalité Emergente - propose de développer une méthode de développement d'applications reposant sur ces principes et définie en trois points : un processus, des notations et des outils. Le processus d'ADELFE est basé sur le Rational Unified Process et y ajoute des activités spécifiques à l'ingénierie orientée agent. Les notations sont une extension des notations UML et A-UML. Des outils ont été développés ou étendus afin de prendre en charge à la fois les notations, grâce à OpenTool, et le processus, grâce à un outil d'aide au suivi appelé Adelfe-Toolkit.

La pertinence de cette méthodologie a été mise à l'épreuve au cours de développements d'applications diverses. Nous présentons ici les résultats obtenus pour un problème de résolution dynamique d'emploi du temps, ETTO (pour Emergent Time Tabling Organisation), et pour un problème de transport multi-robot de ressources.

Institut de Recherche en Informatique de Toulouse - UMR 5505
Université Paul Sabatier, 118 route de Narbonne. 31062 TOULOUSE cedex 4

Gauthier Picard

METHODOLOGY FOR DEVELOPING ADAPTIVE MULTI-AGENT SYSTEMS AND DESIGNING APPLICATIONS WITH EMERGENT FUNCTIONALITY

Supervisor :
Marie-Pierre Gleizes, maître de conférences
Université Paul Sabatier

Abstract

Environments within which applications are embedded are growing in complexity and dynamicity, considering the large number and the diversity of the takeholders. Functions of such systems become more and more difficult to define, and their specifications are often incomplete, even if their components are easily identifiable and specifiable. Without new design and modeling methods, managing such project will become too constraining, long and costly to cope with.

We propose to use cooperative self-organising adaptive multi-agent systems (AMAS) to tackle these design problems. The functionality of such systems emerges from cooperative interactions between agents. Nevertheless, developing using AMAS is still an ad-hoc process and reduced to a small group of users. Several applications have been designed by using AMAS, but it has never been executed by novices and non AMAS experts. To answer to this lack of visibility and openess, the ADELFE project – for *Atelier de DEveloppement de Logiciels à Fonctionnalité Emergente* or Toolkit for developing applications with emergent functionalities – proposes to develop a methodology based on these emergence oriented principles. This methodology is defined in three points : process, notations and tools. The ADELFE process is based on the Rational Unified Process and extends or adds some agent specific activities. Notations are extensions of UML and A-UML. Tools have been developed or extended in to support notations, with OpenTool, and following the process is eased by using AdelfeToolkit.

The relevance of this approach has been confronted to the development of experimental applications. Some results from a dynamic timetable solver, ETTO, and from a multi-robot transportation task are presented and analysed.

Institut de Recherche en Informatique de Toulouse - UMR 5505
Université Paul Sabatier, 118 route de Narbonne. 31062 TOULOUSE cedex 4

Je dédie cette thèse
à Francisco, mon grand-père

JE tiens à remercier...

... Marie-Pierre GLEIZES, maître de conférences à l'Université Paul Sabatier, Toulouse III, ma directrice de thèse, pour m'avoir dirigé, guidé, soutenu et encouragé tout au long de mon cursus scientifique, de la maîtrise à la thèse. Sans ton optimisme et ton dynamisme, je ne serais certainement pas là où j'en suis. Tu as su me faire aller aux quatre coins du monde pour apprendre et découvrir, ce qui n'était pas chose aisée. Bien sûr, ce que tu m'as apporté ne s'arrête pas au seul domaine scientifique ou pédagogique et je t'en serai longtemps redevable.

... Pierre GLIZE, ingénieur CNRS à l'IRIT, responsable de l'équipe SMAC, pour m'avoir fait découvrir tous les aspects de la coopération, tant au niveau artificiel qu'humain. Nos discussions – encore incessantes – ne font que confirmer mon intérêt pour les systèmes émergents en particulier et la science en général. Il y a encore du chemin a parcourir et j'espère continuer dans cette voie pour ces échanges enrichissants et ton *feedback* toujours positif – pour le petit système apprenant que je suis.

... Claudette CAYROL, professeur à l'Université Paul Sabatier, Toulouse III, directrice de la formation doctorale RCFR, pour avoir accepté de présider avec ouverture et intérêt mon jury de thèse.

... Philippe MATHIEU, professeur à l'Université de Lille I, et Michel OCCELLO, professeur à l'Université de Grenoble II, pour avoir accepté d'évaluer ma thèse de manière honnête et intègre et donc de lire consciencieusement ce mémoire. Vos nombreuses questions, pré ou post-soutenance, ont été la preuve de l'intérêt porté à mon travail, ce en quoi je vous suis extrêmement reconnaissant.

... Carole BERNON, maître de conférences à l'Université Paul Sabatier, Toulouse III, pour les excellentes relations de travail que nous entretenons tous les jours et pour ta participation à mon jury de thèse – qui fut une première pour toi, je t'en remercie grandement.

... Thierry MILLAN, maître de conférences à l'Université Paul Sabatier, Toulouse III, pour ta collaboration lors du projet ADELFE et ta perpétuelle bonne humeur.

... L'équipe SMAC de l'IRIT, mon équipe d'accueil depuis cinq ans, pour m'avoir accepté en tant que nouvel *agent* – coopératif, je l'espère – et m'avoir offert des conditions de travail exceptionnelles. Merci en particulier aux permanents, André MACHONIN, Christine RÉGIS et Valérie CAMPS, ainsi qu'aux autres thésards, Jean-Pierre GEORGÉ, Davy CAPERA, Jean-Pierre MANO et Kévin OTTENS.

... Mes amis toulousains Jean-Pierre, Davy, David F., David C., Vincent, et Emilie (non, pas Emile !), pour m'avoir rendu le travail plus agréable encore et m'avoir soutenu – certes inconsciemment – lors de périodes difficiles que je partage rarement.

... Mes amis clermontois Stéphane, Thierry, Frédéric, Didier, Sylvain, Jérôme et tous les autres, pour avoir gardé le contact avec moi toutes ces années et pour être la source de mes meilleurs souvenirs d'enfance.

... Mes amis *incarniens* – et toulousains – Gilles, Alegz, Law, Didier, Martin, David, Boud, les *Ludopathes* et les autres pour m'offrir du rêve et des larmes... Merci tout particulièrement à Julien, mon complément créateur et artistique, compagnon de douleurs et de joies, grâce à qui ma flamme intérieure du changement est constamment ravivée.

... Ma famille, Violette, Alain, Adrien, Aurélie et tous les autres pour leur soutien tout au long de mon existence. Merci aussi à Renée et Michel pour leur accueil et leur gentillesse.

... Celles qui partagent ma vie, à savoir Yoko, pour ces ronrons apaisants et son *zen* sans égal, mais surtout ma petite Sophie, qui sait écouter le non dit et me comprendre comme personne.

Table des matières

Introduction **1**

I Contexte et état de l'art : conception de systèmes émergents **5**

1 Les systèmes multi-agents adaptatifs **7**

1.1 Emergence dans les systèmes artificiels 8

1.1.1 La notion d'émergence 8

1.1.1.1 Les systèmes émergents 9

1.1.1.2 Une définition de l'émergence pour les systèmes artificiels . 10

1.1.2 Une technique pour l'émergence : l'auto-organisation 11

1.1.2.1 Principe d'auto-organisation 11

1.1.2.2 L'auto-organisation dans la nature 12

1.1.3 Le calcul émergent 15

1.1.4 Adaptation et apprentissage par auto-organisation 15

1.2 Les systèmes multi-agents 16

1.2.1 La notion d'agent 16

1.2.2 La notion de système multi-agent 17

1.2.3 La notion d'environnement 17

1.2.4 L'adaptation par auto-organisation 18

1.3 La coopération comme critère d'auto-organisation : la théorie des AMAS . . . 19

1.3.1 La notion de coopération 19

1.3.2 L'adéquation fonctionnelle et la coopération 20

1.3.3 Conséquences méthodologiques 21

1.3.4 Définition de la coopération 21

1.3.5 Discussion 22

1.4 De la théorie à la pratique 23

1.4.1 ANTS : simulation de fourmilière 23
1.4.2 ABROSE : courtage en ligne 25
1.4.3 STAFF : prévision de crues 25
1.4.4 Autres applications des AMAS 26
1.4.5 Analyse et constat 27

2 Ingénierie orientée agent **29**
2.1 L'influence du paradigme objet 31
2.1.1 L'agent, évolution de l'objet ? 31
2.1.2 Le processus 33
2.1.3 Les notations et langages de spécification orientés objet 33
2.1.3.1 UML et ses extensions 33
2.1.3.2 Autres notations graphiques 35
2.2 L'apport du naturel 35
2.2.1 Les animaux sociaux 36
2.2.2 Les réseaux neuronaux de Kohonen 37
2.2.3 Les algorithmes génétiques 38
2.2.4 Modèles sociaux et organisationnels 40
2.3 Alternatives et autres sources d'influences 41
2.3.1 Ingénierie des connaissances 41
2.3.2 Ingénierie des besoins 42
2.3.3 Les plates-formes multi-agents et de simulation 44
2.4 Méthodologie et comparaison de méthodes 44
2.4.1 Les cadres de comparaisons existants et problématique 44
2.4.1.1 Les concepts clés et propriétés 45
2.4.1.2 Les notations et langages de modélisation 45
2.4.1.3 Le processus 46
2.4.1.4 La pragmatique 46
2.4.2 Analyse de méthodes existantes 46
2.4.2.1 AAII 47
2.4.2.2 Aalaadin 49
2.4.2.3 Cassiopée 51
2.4.2.4 DESIRE 53
2.4.2.5 Gaia 56
2.4.2.6 MaSE 58

2.4.2.7 MASSIVE . . . 61
2.4.2.8 MESSAGE . . . 64
2.4.2.9 PASSI . . . 67
2.4.2.10 Prometheus . . . 70
2.4.2.11 TROPOS . . . 72
2.4.2.12 Voyelles . . . 74
2.5 Analyse de la comparaison et expression des besoins . . . 77
2.5.1 Lecture suivant l'axe des méthodes . . . 77
2.5.2 Lecture suivant l'axe des notions méthodologiques orientées agents . 79
2.5.3 Bilan et expression des besoins pour une méthode des AMAS . . . 79

II ADELFE : méthode de développement de systèmes multi-agents adaptatifs 83

3 Le processus d'ADELFE 85
3.1 Le SPEM . . . 86
3.1.1 Le SPEM, UML et le MOF . . . 86
3.1.2 Définition de processus avec le SPEM . . . 86
3.1.3 Les guides . . . 88
3.1.4 Pourquoi modéliser le processus d'ADELFE en SPEM ? . . . 88
3.2 Survol du processus . . . 89
3.2.1 Spécificités d'ADELFE . . . 90
3.2.2 Un exemple tutoriel : ETTO . . . 91
3.3 Les besoins préliminaires (WD_1) . . . 92
3.3.1 Définir les besoins utilisateur (A_1) . . . 92
3.3.2 Valider les besoins utilisateur (A_2) . . . 94
3.3.3 Définir les besoins consensuels (A_3) . . . 94
3.3.4 Établir la liste des mots-clés (A_4) . . . 95
3.3.5 Extraire les limites et les contraintes (A_5) . . . 95
3.4 Les besoins finals (WD_2) . . . 95
3.4.1 Caractériser l'environnement (A_6) . . . 96
3.4.1.1 Déterminer les entités (S_1) . . . 96
3.4.1.2 Définir le contexte (S_2) . . . 98
3.4.1.3 Caractériser l'environnement (S_3) . . . 99
3.4.2 Déterminer les cas d'utilisation (A_7) . . . 100

3.4.2.1 Inventorier les cas d'utilisation (S_1) . . . 100
3.4.2.2 Identifier les échecs à la coopération (S_2) . . . 101
3.4.2.3 Élaborer les diagrammes de séquences (S_3) . . . 102
3.4.3 Élaborer les prototypes d'interfaces utilisateur (A_8) . . . 103
3.4.4 Valider les prototypes d'interfaces utilisateur (A_9) . . . 103
3.5 L'analyse (WD_3) . . . 103
3.5.1 Analyser le domaine (A_{10}) . . . 104
3.5.1.1 Identifier les classes (S_1) . . . 104
3.5.1.2 Étudier les relations entre classes (S_2) . . . 104
3.5.1.3 Construire les diagrammes de classes préliminaires (S_3) . . . 106
3.5.2 Vérifier l'adéquation aux AMAS (A_{11}) . . . 106
3.5.2.1 Vérifier l'adéquation aux AMAS au niveau global (S_1) . . . 107
3.5.2.2 Vérifier l'adéquation aux AMAS au niveau local (S_2) . . . 108
3.5.3 Identifier les agents (A_{12}) . . . 109
3.5.3.1 Étudier les entités relativement au domaine (S_1) . . . 110
3.5.3.2 Identifier les entités potentiellement coopératives (S_2) . . . 111
3.5.3.3 Déterminer les agents (S_3) . . . 112
3.5.4 Étudier les interactions entre entités (A_{13}) . . . 112
3.6 La conception (WD_4) . . . 113
3.6.1 Un modèle d'agent coopératif . . . 114
3.6.1.1 Les différents modules . . . 114
3.6.1.2 Fonctionnement interne . . . 117
3.6.2 Étudier l'architecture détaillée et le modèle multi-agent (A_{14}) . . . 117
3.6.2.1 Déterminer les paquetages (S_1) . . . 118
3.6.2.2 Déterminer les classes (S_2) . . . 118
3.6.2.3 Utiliser les patrons de conception (S_3) . . . 119
3.6.2.4 Élaborer les diagrammes de composants et de classes (S_4) . . . 119
3.6.3 Étudier les langages d'interaction (A_{15}) . . . 119
3.6.4 Concevoir les agents (A_{16}) . . . 120
3.6.4.1 Remplir les modules des agents . . . 121
3.6.4.2 Définir l'attitude coopérative et les situations non coopératives (SNC) . . . 123
3.6.5 Prototypage rapide (A_{17}) . . . 127
3.6.6 Compléter les diagrammes de conception (A_{18}) . . . 128
3.7 Les travaux suivants . . . 128

3.7.1 Développement et Implémentation 128
3.7.2 Tests et validation 129
3.8 Bilan et remarques sur le processus d'ADELFE 130

4 Les notations et outils utilisés dans ADELFE 133
4.1 Les stéréotypes d'ADELFE 134
4.1.1 UML est-il adéquat pour ADELFE ? 134
4.1.2 Comment la notation peut-elle être étendue ? 134
4.1.3 Description des stéréotypes d'ADELFE 135
4.1.3.1 Le stéréotype «cooperative agent» 135
4.1.3.2 Le stéréotype «characteristic» 136
4.1.3.3 Le stéréotype «perception» 137
4.1.3.4 Le stéréotype «action» 137
4.1.3.5 Le stéréotype «skill» 137
4.1.3.6 Le stéréotype «aptitude» 138
4.1.3.7 Le stéréotype «representation» 138
4.1.3.8 Le stéréotype «interaction» 139
4.1.3.9 Le stéréotype «cooperation» 139
4.2 Extensions d'A-UML dans ADELFE 140
4.2.1 Les protocoles A-UML dans ADELFE 141
4.2.1.1 Contraintes de spécification des protocoles 141
4.2.1.2 Utilisation des diagrammes de protocole 142
4.2.1.3 Diagrammes de séquence d'instances enrichis 144
4.2.2 Transformation en machine à états finis 144
4.2.2.1 Construction de machines à états finis à régions concurrentes 144
4.2.2.2 Jonctions A-UML et clauses IF 145
4.2.2.3 Remarques 147
4.3 Les outils associés à ADELFE 148
4.3.1 Prise en charge des notations avec OpenTool 148
4.3.1.1 Stéréotypage ADELFE et diagrammes de classes étendus . . 149
4.3.1.2 Diagrammes de protocole A-UML et transformation automatique en machines à états finis 150
4.3.1.3 Simulation pour prototypage rapide 150
4.3.1.4 Limites actuelles 151
4.3.2 Prise en charge du processus avec AdelfeToolkit 151

4.3.2.1 Utilisation du métamodèle SPEM . . . 151
4.3.2.2 Connexions avec d'autres outils . . . 152
4.3.2.3 Outil d'adéquation des AMAS . . . 152
4.3.2.4 Limites actuelles . . . 153
4.4 Bilan et remarques sur les notations et les outils d'ADELFE . . . 154

III Expérimentations et Analyses 155

5 Expérimentation d'ETTO 157
5.1 Les variantes du problème . . . 158
5.1.1 Variante 1 : résolution possible sans relâcher les contraintes . . . 158
5.1.2 Variante 2 : résolution possible en relâchant certaines contraintes . . . 158
5.1.3 Variante 3 : les contraintes peuvent varier en temps réel . . . 159
5.1.4 Variante 4 : ouverture du système . . . 159
5.2 Expérimentations et résultats . . . 159
5.2.1 Le comportement des agents . . . 160
5.2.2 Expérimentation de la variante 1 . . . 160
5.2.3 Expérimentation de la variante 2 . . . 162
5.2.4 Expérimentation des variantes 3 et 4 . . . 162
5.3 Discussion . . . 164

6 Un problème de transport de ressources en robotique collective 167
6.1 Transport collectif de ressources – travaux pré-conception . . . 168
6.1.1 Les besoins et le cahier des charges . . . 169
6.1.2 Analyse . . . 170
6.2 Conception des robots transporteurs . . . 171
6.2.1 Définition des différents modules . . . 171
6.2.2 Décision et choix des actions . . . 172
6.3 Étude des comportements coopératifs . . . 172
6.3.1 Gestion coopérative des blocages . . . 173
6.3.2 Anticipation des situations non coopératives . . . 173
6.4 Expérimentations et résultats . . . 175
6.4.1 Dispositif expérimental . . . 175
6.4.2 Comparaison réaction/anticipation . . . 175
6.4.3 Émergence de sens de circulation . . . 176

6.4.4 Question d'optimalité 177
6.4.5 Environnements difficiles 178
6.4.5.1 Environnement avec virages et impasses 178
6.4.5.2 Environnement avec fermeture aléatoire de couloirs 179
6.5 Discussion 179

7 Analyse et Perspectives **183**

Conclusion **195**

Bibliographie **199**

Liste des figures **211**

Liste des tables **215**

Introduction

Méthololologie de développement de systèmes multi-agents adaptatifs et conception de logiciels à fonctionnalité émergente. Derrière ce titre, se cache la volonté de faire découvrir aux non spécialistes des systèmes multi-agents, les enjeux et les possibilités de développement de systèmes adaptés aux besoins grandissants des utilisateurs de logiciels.

Motivations

Les environnements dans lesquels sont plongées les applications d'aujourd'hui deviennent de plus en plus complexes, dynamiques et imprévisibles compte tenu du grand nombre et de la diversité des acteurs en jeu. Les fonctions de tels systèmes deviennent par conséquent plus difficiles à définir, et leur spécification est souvent incomplète, même si les composantes restent pleinement identifiables et spécifiables. Par exemple, l'*Ubiquitous Computing*, ou informatique diffuse, propose d'utiliser la large diffusion des processeurs dans les appareils de la vie courante (comme les chaînes hi-fi, les téléphones, les systèmes domotiques, etc.) pour proposer des fonctionnalités avancées obtenues par la composition de leurs fonctions en réponse aux désirs des utilisateurs – comme *Ecouter un bon Coltrane avec une lumière tamisée de manière automatique tout en coupant le téléphone...* Seulement la fonctionnalité d'un tel système dépend énormément des utilisateurs, donc de l'environnement, et reste difficilement spécifiable en tant que système atomique (non décomposable) et rigide. Si de nouvelles méthodes de conception et de modélisation ne sont pas mises au point, la gestion de tels projets deviendra de plus en plus contraignante, longue et coûteuse.

De l'émergence aux systèmes multi-agents

Dans la nature, des phénomènes analogues à ceux rencontrés en conception des système artificiels apparaissent et sont bien connus des biologistes : les phénomènes *émergents*. Les systèmes dotés de propriétés émergentes, sont des systèmes dont les parties, de micro-niveau, sont facilement identifiables et modélisables, mais dont le comportement de macro-niveau obtenu par composition de toutes les actions des parties, est non déductible des théories de micro-niveau. Par exemple, les termites sont des entités simples dont les éthologues peuvent fournir des modèles comportementaux réalistes (de micro-niveau). Pourtant, rien dans ces modèles ne permet de déduire la capacité (de macro-niveau) pour plusieurs termites, de construire des arches et des ponts dans leur nid. Grâce à des interactions internes et à la pression de l'environnement, les systèmes émergents fournissent des fonctionnalités

supérieures à la simple somme des fonctions des parties en jeu.

Une technique possible pour obtenir des systèmes ayant de telles propriétés émergentes et d'adaptation à leur milieu est l'*auto-organisation*. Ce mécanisme consiste en la capacité, pour les parties du système à changer leurs interactions et leur place dans l'organisation afin de produire, globalement, une fonction plus adaptée à leur environnement. Ce mécanisme est tout à fait envisageable dans des systèmes artificiels, pour lesquels, par exemple, l'environnement est l'ensemble des utilisateurs auxquels le système devra fournir une fonction la plus adaptée et adéquate possible. L'auto-organisation nécessite néanmoins de doter les parties du système de capacités locales de décision et de critères de réorganisation – *quand changer la structure, et donc la fonction du système ?*

Concernant les capacités décisionnelles des parties d'un système auto-organisateur, le domaine *multi-agent* propose de décomposer les systèmes en entités autonomes capables de raisonnements : les *agents*. C'est un paradigme de modélisation adéquat pour construire des systèmes adaptatifs par auto-organisation du moment où l'on fournit un critère efficace de réorganisation. En cela, l'équipe SMAC de l'IRIT propose l'utilisation de la *coopération* entre les agents. Cette notion, bien qu'inspirée par des phénomènes sociaux ou biologiques, possède une définition générique pour les systèmes artificiels ayant pour but de réduire les interactions inutiles ou antinomiques entre les parties du système et entre le système et son environnement. Nous parlerons alors de systèmes multi-agents adaptatifs (ou *AMAS*) par auto-organisation coopérative.

Vers une méthode plus adaptée : ADELFE

Toutefois, la conception de systèmes auto-organisateurs par coopération est restée confidentielle et réduite à un groupe autour de ses créateurs directs. Certes de nombreuses applications ont été conçues grâce à ces systèmes, mais jamais par des novices, non experts du domaine. Pour répondre à ce manque de visibilité et d'ouverture, le projet ADELFE – pour ***A**telier de **DE**veloppement de **L**ogiciels à **F**onctionnalité **E**mergente*[1] – propose de développer une méthode de développement d'applications reposant sur ces principes.

La méthode ADELFE, qui constitue l'ossature de ce mémoire, est définie en trois points : un processus, des notations et des outils. Un processus consiste en la définition d'étapes et de travaux à accomplir afin de développer une application. Le processus d'ADELFE s'inscrit dans le *Rationnal Unified Process* (ou RUP), qui est un processus issu de l'ingénierie orientée objet, et propose des activités d'analyse et conception spécifiques à la problématique des AMAS, en considérant les agents comme des objets de programmation de niveau supérieur. Compte tenu des spécificités agents, et du cadre du RUP, les notations utilisées pour la modélisation sont des extensions de l'UML (ou *Unified Modeling Language*), qui est la notation orientée objet la plus diffusée, et de son dialecte orienté agent, A-UML. Enfin, afin de faciliter à la fois le suivi du processus et la manipulation des notations, plusieurs outils ont été développés ou étendus. AdelfeToolkit est une plate-forme de développement qui trace l'avancée d'un projet et facilite le suivi du processus en proposant des raccourcis vers des applications

[1] Projet RNTL ayant comme partenaires : Artal Technologies (`http://www.artal.fr`), l'IRIT (`http://www.irit.fr`), le L3I (`http://www-l3i.univ-lr.fr`) et TNI-Valiosys (`http://www.tni-valiosys.com`).

dédiées et en illustrant chacune des activités par un exemple, ETTO (pour *Emergent Time Tabling Organisation*), un système de gestion d'emploi du temps en temps réel. Les notations sont prise en charge par OpenTool[2], un logiciel de conception UML, qui a dû être étendu, pour l'occasion.

Structure du mémoire

Ce mémoire, structuré en quatre parties, présente les résultats des recherches effectuées pour la conception de systèmes émergents basés sur l'auto-organisation coopérative en général, et lors du projet ADELFE en particulier. Dans la première partie, le chapitre 1 présente le contexte d'étude, à savoir les systèmes émergents, les systèmes multi-agents et les AMAS. Ce chapitre a principalement pour but de clarifier nos intentions quant à la façon dont nous envisageons le fonctionnement des systèmes adaptatifs.

La seconde partie est dédiée à ADELFE. Notre souhait de proposer une méthode de développement pour de tels système implique l'étude et l'analyse des méthodes existantes dans le domaine multi-agent. Pour cela, le chapitre 2 présente un état de l'art de l'ingénierie orientée agent ainsi que l'expression de nos besoins pour une méthode des AMAS. Les deux chapitres suivants présentent les trois composantes d'ADELFE : le processus (chapitre 3), les notations et les outils (chapitre 4).

Afin de valider notre approche, ADELFE a été utilisée pour développer des applications de natures diverses – gestion d'emploi du temps, bio-informatique, inférence adaptative ou robotique collective – et nous présenterons deux applications dans la troisième partie. Le chapitre 5 présente des résultats obtenus grâce à un prototype d'ETTO afin de montrer la pertinence de l'approche par systèmes multi-agents adaptatifs pour résoudre des problèmes de contraintes dynamiques. Un autre exemple d'application de nature complètement différente, car ne faisant pas intervenir des acteurs humains, est celui de la simulation de collectifs de robots devant accomplir une tâche de transport de ressources dans un environnement contraint. Cet exemple est développé dans le chapitre 6.

Enfin, le chapitre 7 présente un bilan sur l'état actuel d'ADELFE et les perspectives en découlant, à plus ou moins long terme.

[2]OpenTool appartient à la société TNI-Valiosys.

Première partie

Contexte et état de l'art : conception de systèmes émergents

1 Les systèmes multi-agents adaptatifs

CONCEVOIR des systèmes de plus en plus complexes en terme de fonctionnalités, d'architectures, d'entrées/sorties ou bien d'adaptation nécessite la définition de nouveaux paradigmes systémiques. Ces besoins sont à l'origine des travaux sur les systèmes multi-agents, composés de plusieurs entités appelées agents, pour aller au-delà de la simple décomposition fonctionnelle ou conceptuelle en programmation classique. Ici, le concepteur délègue une partie du fonctionnement cognitif aux entités du système, qui ne sont plus des entités fonctionnelles simples, mais des agents autonomes, réactifs, proactifs ou sociaux.

Afin de rendre les systèmes plus adaptatifs à leur environnement, qui se veut plus complexe et plus ouvert, compte tenu des technologies grandissantes de l'information, de la diversité des sources et des types de données, une solution est de considérer des systèmes multi-agents adaptatifs auto-organisateurs capables de modifier leur structure et/ou organisation, et ainsi leur fonction, en réponse aux fluctuations environnementales. Ainsi, l'environnement du système remplit deux rôles primordiaux pour le système. Il est à la fois un but, dans le sens où le système doit atteindre une adéquation fonctionnelle pour être adapté, et un moyen de pression, car l'organisation du système changera en fonction de ses points d'entrées provenant de l'environnement.

Comment guider cette auto-organisation reste pourtant une problématique majeure. Dans ce chapitre, la notion de coopération est proposée comme critère local –au sein des agents– de réorganisation du système. Le principe de localité est important dans de tels systèmes qui se veulent une réponse aux problèmes complexes, comme l'informatique diffuse, dans lesquels une définition claire de la fonctionnalité du système est quasi-impossible. Seules les parties du système sont clairement identifiables et spécifiables. La fonction globale du système au macro-niveau est alors une conséquence –irréductible– de l'activité inter-agent de micro-niveau : on parlera alors de phénomènes émergents.

Ce chapitre développe les notions introduites ci-dessus en quatre points. Le paragraphe 1.1 établit le cadre systémique dans lequel s'inscrit ce travail : l'émergence et l'auto-organisation. Le paragraphe 1.2 affiche les systèmes multi-agents comme un paradigme pertinent pour l'exploitation de la notion d'émergence dans les systèmes artificiels. Le paragraphe 1.3 présente un cas particulier de systèmes multi-agents, les AMAS (*Adaptive Multi-Agent Systems*), dans lesquels l'auto-organisation est obtenue par coopération des agents et qui nous servira de cadre théorique tout au long de ce travail. Enfin, le paragraphe 1.4 présente quelques exemples d'application de la théorie des AMAS à des problèmes divers.

1.1 Emergence dans les systèmes artificiels

Comme le souligne Horn [2001] d'IBM, dans le but d'introduire le concept d'*autonomic computing*, sans nouvelles approches, les problèmes de conception de systèmes informatiques vont empirer. En effet, de nombreuses applications nécessitent le développement de systèmes caractérisés des spécifications incomplètes pour les raisons suivantes :

- le système doit évoluer dans un environnement dynamique ;
- le système est ouvert, i.e. dont la cardinalité est variable ;
- le système est complexe ;
- le système produit une fonction non-linéaire, i.e. dont une légère variation en entrée peut produire une grande variation en sortie du système ;
- le système comporte un grand nombre de composantes en interaction ;
- l'organisation interne du système est a priori inconnue ;
- le contrôle du système ne peut être centralisé.

Face à de telles difficultés, une possibilité est d'explorer de nouvelles voies théoriques que celles habituellement utilisées en informatique ou d'exploiter de nouvelles notions. Le monde naturel est alors source d'inspiration pour la conception de systèmes ou l'élaboration de nouvelles théories. Depuis quelques années, la notion d'**émergence** pour les systèmes artificiels s'est vue mise en exergue dans de nombreux domaines comme la thermodynamique, la physique ou bien la sociologie.

1.1.1 La notion d'émergence

Le terme d'émergence est rattaché au postulat suivant : *"Le tout est plus que la somme de ses parties."* Ce courant de pensée pré-socratique, ainsi que la notion d'auto-organisation (cf. paragraphe 1.1.2) remonte à l'antiquité grecque et ont été retrouvés dans les écrits de Thalès ou d'Anaximandre [Ali *et al.*, 1997].

Ces notions, ainsi que le "tout avant les parties" d'Aristote ou bien le "gestalt" de Goethe donneront naissance (vers 1920) au mouvement dit proto-émergentiste, appuyé par des philosophes tels que G.H. Lewes, C.D. Broad ou bien J.C. Smuts [Goldstein, 1999]. Ici, le phénomène d'émergence est considéré comme une boîte noire possédant des entrées et des sorties multiples de haut niveau (voir figure 1.1). Ce concept de boîte noire implique l'absence totale d'explication sur l'émergence, mais permet de l'identifier le cas échéant.

Bien sûr, l'absence d'explication ne put rester en l'état. Le néo-émergentisme, représenté par Holland ou Forrest notamment cherche à ouvrir cette boîte noire et à comprendre les processus internes aux phénomènes émergents (voir figure 1.1) [Holland, 1998; Forrest, 1991]. Ce courant est intimement lié à la théorie de la complexité et trouve des échos en thermodynamique, mathématiques du chaos ou en synergétique [Haken, 1978]. Il consiste principalement à élaborer des méthodes visant à expliquer et démystifier le phénomène d'émergence.

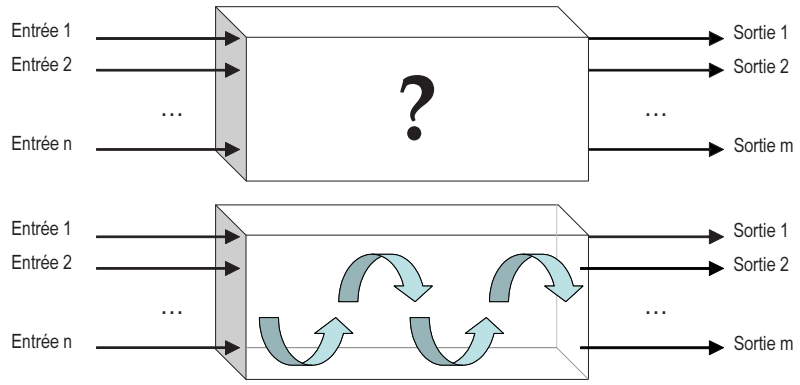

Figure 1.1 — La métaphore de la boîte noire : le proto-émergentisme (en haut) et le néo-émergentisme (en bas).

1.1.1.1 Les systèmes émergents

Actuellement, plusieurs définitions de l'émergence co-existent, chacune se focalisant sur une particularité plutôt que de donner une description précise :

L'*auto-organisation* : Goldstein définit l'émergence comme l'apparition de processus d'auto-organisation [Goldstein, 1999]. Nous verrons plus loin l'importance de cette notion d'auto-organisation dans notre travail en tant que technique pour construire des systèmes émergents[1] ;

L'*interdépendance des niveaux* : Langton se sert de cette notion pour définir l'émergence en termes de feedback entre macro-niveau et micro-niveau [Langton, 1990]. Là encore, les notions de feedback et de couplage environnement-système sont capitales dans une approche de conception de systèmes adaptatifs. Ceci implique la prise en compte de l'environnement comme moteur et acteur de l'adaptation et de l'évolution du système ;

L'*irréductibilité* des propriétés de macro-niveau aux propriétés de micro-niveau [Ali *et al.*, 1997]. Ceci implique l'impossibilité de déduire, à partir des propriétés de micro-niveau, les propriétés de macro-niveau. Inversement, et du point de vue méthodologique, vouloir construire un système par émergence, c.-à-d. spécifier les entités de micro-niveau en donnant les caractéristiques de macro-niveau, devient chose difficile ;

La *nouveauté* : sur laquelle se focalise Van de Vijver, implique la non prédictibilité des phénomènes de macro-niveau [Van de Vijver, 1997].

Malgré les disparités entre ces points de vue, quelques propriétés interdépendantes et communes permettent de caractériser un phénomène comme émergent ou non. Ainsi, un système sera dit émergent [Georgé, 2003, 2004] si :

- le phénomène est *ostensible*, c.-à-d. qu'il s'impose à l'observateur au macro-niveau ;
- le phénomène est *radicalement nouveau*, c.-à-d. qu'il est imprévisible à partir d'une connaissance du micro-niveau ;
- le phénomène est *cohérent* et *corrélé*, c.-à-d. qu'il a une identité propre mais liée aux parties de micro-niveau ;

[1] Par abus de langage, un système dit émergent est un système possédant une fonction globale émergente.

– le phénomène produit une *dynamique* particulière, c.-à-d. que le phénomène n'est pas pré-défini, s'auto-crée et s'auto-maintient.

Ces derniers traits permettent d'identifier un phénomène émergent pour vérifier si le système supportant le phénomène l'est, mais du point de vue méthodologique, ils ne constituent pas une base suffisante pour construire des systèmes. Par contre, plusieurs caractéristiques communes à ces systèmes peuvent être analysée afin de guider la conception d'applications reposant sur la notion d'émergence :

Les *attracteurs*. Contrairement aux systèmes primitifs où il n'y avait qu'un seul type d'attracteur menant à l'état final, les systèmes émergents possèdent différents types : le point fixe, le cycle limite et l'attracteur étrange. Ces attracteurs ne sont pas pré-donnés et ne dictent pas au système l'état final, mais plutôt les moyens de passer d'état en état.

L'*au-delà de l'équilibre*. Au lieu de se focaliser sur les points d'équilibre du système, on s'intéresse au voisinage de ces points, comme en théorie de la complexité. En de tels points, l'apparition de phénomènes non prévisibles explique le caractère inattendu de l'émergence [Prigogine et Nicolis, 1977].

L'*auto-organisation*. Plus qu'une manière de définir l'émergence, l'auto-organisation, qui correspond au comportement créatif, auto-généré et à la recherche d'adaptation d'un système complexe, est un moyen d'obtenir des phénomènes émergents.

La *non-linéarité*. La linéarité permet de tracer l'activité du niveau local pour interprétation au niveau global. Pour avoir un système émergent, il faut donc que les interactions entre composantes du système soient obtenues de manière non linéaire. Ceci signifie que toute partie du système doit pourvoir être influencée de manière indirecte par d'autres composantes du système. Ceci est le cas dans les systèmes à feedback négatif ou positif.

1.1.1.2 Une définition de l'émergence pour les systèmes artificiels

Les définitions et caractéristiques précédentes sont des notions d'ordre systémique, et ne sont par conséquent liées à aucune discipline particulière. Georgé a proposé une définition d'ordre pragmatique pour les systèmes artificiels [Georgé, 2004]. En effet, la conception de systèmes informatiques n'exploitait que rarement la notion d'émergence. Toutefois, Holland s'est appuyé sur certains paradigmes naturels pour obtenir des systèmes possédant des caractéristiques de systèmes émergents, comme les algorithmes génétiques ou les automates cellulaires [Holland, 1998].

La définition suivante, en trois points, propose d'identifier l'objet de l'émergence (ce qui émerge), la condition, et la méthode [Georgé, 2004] :

Objet	Le système artificiel a pour finalité de réaliser une fonction adéquate à ce que l'on attend du système. C'est cette fonction, pouvant évoluer au cours du temps, que nous voulons faire émerger.
Condition	Cette fonction est émergente si le codage du système ne dépend pas explicitement de la manière d'obtenir ou de la connaissance de cette fonction. Ce codage doit contenir des mécanismes permettant l'adaptation du système au cours de ses échanges avec l'environnement afin de tendre à tout instant vers la fonction adéquate.
Méthode	Pour changer de fonction, il suffit de changer l'organisation des composants du système. Ces mécanismes sont spécifiés par des règles régissant l'auto-organisation entre les composants et ne dépendant pas de la connaissance de la fonction collective.

Le premier point de cette définition sert à identifier sur quoi porte l'émergence. Ici, la fonction du système doit émerger. Cette définition pragmatique vise à écarter les problèmes d'émergence de formes ("patterns") comme dans les automates cellulaires. Le deuxième point souligne la caractéristique d'irréductibilité des systèmes émergents. En effet, si la fonction globale du système est explicitement connue par les parties du système, la théorie de description du macro-niveau est déductible des fonctions de micro-niveau. Enfin, le troisième point, d'ordre méthodologique, propose de voir l'auto-organisation comme moyen d'obtenir des phénomènes émergents.

1.1.2 Une technique pour l'émergence : l'auto-organisation

Afin d'obtenir des systèmes adaptatifs pour des environnements dynamiques ou complexes, la conception de systèmes à fonctionnalité émergente semble être une solution prometteuse. En effet, spécifier un modèle pré-défini de macro-niveau pour un système plongé dans un environnement dynamique, c'est fortement contraindre l'espace des possibilités. L'étude d'un système devient alors l'étude de ses sous-parties prises isolément [von Bertalanffy, 1993]. Considérant les notions et les éléments soulevés dans le paragraphe précédent, l'auto-organisation est un moyen technique d'obtenir des phénomènes émergents. Il n'est pas étonnant de rapprocher ces deux notions (émergence et auto-organisation) : elles sont souvent confondues car elles possèdent des caractéristiques communes comme la non-linéarité, ou "l'au-delà de l'équilibre" [Prigogine et Nicolis, 1977; Heylighen, 1999]. Ce paragraphe a pour but d'expliquer ce qu'est l'auto-organisation et de présenter quelques exemples naturels d'auto-organisation.

1.1.2.1 Principe d'auto-organisation

Considérons un système S réalisant une fonction f_S et plongé dans un environnement E. Ceci signifie que le système possède des entrées (percepts) provenant de l'extérieur ainsi que des sorties, ou moyens d'agir, vers l'environnement. S est composé de plusieurs parties P_i réalisant des sous-fonctions f_{P_i}. La fonction globale du système f_S est alors une composition

de ces sous-fonctions : $f_S = f_{P_1} \circ f_{P_2} \circ \ldots \circ f_{P_n}$. La définition de cette composition reste difficile à décrire et dépend du type de système et de la nature de sa décomposition : spatiale, fonctionnelle pure ou temporelle par exemple. Par contre, elle peut être définie en terme de liens d'interactions fonctionnelles entre les parties, comme le montre la figure 1.2.

Généralement, on ne peut garantir que l'opération $\circ$ soit commutative : nous considérerons que $f_{P_i} \circ f_{P_j} \neq f_{P_j} \circ f_{P_i}$. Par conséquent, si une partie du système change ses liens d'interaction, le système produira une nouvelle fonction. Un changement de liens peut correspondre à une destruction, une création ou une modification de passage de valeurs de partie en partie pour des systèmes fonctionnels ou organisationnels. Pour des systèmes spatialement distribués, une modification de la position dans l'espace implique un changement d'interaction entre les parties en mouvement conduisant à un changement de fonctionnalité pour le voisinage et donc pour le système dans son ensemble. Ce principe est appelé *principe d'auto-organisation*.

Du point de vue méthodologique, nous pouvons alors considérer un système artificiel dont le concepteur a seulement pré-donné les spécifications et comportement des parties qui le composent et dont il sait qu'elles sont suffisantes pour obtenir une organisation produisant une fonction adéquate. Ici, le concepteur n'a pas à donner une organisation interne *a priori*. L'organisation est un résultat de l'activité du système. Ainsi, nous remplaçons la tâche de définition de la fonction globale du système, par la définition des fonctions des sous-parties du système et des moyens de réorganisation : quand et comment réorganiser les parties pour obtenir une configuration produisant une fonction en adéquation avec l'environnement.

1.1.2.2 L'auto-organisation dans la nature

L'auto-organisation n'est pas un principe purement théorique, mais un phénomène souvent observé dans la nature. Voici quelques exemples significatifs provenant de la physique, de la biologie, de la chimie, ou de l'éthologie correspondants à des niveaux différents d'auto-organisation :

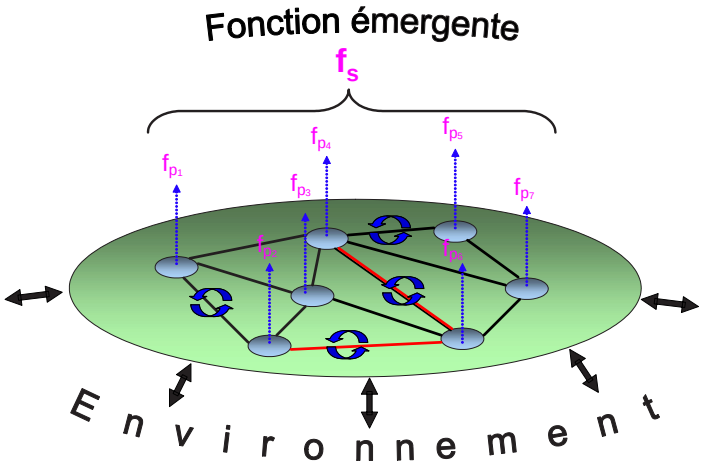

Figure 1.2 — Adaptation à un environnement dynamique et auto-organisation.

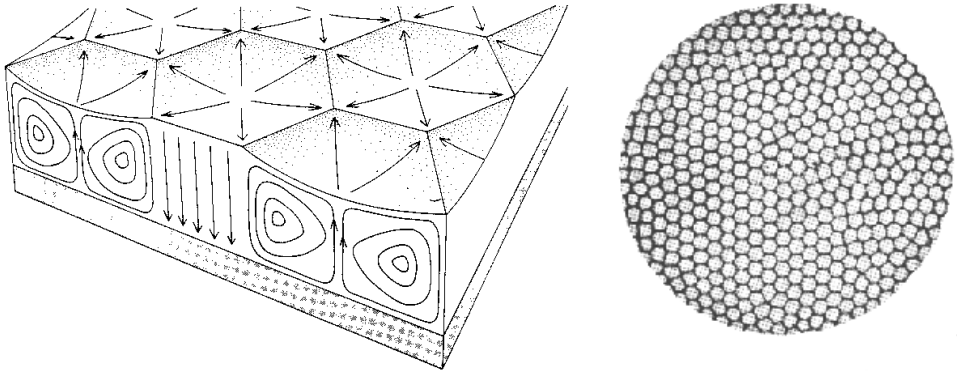

Figure 1.3 — Un exemple d'auto-organisation : l'expérience des cellules de Bénard.

Auto-organisation d'ordre moléculaire. L'expérience des cellules de Bénard permet d'observer un phénomène d'auto-organisation des molécules en cellules de convection hexagonales (voir figure 1.3). Dans cette expérience, où l'on chauffe un liquide par le bas, le système atteint un état d'équilibre lorsque la chaleur du bas circule en flux vers le haut à cause de la différence de densité. Par contre, si la chaleur continue à augmenter, le système passe d'un état d'équilibre à un état chaotique (remous), dû à la trop grande différence de température entre le haut et le bas, qui s'intensifie jusqu'à ébullition. Mais avant d'atteindre cet état, le système passe par un point critique "au-delà de l'équilibre" durant lequel les flux se forment en hexagones réguliers sur la surface du liquide. Si la température continue à monter, le phénomène disparaît. Ici, dans un système physique complexe, de l'auto-organisation de millions de molécules résulte un phénomène émergent, une structure observable cohérente, dans un état de non-équilibre proche du chaos.

Un autre exemple est la réaction de Belousov-Zhabotinski dans laquelle apparaît une oscillation de la couleur lors de l'oxydation de l'acide malonique ou propanedioïque par l'ion bromate catalysée par le cérium. Un cliché instantané de la solution montre une formation de spirales et de formes circonvolutoires alternées comme dans la figure 1.4. Ce phénomène auto-organisé est le résultat de processus pouvant être parfaitement représentés par le modèle de réaction-diffusion de Turing [Turing, 1952].

Auto-organisation d'ordre cellulaire. Le modèle réaction-diffusion de Turing a tout d'abord été utilisé pour expliquer les processus de morphogénèse des organismes ainsi que la formation de rayures sur le pelage d'animaux [Ball, 1998]. De la même manière, les formes triangulaires apparaissant sur certains coquillages peuvent être modélisées et simulées par les automates cellulaires, initiés par Ulam et von Neumann [von Neumann, 1966; Holland, 1998]. Un autre exemple est celui de l'ontogenèse auto-organisée au sein du système biologique en devenir où la différenciation et l'organisation de cellules simples induit la constitution de structures complexes voire conscientes [Maturana et Varela, 1994].

Auto-organisation d'ordre social. De la même manière que les biologistes, les physiciens ou les mathématiciens, les spécialistes du comportement (éthologues, psychologues ou sociologues) ont identifié des phénomènes auto-organisés dans des sociétés d'indivi-

Figure 1.4 — La réaction de Belousov-Zhabotinski : un exemple d'auto-organisation par réaction-diffusion.

dus comme chez les fourmis, les termites ou bien les humains [Bonabeau *et al.*, 1997]. L'exemple classique est le fourragement chez les fourmis qui empruntent toujours le chemin le plus court entre deux points sur lequel elles effectuent des allers-retours et ce sans connaître *a priori* leur environnement, sans communication directe, mais seulement grâce à un dépôt phéromonal le long du trajet. Les règles que suivent les fourmis sont simples : les phéromones les attirent (proportionnellement à la densité) et elle déposent des phéromones au cours de leur déplacement. Plus un trajet est emprunté par l'ensemble des fourmis (donc densité plus élevée en phéromones), plus les fourmis emprunteront ce chemin. Ce phénomène auto-organisé et auto-entretenu est très robuste aux variations de l'environnement si, par exemple, le trajet le plus court était bouché, le collectif emprunterait le second trajet le plus court. Des modèles probabilistes permettent d'expliquer ou de simuler ces comportements et ont donné naissance aux *ant algorithms* [Colorni *et al.*, 1992; Dorigo *et al.*, 1996] et au concept de *swarm intelligence* [Bonabeau *et al.*, 1999].

D'autres animaux ont des comportements collectifs auto-organisés, comme les termites dans la construction de leur nid. Certains comportements grégaires comme ceux des buffles, des étourneaux ou des maquereaux sont identifiés comme résultant d'une auto-organisation. D'autres comportements auto-organisés apparaissent dans des sociétés humaines par exemple, comme la répartition des agglomérations sur un territoire ou bien la synchronisation d'applaudissements dans un public. Ce dernier phénomène a donné lieu à de nombreux modèles mathématiques, mais de trop haut niveau conceptuel pour expliquer les comportements de chaque individu.

Ainsi, nous avons vu grâce à cet ensemble non exhaustif, de phénomènes naturels d'auto-organisation, la possibilité de modéliser des systèmes à partir de règles simples de niveau local (les parties du systèmes) afin d'obtenir des systèmes plus robustes. Néanmoins, Heylighen soulève le problème de la difficulté de contrôler de tels systèmes ainsi que la *quasi*-impossibilité de prouver des propriétés, comme la terminaison par exemple [Heylighen, 1999]. Ceci est en partie dû au caractère non-linéaire des systèmes auto-organisateurs [Prigogine et Nicolis, 1977]. Heylighen établit que la meilleure approche possible est d'identifier des *points leviers*, c.-à-d. des paramètres dont de faibles variations entraînent des effets remarquables mais prédictibles.

1.1.3 Le calcul émergent

Le but de tout système artificiel est de calculer, bien que l'interprétation de ce calcul soit directe ou indirecte. Dans le cadre des systèmes à fonctionnalité émergente, ce calcul va donc être de nature émergente. Le terme de calcul émergent, ou *emergent computation*, a été défini par Forrest [Forrest, 1991]. Plusieurs techniques se regroupent sous ce terme comme les modèles connexionnistes, les systèmes à classifieurs, les automates cellulaires, les modèles biologiques, les modèles de vie artificielle, et l'étude de la coopération dans les systèmes sociaux sans autorité centrale. Le but est que le résultat du calcul demandé soit un épiphénomène du système dû aux activités *parallèles* des parties du système. En d'autre mots, le système ne calcule pas à partir d'une formule pré-définie, mais il est la formule (ou l'algorithme ou le modèle). Par exemple, dans l'environnement de programmation émergente de Georgé, les instructions d'un programme s'auto-organisent pour obtenir une fonction calculante cohérente [Georgé, 2004].

L'intérêt d'une telle approche est le déplacement, et l'atténuation, de la difficulté de spécification de la fonction de macro-niveau (que doit calculer le système dans son ensemble) vers les sous-parties du système. Malgré tout, bien que la composante fonctionnelle des sous-parties soit d'ordre de difficulté inférieur, la prise en compte de la dynamique des interactions reste le point clé de la conception de tels systèmes –une fois les parties identifiées. Ceci implique que les parties du système ne doivent pas être de simples fonctions répondant aux requêtes, mais plutôt des entités autonomes, capables de décisions plus ou moins fines et interagissantes de manière pro-active. Cette remarque justifie l'utilisation des systèmes multi-agents comme paradigme de programmation de systèmes artificiels auto-organisateurs.

1.1.4 Adaptation et apprentissage par auto-organisation

Nous avons vu que par la modification de son organisation interne, un système auto-organisateur va produire une fonction différente. Ce changement est dû à la pression environnementale, mais ne va pas se faire de façon immédiate la plupart du temps. Le système s'adapte pas à pas à son environnement de manière non supervisée. Le système et son environnement sont sous influence mutuelle, c.-à-d. un processus d'ajustement mutuel dû au couplage structurel, comme la notion de feedback en cybernétique [von Foerster, 1960]. Cette adaptation progressive correspond à un mode d'apprentissage comme on peut l'observer dans les réseaux de neurones formels mais aussi dans les réseaux bayésiens. Mais ici, contrairement aux méthodes classique d'apprentissage, l'adaptation du système n'est pas fondée sur une évaluation globale, comme par la sélection des "meilleurs" individus lors de la phase de sélection d'un algorithme génétique ou bien par la connaissance de l'erreur entre la sortie effective et la sortie théorique dans un réseau de neurones.

Concernant la conception de systèmes adaptatifs, et donc apprenants, ces remarques ont un écho important. Ceci implique une possible décomposition récursive du système si une ou plusieurs de ses parties doivent être dotées de capacités d'apprentissage. En effet, une partie apprenante pourra produire la fonction souhaitée grâce à un sous-système auto-organisateur comme dans ABROSE ou STAFF par exemple (voir paragraphes 1.4.2 et 1.4.3)

[Gleizes *et al.*, 2000; Georgé *et al.*, 2003b].

1.2 Les systèmes multi-agents

Le paragraphe précédent conclut sur le besoin de se munir de paradigmes de modélisation dans lesquels un système peut être décomposé en sous-parties autonomes et actives : les systèmes multi-agents. Ce paragraphe présente les notions d'agent, de système multi-agent et d'environnement qui sont les concepts clés de ces systèmes. La notion d'agent est issue de l'évolution du domaine de l'intelligence artificielle distribuée, comme dans DVMT de Lesser et Corkill [Lesser et Corkill, 1983]. A partir du moment où le contrôle des entités fut décentralisé, MACE de Gasser *et al.* par exemple, les termes de système multi-agent et d'agent apparurent comme paradigme de conception de système artificiels [Gasser *et al.*, 1987].

1.2.1 La notion d'agent

Une définition communément admise du concept d'agent est la suivante [Ferber, 1995, chap. 1] : un agent est une *entité physique ou virtuelle* capable d'*agir* (ou *communiquer*) sur son environnement de manière *autonome* et ce en fonction de *perceptions et connaissances partielles*. Le comportement de l'agent est guidé par un ou des objectifs individuels. Les agents sont destinés à être plongés dans un environnement afin d'effectuer une tâche en commun qui sera déterminée par les *compétences* des agents. La distribution des tâches sera guidée par les accointances de ces agents, c.-à-d. les interactions entre agents. Ferber distingue deux sous-types d'agents : les *agents situés* qui évoluent dans un environnement physique, dont les actions concernent des modifications des objets l'entourant, et les *agents communicants* dont les actions sont purement sociales (envoi de messages, d'intentions, etc.). Bien sûr, un agent peut être à la fois situé et communicant, c'est le cas de robots capables de soulever des boîtes et de communiquer par radio, par exemple.

L'autonomie d'un agent peut-être définie en trois points [Arlabosse *et al.*, 2004] :
- les agents ont une existence propre et indépendante de l'existence des autres ;
- les agents sont capables de maintenir leur viabilité dans des environnements dynamiques, sans contrôleur extérieur ;
- les agents sont capables de prises de décision quant à leur comportement à venir (action, communication).

Cette définition de l'autonomie des agents reste toutefois limitée. Les premier et deuxième points doivent se préciser en une indépendance vis-à-vis des entités de même niveau conceptuel – un agent est autonome face aux autres agents. En effet un système artificiel sera toujours fortement dépendant de l'environnement d'exécution, du système d'exploitation ou de l'architecture réseau dans lesquels il prend place. Le troisième point constitue l'un des principaux enjeux de l'intelligence artificielle actuelle. Néanmoins, ce point sera souvent traduit comme une directive de programmation tendant à éviter des appels directs de méthodes des objets représentant les agents, en programmation orientée objet par exemple.

Le comportement interne d'un agent est souvent représenté comme un cycle de vie *perception-décision-action*. Lors de la phase de perception, l'agent met à jour ses connexions avec l'environnement (senseurs, capteurs ou boîte aux lettres). Lors de la phase de décision, les perceptions permettent à l'agent de mettre à jour ses connaissances et croyances sur le monde, puis de décider d'une action à exécuter pendant la phase d'action. Ce cycle de vie se retrouve à la fois dans les agents réactifs mais aussi dans certaines architectures d'agents comme les BDI (*Belief-Desire-Intention*) où un agent, à partir d'événements extérieurs, met à jour ses buts et sa base de faits ; à partir de ses nouvelles connaissances, choisit le plan approprié pour atteindre ses buts ; puis à partir du plan déduit ses intentions d'exécution ; puis, enfin, passe à l'action [Rao et Georgeff, 1995].

1.2.2 La notion de système multi-agent

Agréger des agents dans un ensemble ne signifie pas forcément que le système est un système multi-agent. Parler de systèmes multi-agents implique la prise en compte de problématiques organisationnelles et collectives portant sur les entités, la dynamique et la structure [Arlabosse *et al.*, 2004].

Un agent au sein d'un système peut avoir un *rôle* ou une certaine *tâche* à remplir vis-à-vis des autres agents ; ce qui n'est pas le cas dans des systèmes mono-agents. La dynamique induite par le regroupement d'agents doit être prise en compte à la fois au niveau des agents (Quels sont leurs plans ? Quels service offrent-ils aux autres ? Quelles capacités collectives possèdent-ils ?) et au niveau du collectif (Quelles sont les interactions possibles ? Quelles collaborations/coopérations/formations peuvent-ils adopter ?). Enfin, la façon dont sont agencés les agents au sein de la société et dans quel contexte est chose primordiale. Les notions d'organisation, avec des modèles comme AGR (Agent/Groupe/Rôle) [Ferber et Gutknecht, 1998], ou bien d'infrastructures collaboratives, sont des sujets de recherche importants dans le domaine multi-agent.

Un système multi-agent est donc un système composé d'agents autonomes ayant pour but de fournir une fonction collective. Les mécanismes permettant d'obtenir cette fonction prennent en compte les enjeux sociaux entre agents, comme la coopération. Dans ce mémoire, nous nous focaliserons sur les aspects méthodologiques de la mise en place de comportements coopératifs afin d'obtenir des systèmes auto-organisateurs.

1.2.3 La notion d'environnement

Ferber considère qu'un système multi-agent est composé d'agents, d'objets, d'un environnement et d'opérations [Ferber, 1995]. Cette définition va à l'encontre des définitions utilisées en biologie ou physique pour l'auto-organisation et l'émergence, dans lesquelles l'environnement est tout ce qui n'est pas agent. Ici, nous choisirons de présenter l'environnement selon deux axes :

- *l'environnement d'un agent* est constitué de tout ce qui l'entoure : les objets ou les autres agents. On distingue deux types d'environnements, physique et social, pour un agent :
  - l'*environnement physique* correspond à tous les objets que l'agent peut percevoir ou sur lesquels il peut agir. Par exemple, si l'on considère une fourmi comme étant un

agent, son environnement physique est composé de nourriture et d'obstacles. On parle d'environnement physique essentiellement dans le cas d'agents situés ;
- l'*environnement social* correspond aux autres agents avec lesquels un agent est en interaction par envoi de message par exemple. On parle d'environnement social surtout dans le cas d'agents communicants.

▷ *l'environnement du système* est composé de tout ce qui n'est pas dans le système : les objets ou d'autres agents faisant partie d'un autre système multi-agent. L'environnement du système est ce qui va le guider vers une adéquation fonctionnelle dans le cas de systèmes auto-organisateurs.

Concernant sa nature et sa dynamique, l'environnement peut être caractérisé en utilisant les termes suivants [Russel et Norvig, 1995; Wooldridge, 2000; Lind, 2001] :

Environnement accessible (par opposition à "inaccessible"). Le système peut obtenir une information complète, exacte et à jour sur l'état de son environnement. Dans un environnement inaccessible, seule une information partielle est disponible. Par exemple, un environnement tel que l'Internet n'est pas accessible car il est impossible de tout connaître à propos de lui. Un robot qui évolue dans un environnement possède des capteurs pour le percevoir et, en général, ces capteurs ne lui permettent pas de connaître tout de son environnement qui est alors considéré comme inaccessible.

Environnement continu (par opposition à "discret"). Dans un environnement continu, le nombre d'actions et de perceptions possibles dans cet environnement est infini. Dans un environnement discret, le système possède des perceptions distinctes, clairement définies qui décrivent l'environnement. Par exemple, dans un environnement réel tel que l'Internet, le nombre d'actions qui peuvent être effectuées par les utilisateurs est illimité. Mais, dans un environnement simulé tel qu'un éco-système, le nombre d'actions ou de perceptions qu'une entité virtuelle (comme une fourmi ou un robot) peut être limité, l'environnement est alors discret.

Environnement déterministe (par opposition à "non déterministe"). Dans un environnement déterministe, une action a un effet unique et certain. Si le système agit dans son environnement, il n'y a aucune incertitude sur l'effet de son action sur l'état de l'environnement. L'état suivant de l'environnement est complètement déterminé par l'état courant. Dans un environnement non déterministe, une action n'a pas un effet unique garanti. Par nature, le monde physique réel est un environnement non déterministe.

Environnement dynamique (par opposition à "statique"). L'état d'un environnement dynamique dépend des actions du système qui se trouve dans cet environnement mais aussi des actions d'autres processus. Aussi, les changements ne peuvent pas être prédits par le système. Un environnement statique ne peut changer sans que le système agisse. Par exemple, l'Internet est un environnement hautement dynamique, son changement n'est pas simplement lié aux actions d'un seul utilisateur (vu comme un système plongé dans cet environnement).

1.2.4 L'adaptation par auto-organisation

Les systèmes multi-agents ne sont par forcément des systèmes auto-organisateurs : dans certains cas, une organisation complète peut être déterminée *a priori*. Certains modèles

multi-agents, comme AGR, proposent l'organisation comme moyen de spécification d'un système plutôt qu'un résultat de l'activité des agents observable *a posteriori* [Ferber et Gutknecht, 1998]. Dans le cas contraire, celui qui nous intéresse, le système doit s'adapter à son environnement par auto-organisation des agents. L'organisation finale observée est alors la plus adaptée à l'état courant de l'environnement. Ceci nécessite néanmoins de définir un critère de réorganisation. Ces critères correspondent à une heuristique d'exploration de l'espace des possibilités de fonctions du système : gestion de coûts, modèles stochastiques, modèles inspirés de la nature, ou bien des modèles inspirés de la sociologie, comme la coopération. Cardon et Guessoum proposent l'évaluation de l'écart entre le sens de variation du comportement de l'agent et la variation de l'organisation soumise à la pression environnementale, appelée tendance fondamentale [Cardon et Guessoum, 2000], ce qui implique une plasticité de la structure organisationnelle.

1.3 La coopération comme critère d'auto-organisation : la théorie des AMAS

La théorie des systèmes multi-agents adaptatifs, *Adaptive Multi-Agent Systems* (ou *AMAS*), repose sur le principe que si tous les agents d'un système sont coopératifs, alors le système est fonctionnellement adéquat et donc adapté à son environnement. Ici, la notion de coopération, inspirée de la métaphore sociale, est à la fois définie pour les agents et pour les systèmes multi-agents de manière proscriptive comme critère de réorganisation.

1.3.1 La notion de coopération

De nombreux travaux sur la coopération ont montré l'intérêt d'une telle approche comme moyen d'optimisation des performances (collectives ou individuelles), basé des fonctions de satisfaction de coûts par exemple. La coopération est aussi un moyen pour des agents d'accomplir des tâches impossibles pour un individu isolé [Ferber, 1995] et donc obtenir des fonctionnalités de plus haut niveau, comme s'adapter ou apprendre [Glize *et al.*, 1997].

Nous pouvons distinguer deux types de coopération : la coopération intentionnelle et la coopération réactive. La coopération intentionnelle est issue de l'intelligence artificielle distribuée, et implique des capacités cognitives de représentation du monde et de planification, comme dans les travaux initiateurs des systèmes multi-agents [Durfee *et al.*, 1987]. L'intentionnalité implique l'étude des états mentaux des agents et de leurs attitudes, à laquelle certains travaux de Castelfranchi ont apporté des formalisations logiques précises [Conte, 1992; Castelfranchi, 1998]. Un autre type de coopération, fortement inspiré de la nature et des insectes sociaux, se focalise sur la coopération d'agents réactifs ne répondant qu'aux stimuli de leur environnement comme dans les travaux de Deneubourg *et al.* [Deneubourg *et al.*, 1990], de Steels sur les collectifs de robots [Steels, 1996], de Drogoul sur la simulation de collectifs humains pour robotiques [Drogoul, 2000] ou bien de Topin *et al.* sur la simulation de fourmis coopératives dans ANTS [Topin *et al.*, 1999a,b].

Afin de proposer une méthode de conception, nous allons donner notre définition de la

coopération, issue, notamment, de ces derniers travaux, après avoir présenté brièvement les bases de la théorie des AMAS.

1.3.2 L'adéquation fonctionnelle et la coopération

Glize distingue trois catégories d'interactions système-environnement [Glize, 2001] :
- *coopérative* : l'activité du système sur son environnement et inversement implique une modification favorisant les activités de l'autre, comme dans les organismes symbiotiques par exemple ;
- *antinomique* : l'activité du système sur son environnement et inversement empêche l'autre de réaliser son activité ;
- *indifférente* : l'activité du système sur son environnement et inversement n'a aucune conséquence sur l'autre.

Une activité non coopérative est donc une activité antinomique ou indifférente. Le rapport entre la nature de l'activité système-environnement et l'adéquation fonctionnelle du système, c.-à.-d. s'il réalise une fonction adaptée à son environnement, se pose dans l'axiome suivant :

Axiome. *Un système fonctionnellement adéquat n'a aucune activité antinomique sur son environnement.*

En effet, un système doit au moins posséder la propriété de ne pas perturber son environnement, dans le cadre du couplage système-environnement. La véracité de cet axiome ne peut pas être prouvée, car il faudrait un observateur extérieur à l'activité de tous les systèmes évoluant dans un certain univers physique, tout en n'interagissant aucunement avec celui-ci afin de ne pas le perturber. A partir de cet axiome, Glize construit un raisonnement important sur les relations d'inclusion des différents types de systèmes :

Lemme 1. *Tout système coopératif est fonctionnellement adéquat.*

La démonstration de ce lemme est basée sur l'axiome précédent et sur le fait qu'un système coopératif n'a pas d'activité antinomique ou indifférente.

Lemme 2. *Pour tout système S fonctionnellement adéquat, il existe au moins un système coopératif $S^\star$ qui soit fonctionnellement adéquat dans le même environnement.*

La démonstration consiste en une expérience de pensée de déconstruction du système S pour en construire un nouveau $S^\star$. Elle est réalisée en quatre étapes : définir un algorithme de construction d'un système coopératif, montrer que cet algorithme se termine, montrer que le système coopératif obtenu est équivalent au système initial pour l'environnement, montrer que le nouveau système est fonctionnellement adéquat.

Lemme 3. *Tout système à milieu intérieur coopératif est un système coopératif.*

Le milieu intérieur correspond aux parties du système ainsi qu'aux supports physiques nécessaires à leurs échanges. Un système à milieu intérieur coopératif possède des échanges coopératifs avec son environnement, car ceux-ci sont un sous-ensemble des échanges que réalisent ses parties.

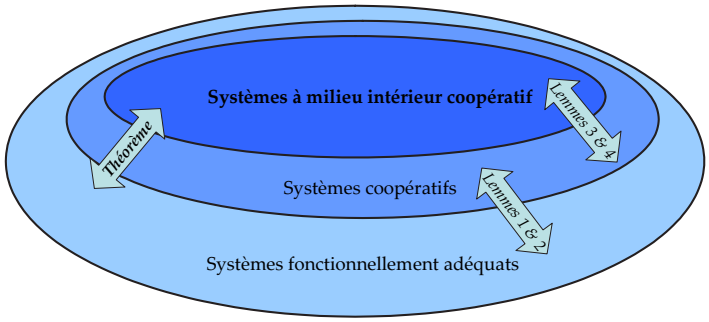

Figure 1.5 — Le théorème de l'adéquation fonctionnelle : les relations d'inclusion entre systèmes adéquats et coopératifs.

Lemme 4. *Pour tout système coopératif S, il existe au moins un système à milieu intérieur coopératif S* qui réalise une fonction équivalente dans le même environnement.*

La méthode est identique à celle du lemme 2, mais ici l'objet de la construction est maintenant chaque partie du système. Ce raisonnement conduit naturellement au théorème suivant qui est résumé dans la figure 1.5 [Camps *et al.*, 1998; Glize, 2001] :

Théorème de l'adéquation fonctionnelle. *Pour tout système fonctionnellement adéquat, il existe au moins un système à milieu intérieur coopératif qui réalise une fonction équivalente dans le même environnement.*

1.3.3 Conséquences méthodologiques

Le théorème de l'adéquation fonctionnelle a d'importantes répercussions sur la conception de systèmes multi-agents adaptatifs par auto-organisation. Pour tendre vers une adéquation fonctionnelle du système avec son environnement, on va chercher à établir et maintenir un état interne coopératif en se focalisant sur les interactions entre les agents. Mais, tant que nous ne pouvons pas spécifier de manière sûre le comportement coopératif des parties, nous décomposons les parties en sous-parties plus élémentaires capables de s'auto-organiser pour obtenir la fonction non formellement spécifiée.

L'auto-organisation a lieu lorsque le système n'est plus fonctionnellement adéquat, donc lorsqu'un ou plusieurs agents sont dans des états, jugés localement, de non coopération avec leur environnement. Rendre les agents coopératifs se limitera donc à fournir des règles de détection de la non coopération et des moyens d'en sortir pour revenir (pas toujours immédiatement) dans un état d'interaction coopérative. Enfin, pour concevoir un système fonctionnellement adéquat, il suffit de s'assurer que tous les agents sont coopératifs (ou en ont les moyens) ; ce qui restera notre ligne de conduite tout au long de ce travail.

1.3.4 Définition de la coopération

Les résultats précédents ne font aucun présupposé sur la notion de coopération. Pourtant, ceci ne signifie pas pour autant que toutes les définitions sont acceptables. Considérant

des agents suivant un cycle *perception-décision-action*, voici une définition de la coopération en trois points correspondant à ces trois phases [Glize, 2001] :

Définition. *Un agent est dit* ***coopératif*** *si et seulement si* $c_{per} \wedge c_{dec} \wedge c_{act}$ *est vraie avec :*

- c_{per} *: l'agent est apte à tout instant à interpréter de manière non ambiguë un signal de son environnement (physique ou social) ;*
- c_{dec} *: l'agent peut employer ses propres compétences sur l'information produite par le traitement du signal ;*
- c_{act} *: la transformation du milieu que l'entité réalise est profitable à son environnement.*

Une définition proscriptive de la coopération, permettant de définir les conditions de réorganisation des agents (voir paragraphe précédent), est donc la suivante :

Définition. *Un agent est dit en* ***situation non coopérative*** *(ou* ***SNC****) si et seulement si* $\neg c_{per} \vee \neg c_{dec} \vee \neg c_{act}$ *est vraie.*

Un agent peut donc être dans trois SNC différentes ($\neg c_{per}$, $\neg c_{dec}$ ou $\neg c_{act}$) ayant chacune des interprétations différentes :

$\neg c_{per}$	L'*incompréhension* est détectée lorsque l'agent est incapable d'extraire le contenu informatif d'un signal. L'*ambiguïté* est détectée lorsque l'agent attribue plusieurs interprétations à un même signal.
$\neg c_{dec}$	L'*incompétence* survient lorsque l'agent est incapable d'exploiter dans son raisonnement un signal reçu. L'*improductivité* survient lorsque l'utilisation d'un signal ne débouche sur aucune conclusion.
$\neg c_{act}$	La *concurrence* est une situation dans laquelle un agent croit que l'action d'un autre agent va aboutir au même état du monde (les deux agents ont le même but). Le *conflit* correspond à une situation lors de laquelle un agent croit que l'action d'un autre agent va être incompatible avec la sienne (les deux agents ont deux buts antinomiques). L'*inutilité* survient lorsqu'un agent croit que son action ne va pas modifier le monde (l'agent ne fait rien par exemple).

1.3.5 Discussion

Contrairement aux algorithmes classiques d'apprentissage comme les réseaux neuronaux ou les algorithmes génétiques, l'adaptation par coopération n'introduit pas de biais conduisant à des attracteurs locaux, empêchant le système de devenir fonctionnellement adéquat. En effet, le coeur de l'adaptation réside dans le couplage système-environnement

qui guide l'organisation interne et non l'algorithme au sens strict du terme [Glize, 2001]. Par contre, le contrecoup d'une telle approche est l'impossibilité de démontrer des propriétés de convergence ou d'arrêt sans faire d'hypothèses fortement contraignantes sur l'environnement.

La définition précédente de la coopération permet d'envisager des agents coopératifs intentionnels ou réactifs, communicants ou situés, comme nous le montreront les exemples du paragraphe 1.4.

Enfin, les AMAS possèdent plusieurs propriétés permettant de situer cette théorie par rapport aux théories de l'émergence (paragraphe 1.1) :

- L'application matérielle de la théorie des AMAS se situe dans le cadre de la théorie des systèmes ;
- Un système coopératif dans l'environnement est fonctionnellement adéquat, ce qui lui évite de connaître la fonction globale qu'il doit réaliser pour s'adapter ;
- Même si un système n'a pas de but, il peut agir pertinemment dans son milieu. En fonction de ses perceptions de l'environnement, des représentations qu'il en possède et de ses compétences, il agira au mieux pour que son comportement soit coopératif ;
- La notion de feedback n'est pas contraignante dans cette théorie car le système doit seulement juger si les changements s'opérant dans le milieu sont coopératifs de son point de vue sans savoir si ces changements dépendent de ses propres actions passées.

Ces points confortent l'idée que les systèmes multi-agents sont un moyen technique pertinent pour la conception de systèmes à fonctionnalité émergente, comme vont l'illustrer les exemples du paragraphe suivant.

1.4 De la théorie à la pratique

Depuis quelques années la théorie des AMAS a été appliquée à divers problèmes de natures différentes et permettant de préciser certains notions comme la définition des SNC ou bien la décomposition d'agents en systèmes multi-agents [Topin *et al.*, 1999a; Gleizes *et al.*, 2000; Georgé *et al.*, 2003b].

1.4.1 ANTS : simulation de fourmilière

ANTS est une application ayant pour but de simuler le comportement d'une fourmilière lors de tâches de fourragement, c.-à-d. de recherche de nourriture. Ici la fourmilière est modélisée comme un système multi-agent devant s'adapter au mieux à son environnement physique constitué de points de nourriture, d'obstacles et de sol praticable. Les agents sont facilement identifiables : les fourmis fourrageuses.

Le comportement nominal des fourmis a été déterminé par des éthologues, comme des vecteurs de préférences sur les actions à mener en fonction des perceptions [Topin *et al.*, 1999a,b]. Les fourmis explorent l'environnement et déposent des phéromones lors de leurs déplacements. Ainsi, comme dans l'exemple donné dans le paragraphe 1.1.2.2, plus un chemin est fréquenté, plus il attire les fourmis par une forte concentration en phéromones.

Le but de l'application de la théorie des AMAS à ce problème n'était pas de simplement

modéliser le comportement des fourmis dites "naturelles", mais de montrer l'apport de comportements coopératifs en terme d'efficacité, c.-à-d. le nombre de ressources rapportées au nid en un temps donné. Topin *et al.* ont identifié six situations (SNC) dans lesquelles un comportement coopératif pourrait améliorer le comportement nominal des fourmis naturelles. Les fourmis ainsi obtenues sont appelées robots-fourmis :

1. Un premier cas de *concurrence* survient lorsqu'une fourmi voit deux ressources. Une fourmi naturelle est attirée proportionnellement par la plus importante. Mais pour être plus coopératif, le robot-fourmi va vers la ressource ayant le plus de disponibilité, même si elle est moins importante, ou si la ressource se trouve plus loin, pour ne pas aboutir au même état du monde où les ressources sont inégalement exploitées.
2. Un deuxième cas de *concurrence* survient lorsqu'une fourmi en mode exploration voit une autre fourmi en exploration. Une fourmi naturelle ira indifféremment vers la fourmi ou ailleurs. Le robot-fourmi est en SNC car il visitera une zone déjà couverte par l'autre robot-fourmi. Il évite donc cette zone et il n'ira pas dans la direction du robot-fourmi perçu afin d'améliorer l'exploration collective de l'environnement.
3. Une troisième situation de *concurrence* survient quand une fourmi suit sa piste de phéromone et voit des ressources. Une fourmi naturelle est attirée proportionnellement par la phéromone et la ressource, elle a donc plus de chance de suivre la piste, qui comporte a priori une grande concentration de phéromone. Un robot-fourmi risque de se retrouver en situation de concurrence s'il se dirige vers des ressources exploitées par d'autres agents. Pour éviter cette concurrence potentielle, il ira s'il le peut vers des ressources nouvelles. Il va vers les ressources, même si la phéromone est présente en grande quantité.
4. Parce qu'un robot-fourmi est coopératif, il donne spontanément des informations aux autres robots-fourmis s'il croit que ces informations peuvent leur être utiles. C'est ce que l'on appelle la communication spontanée. Ainsi, une fourmi qui rentre au nid et voit en passant des ressources, détient une information sur la localisation de ces ressources. Une fourmi naturelle ne prend pas en compte l'information liée à la perception de cette ressource. Par contre le robot-fourmi continue son chemin vers le nid en déposant plus de phéromone dans le but de donner des informations plus exactes aux autres sur l'environnement tel qu'il le perçoit, sinon il serait en situation d'*inutilité*.
5. Une fourmi doit retourner au nid sans être chargée au maximum parce que son temps maximum hors du nid est écoulé. Une fourmi naturelle dépose de la phéromone en retournant au nid, comme elle le fait dans tous les cas puisqu'elle est chargée. Le robot-fourmi, par souci de bien informer ses partenaires, ne laisse pas de trace de phéromone, sinon il est en situation d'*inutilité*. Ici aussi, l'agent dispose d'une information supplémentaire : il n'y a plus de ressource là où il se trouve. Pour coopérer, il communique cela aux autres, en ne déposant pas de phéromone. Si, par la suite, l'agent découvre des ressources, il le communiquera aux autres.
6. Une fourmi qui rentre au nid en ayant trouvé des ressources, recrute d'autres fourmis. La fourmi naturelle recrute en fonction de la quantité de nourriture ramenée au nid. Toujours par souci d'informer de manière la plus juste possible, le robot-fourmi recrute en fonction de la dernière quantité de phéromone déposée. En effet, il dispose d'une

information supplémentaire : la somme des ressources détectées pendant le trajet. Ce critère est bien meilleur pour décider que d'autres robots-fourmis doivent sortir du nid que celui de la nourriture rapportée. Encore, une fois, s'il n'agissait pas ainsi, le robot-fourmi serait *inutile*.

Les résultats présentés par Topin *et al.* sont significatifs de l'apport de tels comportements coopératifs et montrent que les robots-fourmis, ayant les mêmes caractéristiques physiologiques que les fourmis œcophylles, explorent de manière plus optimale l'environnement [Topin *et al.*, 1999b].

1.4.2 ABROSE : courtage en ligne

ABROSE (Agentbased Brokerage Services in Electronic Commerce) est un outil de commerce électronique de médiation de services fondée sur des agents. Le but est d'ouvrir aux utilisateurs, clients ou fournisseurs de services, des opportunités de contacts. Les clients lancent des requêtes en langage naturel, et le système doit leur proposer des fournisseurs de services abonnés les plus pertinents compte-tenu de la requête [Gleizes *et al.*, 2000; Gleizes et Glize, 2000].

Gleizes *et al.* identifient un niveau d'agent de transaction (TA) ayant pour but de représenter un utilisateur sur une place de marché. Lorsqu'un utilisateur cherche un contact, il demande à son TA de lui trouver les adresses les plus pertinentes. Le TA va coopérer et prospecter auprès des autres TA de la place de marché, et enfin rapporter ses résultats à l'utilisateur. Ce dernier peut donner son jugement quant au résultat apporté, par un feedback discret ("ok", "pas ok", "indifférent"). En fonction de ce retour, le TA doit ajuster ses accointances afin de modifier les résultats pour une éventuelle prochaine requête. Cet apprentissage, comme proposé dans le paragraphe 1.3.3, est effectué grâce à une décomposition du TA en un système multi-agent de plus bas niveau raisonnant sur les termes utilisés dans les requêtes de l'utilisateur. Les avantages de cette approche sont que l'application est indépendante du langage et du sujet des requêtes.

Chaque TA possède un réseau d'agents de croyances (BA) représentant un terme du langage naturel, qui se met à jour en fonction des requêtes. Plus un mot est utilisé dans les requêtes de l'utilisateur, plus il va prendre d'importance dans le réseau. Ainsi, lors de requêtes, le TA va proposer les contacts en correspondance avec l'état du réseau de BA. Le même genre de décomposition a été utilisé pour du *profilage adaptatif* ou bien de la *e-éducation* [Georgé *et al.*, 2003a].

1.4.3 STAFF : prévision de crues

L'objet du projet STAFF (Software Tool for Adaptive Flood Forecast) était de réaliser une prévision adaptative de crues. Plus généralement, il s'agissait de fournir un outil de prévision auto-adaptatif dans tout environnement qui dispose de capteurs pour l'observation des phénomènes naturels. Le logiciel STAFF utilise un modèle adaptatif de prévision de crues, lequel est composé d'un AMAS. Chacun des agents a pour objectif de calculer la variation de hauteur d'eau pour un délai unitaire et utilise pour cela une somme pondérée. Le caractère adaptatif du modèle est obtenu grâce à la coopération entre les agents et à l'ajustement de

leurs poids. C'est ce qui rend le modèle générique et améliore ses performances [Sontheimer et Glize, 2000; Régis *et al.*, 2002].

Un système multi-agent est installé sur une station et est connecté aux différents capteurs présents dans le bassin. Chaque capteur est représenté par un agent. Mais ici, la décomposition du système n'est pas immédiate et évidente. En effet, afin d'établir une prévision juste, le système est décomposé en agents horaires devant fournir une prévision pour une heure donnée. Par exemple, pour obtenir une prévision à trois heures, le système station sera composé de trois agents horaires : un agent pour la prévision à $T + 1$ heure, un agent pour $T + 2$ heures et un agent pour $T + 3$ heures. Ces agents doivent se mettre en accord sur la prévision en fonction des données réelles fournies par les capteurs et par les autres agents. Les agents horaires sont composés d'un réseau d'agents capteurs simulant l'influence des données. Ainsi, si l'agent $T + 1$ heure fournit de mauvais résultats (ce qui est vérifié une heure plus tard), il va demander à ces agents capteurs de modifier leur interdépendances afin d'obtenir un résultat plus proche de la réalité observée. Enfin, les prévisions de l'agent $T + 1$ heure vont servir d'entrées à l'agent $T + 2$ heures et ainsi de suite.

Les résultats proposés par Régis *et al.*, montrent l'efficacité d'une telle approche et surtout la robustesse et la flexibilité du système. En effet, aucun modèle mathématique pré-défini n'est utilisé contrairement aux approches classiques. Ici, le système multi-agent construit son propre modèle grâce aux différents poids affectés aux différents agents capteurs dans les agents horaires. Ceci permet d'installer le logiciel dans d'autres stations avec un minimum de difficulté d'installation – seulement une période de quelques secondes d'apprentissage automatique sur des données réelles enregistrées au préalable.

1.4.4 Autres applications des AMAS

La théorie des AMAS a aussi été appliquée dans d'autres domaines variés comme par exemple :

Le *routage adaptatif* de réseau téléphonique où les agents représentent les centres de communication et coopèrent afin de décharger au maximum le réseau [Dotto, 1996] ;

La *robotique collective* où les agents sont des robots essayant de transporter des ressources dans un environnement où la circulation est fortement contrainte [Picard et Gleizes, 2002] ;

Le *contrôle robotique* où le comportement d'un robot est simulé par un système multi-agent d'opérations élémentaires de composition de comportement [Capera, 2001] ;

La *conception de mécanismes* assistée par un AMAS dans lequel les composantes d'un mécanisme vont s'auto-organiser dans l'espace pour respecter la fonctionnalité (une trajectoire par exemple) et les contraintes posées par le concepteur [Capera *et al.*, 2004a] ;

La *programmation émergente* dans laquelle les instructions d'un programme vont s'auto-organiser afin d'obtenir la fonction attendue par le *néo-programmeur*, qui influera sur l'organisation par des commandes simples [Georgé, 2004] ;

Un *réseau de neuro-agents* qui est un réseau de neurones dans lequel les neurones sont des agents qui vont coopérer afin d'apprendre plutôt que d'utiliser des techniques classiques de rétro-propagation [Mano et Glize, 2004].

1.4.5 Analyse et constat

La conception de systèmes émergents soulève de nombreuses difficultés. En effet, la définition de l'émergence reste imprécise et repose sur des assertions permettant de dire *a posteriori* si un système est émergent ou non. Aucun critère pragmatique n'a jusqu'à présent été avancé afin de caractériser *a priori* et donc de spécifier des systèmes émergents (voir §1.1). Méthodologiquement parlant, cela pose un problème.

Cependant, nous avons vu que la nature nous offre des mécanismes au fonctionnement complexes mais aux spécifications simples, comme l'auto-organisation (voir §1.1.2). L'idée est donc que si un concepteur veut développer un système émergent, il devra mettre en place des mécanismes auto-organisateurs. La conception deviendra à la fois plus simple et plus complexe : plus simple, car les entités manipulées sont d'un moindre niveau d'abstraction que le système dans sa globalité, et donc les théories de micro-niveau seront plus simples à définir ; plus complexe, car les paramètres jouant sur le comportement de macro-niveau sont dissociés des comportements de micro-niveau. La gestion de cette complexité passe donc par le choix des paramètres les moins sensibles.

L'utilisation de la coopération comme critère d'auto-organisation (voir §1.3) est un moyen de manipuler ces paramètres, et de rendre le passage du micro au macro plus compréhensible ou "traçable". Néanmoins, les exemples d'applications présentés ci-dessus montrent l'intérêt de l'approche par la théorie des AMAS, mais soulèvent les problèmes d'identification des agents et de la définition des situations non coopératives.

Dans ABROSE (au niveau des TA) ou ANTS, les agents sont directement issus de l'analyse du domaine classique. Le concepteur peut identifier dès les premières phases de conception quelles entités du système doivent être modélisées en tant qu'agents grâce aux propriétés propres aux fourmis ou aux humains s'échangeant des services. Par contre, dans STAFF l'identification des tranches horaires comme agents est bien moins intuitive et a été définie par des concepteurs spécialistes des AMAS. De même, dans ABROSE, au niveau des BA, la décomposition des croyances et connaissances en systèmes multi-agents de termes est aussi un exemple d'identification difficile. Ainsi, la diffusion de la technologie AMAS passe par un effort conséquent en la direction des concepteurs en leur fournissant des méthodes d'identification des agents.

La définition de la coopération restant informelle, car complètement indépendante du domaine ou des techniques de programmation utilisées, la spécification des situations non coopératives, moteur de l'auto-organisation et garantes du bon fonctionnement du système, représente un problème épineux. Comment garantir l'exhaustivité des SNC spécifiées ? Comment garantir la complétude des compétences nécessaires à la coopération ? Comment s'assurer de la valeur des échanges liés à la coopération ? Là encore, un effort méthodologique doit nous pousser à proposer des outils et méthodes permettant une spécification plus complète.

Notre but est de fournir à des concepteurs novices dans cette technologie, une méthode de développement de systèmes adaptatifs, des notations et des outils associés afin de leur faciliter la tâche. Depuis, quelques années, le domaine des méthodes[2] orientées agent est en

[2]Les anglophones ont plutôt tendance à appeler les méthodes des *methodologies*. En français, nous conservons

plein essor et voit apparaître de nombreuses nouvelles méthodes dédiées à des problèmes particuliers. Nous allons étudier dans le chapitre suivant les méthodes existantes les plus pertinentes pouvant être une source d'inspiration pour une méthodologie de conception des systèmes auto-organisateurs.

Une telle méthode devra permettre de concevoir des systèmes multi-agents adaptatifs, ouverts et dynamiques, composés d'agents coopératifs. Ces notions ne sont pas forcément connues des développeurs non spécialistes. Un effort didactique semble nécessaire – en s'intégrant dans des contextes connus par exemple, et en proposant une aide à leur identification. De plus, l'auto-organisation par coopération repose sur les interactions entre les agents, il conviendra donc de se munir d'outils supportant cet aspect, mais aussi de contraindre l'utilisation des concepts et propriétés propres aux systèmes multi-agents, par la mise en place de processus précis et de règles de bon usage.

le terme *méthodologie* pour définir l'étude d'un ensemble de méthodes pour un domaine particulier – d'où le titre de ce mémoire.

2 Ingénierie orientée agent

LA communauté agent a beaucoup oeuvré dans le domaine des méthodes orientées agent depuis quelques années. En effet, malgré la résonance du paradigme multi-agent dans la communauté scientifique, la technologie n'est pas encore devenue un standard pour l'industrie. Nous pouvons attribuer ce défaut de reconnaissance à deux manques majeurs : la communauté multi-agent n'a pas encore montré la réelle utilité de l'approche par agent avec la résolution d'un problème significatif (ou *killer application*) ; la culture multi-agent n'a pas encore (ou pas assez en tout cas) développé les outils pédagogiques de communication vers les non spécialistes (étudiants et entreprises). Ceci inclut des livres de méthodes, des notations spécifiques et des outils facilitant la réalisation de systèmes.

Pour Booch, une *methodology*[1] est "un ensemble de méthodes appliquées tout au long du cycle de développement d'un logiciel, ces méthodes étant unifiées par une certaine approche philosophique générale" [Booch, 1992]. Une méthode de conception est constituée d'un processus de développement, de notations permettant la modélisation et d'outils d'aide à la conception, à la vérification et/ou au suivi du processus. Deux grands courants existent lorsque l'on traite des méthodes orientées agent. Les unes sont une suite logique des méthodes de conception objet s'inspirant des processus, des méthodes et des outils déjà existants. Les autres s'inspirent des modèles exhibés par les sciences du naturel, ce qui est compréhensible lorsque l'on considère les propriétés attribuées aux systèmes multi-agents (voir chapitre 1). Bien sûr, le premier courant possède un avantage certain dans la course à l'accès à l'industrie, car la technologie objet est à l'apogée de son utilisation en entreprise – du moins lors des phases de conception... La deuxième approche est plus risquée mais peut-être nécessaire afin de s'attaquer à la conception de systèmes aux fonctionnalités incomplètement spécifiées comme l'informatique diffuse. Néanmoins, ces deux courants principaux ne sont pas les seuls à avoir inspiré les méthodes orientées agent. D'autres approches, plus restreintes, comme l'ingénierie des connaissances ou des besoins, ont donné naissance à des méthodes plus spécifiques aux systèmes d'information. De même, certaines méthodes se sont ouvertement inspirées des domaines de la simulation biologique ou sociale comme cadre de développement.

Ce chapitre a pour but de montrer un état courant des méthodes orientées agent les plus représentatives. Les méthodes basées sur le paradigme objet sont nombreuses, et seulement certaines des plus citées dans la littérature seront exposées dans le paragraphe 2.1. Celles qui s'inspirent des phénomènes naturels, souvent les plus pertinentes pour résoudre

[1] Attention au distingo entre le terme anglais *methodology* qui détermine un ensemble de méthodes, voire une méthode, et le terme français *méthodologie* qui définie l'étude d'un ensemble de méthodes – ce que nous faisons actuellement.

les problèmes dans un environnement dynamique nécessitant une adaptation par auto-organisation, sont moins nombreuses et moins développées. Certaines d'entre-elles seront présentées dans le paragraphe 2.2. Bien sûr, d'autres alternatives existent et seront brièvement présentées dans la paragraphe 2.3. Enfin, à partir de ces présentations et d'une comparaison entre des méthodes existantes, dans le paragraphe 2.4, nous dresserons les besoins pour la conception de systèmes multi-agents adaptatifs par coopération et exprimerons nos choix quant au développement d'une méthode dédiée à de tels systèmes dans le paragraphe 2.5.

2.1 L'influence du paradigme objet

Les agents sont des entités autonomes, possédant des propriétés internes capables d'agir sur leur environnement. Il n'est pas rare, donc, de modéliser des agents par spécification orientée objet, compte tenu des points communs entre agents et objets. Bien sûr d'autres modélisations sont possibles, et souvent inspirées par la nature. De nombreuses méthodes orientées agent ont déjà vu le jour, mais ont toutes des disparités en terme de modèles d'agent, d'architecture, d'adaptation ou bien de facilité d'utilisation. Il convient donc de les survoler afin de déterminer quelles sont les notions et les méthodes exploitables afin de concevoir des systèmes multi-agents adaptatifs par coopération.

2.1.1 L'agent, évolution de l'objet ?

Van Dyke Parunak et Odell proposent d'expliquer l'évolution des approches de programmation, de la programmation monolithique à la programmation orientée agent, de la manière présentée dans le tableau 2.1. Dans la programmation monolithique, les unités de logiciel représentaient le programme dans sa globalité, entièrement spécifié par le programmeur et contrôlé par le système. La programmation modulaire répondit à l'augmentation en terme de complexité, de besoins de mémoire des applications, ainsi qu'au besoin de réutilisabilité. Par contre l'état des unités (modules) restait toujours sous contrôle externe par instructions d'appel. L'évolution logique fut la décomposition des programmes en objets. Ici, l'état des objets est encapsulé grâce à des méthodes d'accès ou d'instanciation. Par contre en programmation orientée objet classique, les objets sont considérés comme passifs et donc soumis au contrôle par envoi de messages et appel de méthodes. Enfin, afin de répondre aux besoins des agents, la programmation orientée agent, qui n'est pour l'instant qu'à l'état de concept, se propose de concevoir les agents comme des "objets actifs avec initiative" – un peu comme les acteurs en conception objet – qui ont leur propre "thread" de contrôle et sont mus par des règles, des buts ou des intentions [Van Dyke Parunak, 1997; Odell, 2002].

	Programmation monolithique	Programmation structurée/modulaire	Programmation orientée objet	Programmation orientée agent
Comment une unité se comporte-t-elle ? (Code de l'unité)	Non modulaire	Modulaire	Modulaire	Modulaire
Que fait une unité en court d'exécution? (Etat de l'unité)	Externe	Externe	Interne	Interne
Quand une unité s'exécute-t-elle? (Appel de l'unité)	Externe	Externe (appel)	Externe (message)	Interne (règles, buts)

Tableau 2.1 — L'évolution des approches de programmation : de la programmation monolithique à la programmation objet [Odell, 2002].

Odell soulève la question de l'autonomie des agents suivant deux axes : l'autonomie dynamique (de passif à pro-actif en passant par réactif) et l'autonomie non-déterministe (d'attendu à inattendu) [Odell, 2002]. Les objets peuvent être identifiés comme des entités passives au comportement attendu alors que des agents, comme des agents de courtage, possèdent un comportement inattendu entre le réactif et le pro-actif. Les systèmes naturels, comme les colonies de fourmis, se placent en haut des deux échelles : pro-actifs et inattendus.

Concernant les interactions entre agents comparées aux interactions objet à objet, les agents peuvent bien sûr faire appel à des méthodes au sens objet du terme, mais y ajoutent un contenu plus riche, comme par exemple avec les actes de langage FIPA ACL ou KQML ACL – il n'est pas rare d'ailleurs de trouver des librairies dédiées à ces actes dans les plates-formes de développement de systèmes multi-agents. Alors que les objets voient les autres objets comme des fournisseurs de services par appel de méthodes, les notions de conversations, de communications à long terme et de réseaux de partenaires sont aussi des notions importantes en programmation orientée agent. En effet, les agents peuvent construire des réseaux de contacts et donc posséder des capacités cognitives correspondantes comme une mémoire ou des croyances sur les autres agents.

Si nous considérons les agents comme une évolution des objets, voici les différences conceptuelles notables à prendre en compte, par exemple, lors de l'identification des agents dans un système :

Décentralisation : les objets sont vus classiquement comme des unités au contrôle centralisé – bien qu'ils puissent implanter des systèmes distribués. Les agents, au contraire, sont par natures autonomes en décision et donc aucun contrôle global ne peut s'exercer sur eux ;

Classification multiple et dynamique : une fois créé par un autre objet, un objet ne peut changer de classe ou d'héritage. Si l'on pousse le concept d'agent logiciel plus loin, il devient possible pour des agents de muter et de changer de classe ;

Caractéristiques au niveau des instances : les attributs et méthodes possédés par les objets sont définis par leur classe. Dans le cas des agents, ils ne sont pas définis au niveau des classes mais au niveau des instances de classe ;

Faible en impact et en portée : les objets ou les agents peuvent être de faible ou de grosse granularité. Par contre, les objets ont souvent plus d'impact sur le système dans sa globalité qie les agents qui n'ont des actions qu'au niveau des autres agents et non du système [Van Dyke Parunak, 1997]. Cette idée rejoint le principe de localité apparaissant dans la définition d'un agent [Ferber, 1995] ;

Analogie à la nature : les caractéristiques d'autonomie dynamique et de non-déterminisme rendent les agents plus proches des systèmes naturels que n'en sont les objets. La notion de système ouvert avec apparition et disparition d'agents, va aussi dans ce sens.

En définitive, les agents peuvent être vus comme des objets ayant des règles structurelles ou comportementales supplémentaires – définies par des stéréotypes ou des extensions du métamodèle. La plupart des concepts manipulés par la communauté agent peuvent être modélisés en objet (ontologie, dynamique d'état, ...) mais certains axes sont encore peu explorés. Il semble aussi que la caractéristique d'autonomie (qu'elle soit dynamique ou non-

déterministe) d'un agent reste le point difficile de la conception d'agent – comme cela l'est toujours dans le domaine de l'intelligence artificielle. Le problème étant de trouver le bon niveau d'abstraction permettant de qualifier un agent d'autonome ou non. En effet, un agent reste toujours dépendant du système d'exploitation ou de la plate-forme sur laquelle il s'exécute, mais pas vis-à-vis des autres agents ou des objets qui l'entourent.

2.1.2 Le processus

A partir du moment où des concepteurs ont choisi les objets comme paradigme de programmation de systèmes multi-agents, il semble naturel que les processus de développement orientés agent s'inspire directement ou indirectement de processus définis par la communauté objet, comme le *Rational Unified Process* (ou RUP) par exemple [Jacobson *et al.*, 1999]. Il est possible d'identifier cinq phases majeures dans les processus de développement de systèmes multi-agents : l'analyse des besoins, la conception de l'architecture (ou analyse), la conception détaillée, le développement (ou implémentation) et le déploiement [Ricordel et Demazeau, 2000; Arlabosse *et al.*, 2004]. Bien sûr, de nombreuses variations existent et le nombre de phases peut varier grandement, en fonction de l'axe de prédilection de la méthode. Par exemple, Aalaadin ne définit que trois phases qui sont l'analyse, la conception et la réalisation [Ferber et Gutknecht, 1998].

2.1.3 Les notations et langages de spécification orientés objet

Compte-tenu de la diversité des modèles d'agent et de systèmes multi-agents, l'ingénierie orientée agent dispose de nombreuses notations et langages de spécification. Encore une fois, l'influence de l'objet sur les notations multi-agents est importante. Une première école utilise les avancées dans le métamodèle UML[2] et mène de gros efforts de standardisation des concepts agents, au sein de la FIPA[3], par exemple.

2.1.3.1 UML et ses extensions

Odell *et al.* ont proposé une contribution importante : l'extension de la notation UML, appelée A-UML[4] (pour *Agent-UML*) [Odell *et al.*, 2000, 2001]. Ce travail porte essentiellement sur les protocoles d'interactions entre agents (ou AIP[5]) principalement sur trois niveaux de définition. Le premier niveau concerne un protocole dans son ensemble par utilisation de *templates* génériques produisant des paquetages pour protocoles spécifiques. Le deuxième niveau représente les interactions entre agents grâce à des diagrammes d'interactions : diagrammes de séquences modifiés (voir figure 2.1), diagrammes de collaborations, diagrammes d'activités ou bien diagrammes d'états/transitions. Odell *et al.* ont ajouté des types de branchements différents dans les diagrammes de séquence afin de prendre en compte l'indéterminisme du comportement d'un agent – la figure 2.1 montre plusieurs

[2] *Unified Modeling Language* – `http://www.uml.org`
[3] *Foundation for Intelligent Physical Agents* – `http://www.fipa.org`
[4] *Agent-UML* – `http://www.auml.org/`
[5] *Agent Interaction Protocol*

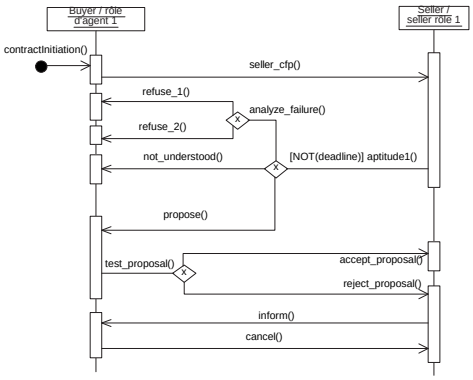

Figure 2.1 — Représentation d'un protocole d'interaction grâce à un diagramme de séquence A-UML.

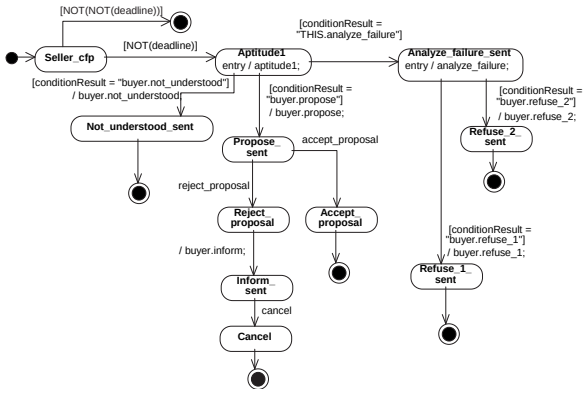

Figure 2.2 — Représentation de l'état interne d'un agent lors d'un protocole d'interaction grâce à un diagramme d'états/transitions.

branchements XOR (OU exclusif). Le troisième niveau de représentation correspond au processus internes aux agents qui sont principalement modélisés par des diagrammes d'états/-transitions (voir figure 2.2).

A-UML prend en compte d'autres considérations telle la spécification des rôles dans les différents diagrammes, comme dans la figure 2.1 où les classes du diagramme de séquences générique sont annotées Classe/Rôle. Par contre des notions importantes pour les systèmes multi-agents sont définies de manière trop floue pour être exploitables. Par exemple, les auteurs proposent un stéréotype «emergence» pour indiquer une possibilité d'émergence dans un diagramme de classes ; mais il semble difficile de définir les règles de bonne utilisation nécessaires à ce stéréotype.

Une autre extension d'UML est la notation AORML (pour *Agent-Object Relationship Modeling Language*) proposée par Wagner, qui est un profil UML pour l'analyse et la conception de systèmes d'information organisationnels basés sur les agents [Wagner, 2002, 2003]. Deux types de modèles sont définis dans AOR : externes et internes. Les modèles externes présentent une perspective d'observateur externe au système, observant des agents et leurs interactions grâce à des éléments comme ceux apparaissant dans la figure 2.3. Plusieurs types de diagrammes permettent de définir les relations entre agents et objets, dans une vision orientée système d'informations : les diagrammes d'agents, les diagrammes de cadre interaction, les diagrammes de séquence d'interaction et les diagrammes de schéma d'interaction. Les modèles internes sont le moyen de définir les vues qu'un agent a sur son environnement. Les actions et comportements des agents, dans la vue interne, sont décrits grâce à des réseaux de Pétri. Par contre, AOR ne donne aucun outil concernant l'auto-organisation ou la coopération, étant donnée son orientation vers les systèmes d'informations.

Contrairement aux notations A-UML et AORML qui ne sont associées à aucune méthode, la notation MESSAGE/UML, qui est un profil UML, a été spécialement développée pour les besoins méthodologiques soulevés par MESSAGE [Caire *et al.*, 2001] qui sera décrite dans le paragraphe 2.4. Pour résumer, pour modéliser des problématiques multi-agents à partir d'UML, deux possibilités d'extensions non exclusives existent :

– modifier la notation en créant un profil comme AOR [Wagner, 2002] ou MESSAGE [Caire *et al.*, 2001] par exemple ;

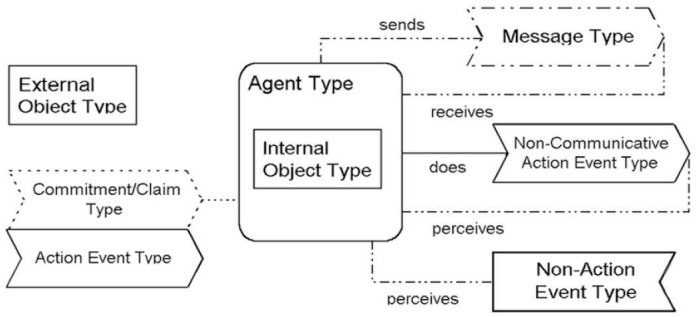

Figure 2.3 — Les différents éléments de modélisation externe d'AOR.

– ajouter du sens aux éléments existants en créant des stéréotypes comme A-UML, par exemple [Odell *et al.*, 2000].

2.1.3.2 Autres notations graphiques

Bien sûr, UML est un standard mais pas une obligation. Certaines approches ont proposé des notations proches de la modélisation objet, mais ayant pour but de décrire une problématique purement multi-agent, comme, par exemple, le langage de description du modèle AGR d'Aalaadin [Ferber et Gutknecht, 1998].

Une autre notation graphique notable est celle utilisée dans OPM/MAS [Sturm *et al.*, 2003]. Cette notation décrit les relations entre objets et processus sur le même plan. Elle est assez complète, voire même trop. En effet, même les principaux utilisateurs et développeurs avouent la trouver un peu trop complexe[6] : plus d'une vingtaine de types de flèches, de noeuds ou de liens ayant des sens différents ou proches parfois existent. Ceci a pour conséquence un langage d'une expressivité impressionnante mais très difficilement manipulable, de manière complète.

2.2 L'apport du naturel

L'ingénierie logicielle orientée objet n'est pas la seule base de travail de spécification utilisée dans le domaine agent. En effet, compte tenu des problématiques soulevées par les systèmes multi-agents, les sciences naturelles ont apporté leurs connaissances et modèles. Ainsi, un axe de l'ingénierie multi-agent se focalise sur des modèles issus des sciences naturelles ou sociales comme l'illustre le groupe de travail ESOA[7] [Di Marzo Serugendo *et al.*, 2004]. Ici, la spécification consiste plus en un effort de correspondance entre les entités du système à définir et les modèles pré-donnés, aux paramètres et constantes près (comme la vitesse d'évaporation de phéromones dans le cas des fourmis).

[6] Aveux recueillis auprès de Onn Shehory lors d'un tutoriel sur les méthodes pour l'ingénierie orientée agent à AAMAS 2004.

[7] *Engineering Self-Organising Applications*

2.2.1 Les animaux sociaux

Les éthologues ont fourni de nombreux modèles de sociétés d'animaux facilement exploitables dans les systèmes multi-agents. L'exemple des fourmis, ou des *ant algorithms*, est souvent repris dans la littérature car significatif de la facilité d'implantation et de la complexité des tâches réalisables [Deneubourg *et al.*, 1990; Colorni *et al.*, 1992; Dorigo *et al.*, 1996; Bonabeau *et al.*, 1997; Van Dyke Parunak, 1997; Bonabeau *et al.*, 1999; Topin *et al.*, 1999a]. Deux comportements collectifs ont été principalement étudiés chez les fourmis : l'optimisation de trajectoire et le tri des cadavres.

Le premier comportement permet aux fourmis de construire, autour de leur nid, un réseau de phéromones vers les sources de nourriture. La théorie des graphes propose bien sûr des solutions pour trouver le plus court chemin, mais de manière totalement centralisée. Chez les fourmis le comportement est totalement partagé entre les fourmis, ce qui en fait une solution pour la recherche de chemin dans des réseaux informatique totalement distribués par nature. L'ingénierie peut se résumer en deux points : (*i*) identifier les entités jouant les rôles de fourmis (par exemple des agents) ou de phéromones (les informations transportées) puis (*ii*) trouver les paramètres impliquant un comportement cohérent (facteur d'évaporation ou d'attraction des phéromones).

Le deuxième comportement collectif des fourmis concerne l'agrégation des cadavres en paquets homogènes dans les nids de certaines fourmis. Ici encore, l'informatique théorique a déjà fourni des solutions d'algorithmes de tris, mais aucun n'est exploitable de manière décentralisée. Ainsi, un domaine d'application de ce modèle est la maintenance de bases de données distribuées.

Les termites sont aussi des insectes sociaux exhibant des comportements collectifs exploitables en informatique. Les termites ont en effet la faculté de construire des nids à l'architecture complexe, avec des ponts et des arches, de manière totalement décentralisée – il n'y a pas d'architecture ou de maître d'ouvrage – avec un mécanisme phéromonal similaire à celui des fourmis [Van Dyke Parunak, 1997]. Les guêpes, elles, adoptent des comportements de différenciation de tâches, avec apparition de chefs, de fourrageuses ou de nurses, alors qu'elles sont toutes génétiquement identiques. Ici, les mécanismes mis en jeux sont purement locaux. Le choix des tâches se fait lors de rencontres entre deux guêpes et est non décidé par la reine, mais plutôt influencé par trois paramètres : la force d'une guêpe (sa mobilité), le seuil de fourragement (l'envie d'aller chercher de la nourriture) et la demande des larves en nourriture. A partir de ces trois paramètres, le comportement des guêpes peut être modélisé stochastiquement grâce à des fonctions de Fermi [Van Dyke Parunak, 1997]. Construire un système artificiel inspiré de ce modèle consistera encore une fois à trouver les bons paramètres.

Dans le même ordre d'idée, Bourjot *et al.* proposent une transposition du comportement social de certaines espèces d'araignées pour la détection de contours dans des images [Bourjot *et al.*, 2002]. Les araignées virtuelles utilisent la stigmergie et les toiles de soies comme support de détection.

Les volées d'oiseaux ou les bancs de poissons ont aussi des comportements collectifs souvent étudiés, appelés *flocking behaviors*. Les poissons ou les oiseaux restent groupés, se

coordonnent, et évitent les collisions, sans toutefois avoir de contrôle central. Les oiseaux ou les poissons observent pourtant trois règles simples : rester à une certaine distance des objets ou des voisins, ajuster sa vitesse à celle des voisins et rester près du centre du groupe. Les problèmes de trafic aérien ou routier sont souvent affichés comme des applications possibles de tels comportements.

Les meutes de loups sont, elles, capables d'entourer une proie sans coordination supervisée. Ce phénomène est à l'origine d'un des problème clé de l'intelligence artificielle distribuée : le problème "proie-prédateur". Les paramètres en jeu sont la vitesse des prédateurs et de la proie, et la répulsion entre prédateurs [Manela et Campbell, 1995].

Tous ces modèles ne sont utilisables que dans des applications très spécifiques : tri, routage, contrôle aérien, ... Pourtant, un effort de généralisation est souvent observé, comme avec Swarm LINDA par exemple, qui est une méthode essayant de permettre l'instanciation des *ant algorithms* à des problèmes divers – mais faisant souvent appel à des agents mobiles – en utilisant le concept des tuples [Tolksdorf et Menezes, 2004].

2.2.2 Les réseaux neuronaux de Kohonen

Les réseaux neuronaux de Kohonen, appelés aussi cartes auto-adaptatives, sont une métaphore de la nature dans laquelle des neurones formels vont s'auto-organiser pour fournir la fonction demandée par l'utilisateur [Kohonen, 2001]. Le principe de ces réseaux est issu de l'observation, dans certaines zones du cerveau, d'une corrélation entre la proximité fonctionnelle et la proximité géographique des objets perçus. En général, l'algorithme permet d'assurer que les données projetées dans un même voisinage sont des données proches.

Tous les neurones de la carte sont interconnectés, mais avec des poids variant selon la distance entre deux neurones. Plus deux neurones sont proches, plus leurs poids sont élevés. Le fonctionnement est simple : on présente un objet en entrée du réseau, et le réseau active en sortie un neurone particulier en fonction de la classe de l'objet. Ces systèmes sont donc principalement utilisés pour des applications de *clustering* (regroupement), de topologie ou de classement de données, comme les algorithmes basés sur les comportements d'agrégations de cadavres de fourmis. L'avantage de cette approche est que l'apprentissage et l'adaptation de ces réseaux est non-supervisé ni guidé par une fonction théorique. Ici, les neurones apprennent à partir du neurone dont la configuration est la plus proche de l'objet présenté. De proche en proche, les neurones vont apprendre par une dispersion pondérée par une "différence de gaussiennes". Par contre, il n'est pas possible de traiter la dynamique de leur environnement : ils ne traitent que des données statiques. De plus, dans les cartes de Kohonen classiques, il peut se produire que des données proches soient projetées dans des zones non voisines de la grille de représentation.

Méthodologiquement parlant, les cartes de Kohonen sont intéressantes dans le sens où ce sont des systèmes auto-organisés non supervisés. Dans ce cas, l'ingénierie consiste à trouver les classes de données à ranger, ou bien la topologie à souligner. Par exemple, la méthode WEBSOM a été définie afin de faciliter l'organisation de documents textuels [Lagus *et al.*, 1999].

2.2.3 Les algorithmes génétiques

Au lieu de s'inspirer des mécanismes au sein d'une même population d'individus permettant d'obtenir un comportement collectif cohérent, le calcul évolutionnaire s'intéresse aux mécanismes d'évolution et d'adaptation des espèces de génération en génération. Le calcul évolutionnaire repose, en premier lieu, sur les théories Darwiniennes de l'évolution des espèces par des mécanismes d'héritage génétique et de sélection naturelle. Plusieurs courants parallèles ont vu le jour, depuis les années soixante, comme les algorithmes génétiques [Holland, 1992; De Jong, 1975; Goldberg, 1989], se focalisant sur les processus d'adaptation, la programmation évolutionnaire [Fogel, 1966] ou les stratégies évolutionnaires [Rechenberg, 1973] affichés comme des solutions d'optimisation.

A partir d'une population initiale d'individus, un algorithme génétique qui converge vers un état d'optimum global pour le problème traité, va explorer de manière dynamique l'espace de recherche de manière stochastique en effectuant des croisements entre individus et des mutations. Les individus sont les points de l'espace d'état qui vont être évalués par une fonction d'adaptation, ou *fitness*. L'algorithme 2.1 montre le fonctionnement global d'un algorithme génétique générique à la Holland. Les phases clés sont :

1. l'*évaluation* (ligne 5) lors de laquelle tous les individus de la population sont notés grâce à la fonction de *fitness* ;
2. la *sélection* (ligne 10) lors de laquelle des parents sont sélectionnés en fonction de leur évaluation ; les "meilleurs parents" (les plus adaptés) sont choisis de préférence ;
3. la *reproduction* (lignes 12 à 16) lors de laquelle des individus de la prochaine génération sont créés par croisements (*crossover*) de certains gènes (bits) d'individus de la génération en cours avec une probabilité p_c, et par mutation de certains d'entre eux ,avec une probabilité p_m appelée facteur de mutation ;
4. le *remplacement* (linge 20) lors duquel la nouvelle génération devient la génération en cours ;
5. l'*arrêt* : l'algorithme s'arrête lorsqu'un certain critère est atteint : au bout de n générations, le meilleur élément de la population ne s'est pas amélioré pendant les x dernières générations, ou bien la différence de valuation entre les deux meilleurs individus est inférieur à un seuil ϵ.

La conception de systèmes multi-agents peut, bien sûr, s'inspirer de ces techniques, en générant par exemple le comportement des agents grâce à un algorithme génétique pour obtenir une population la plus adaptée. Pourtant, comme pour l'application de modèles inspirés des animaux sociaux, un algorithme génétique est un exemple de non linéarité : une infime variation dans les paramètres (nombre de générations, p_m, p_c, ϵ, ...) peut produire des résultats extrêmement distants. Trouver ces paramètres est donc un problème critique. Le deuxième défaut des algorithmes génétiques est la fonction d'adaptation qui est elle aussi souvent difficile à définir, non pas parce qu'elle est éloignée conceptuellement de la tâche à réaliser, mais parce qu'il devient facile de tomber dans des attracteurs locaux et de bloquer ainsi la recherche dans l'espace – bien que des techniques, comme l'injection d'individus totalement nouveaux à chaque génération (*sharing*), atténuent ces problèmes dans certaines circonstances. Un troisième défaut est le temps imparti à l'évaluation et sa nécessaire centralisation. En effet, évaluer un individu revient à lui faire exécuter la totalité de son

Données : Un nombre n de générations de *taille* individus, une fonction d'adaptation f, une probabilité de croisement p_c et une probabilité de mutation p_m
Résultat : Le meilleur individu de la dernière génération

```
1  début
2      //C'est la génération d'un ensemble de génomes (individus)
3      // -> création d'une chaîne de bits au hasard
4      initialisation (population, taille);
5      pour n générations faire
6          //chaque individu est évalué -> donne un nombre réel
7          tEval := évaluation (population);
8          //création d'une population vide
9          nouvellePopulation := création (taille);
10         pour (taille/2) individus faire
11             //Choisir en fonction de l'adaptation deux individus parents
12             ⟨m, p⟩ := sélection (population, f);
13             //Mélanger et croiser les génomes pour obtenir deux enfants
14             ⟨enfant1, enfant2⟩ := croisement (⟨m, p⟩, pc);
15             //Muter les deux individus enfants
16             mutation (enfant1, pm);
17             mutation (enfant2, pm);
18             //Ajouter ces individus à la nouvelle population
19             nouvellePopulation := nouvellePopulation ∪ {enfant1, enfant2}
20         fin
21         population := nouvellePopulation
22     fin
23 fin
```

Algorithme 2.1 — Un exemple d'algorithme génétique

comportement afin de déterminer s'il est adapté ou non – et ceci doit être fait pour tous les individus. Par exemple, dans le cas de la robotique collective, cela revient à simuler à chaque génération le comportement de tous les robots et d'observer le résultat. Ceci, implique donc la quasi-impossibilité d'en faire un processus en temps en réel. Enfin, un dernier défaut est celui du codage des individus en gênes qui n'est souvent pas aussi immédiat qu'une simple traduction bit à bit dans des applications réelles.

Finalement, l'ingénierie évolutionnaire de systèmes multi-agents se résume donc en une phase de conception de l'algorithme (codage des agents, détermination des paramètres), une phase de création et de simulation des individus (exécution de l'algorithme) qui revient à une phase d'apprentissage dans d'autres systèmes, et une phase d'implantation dans le système réel.

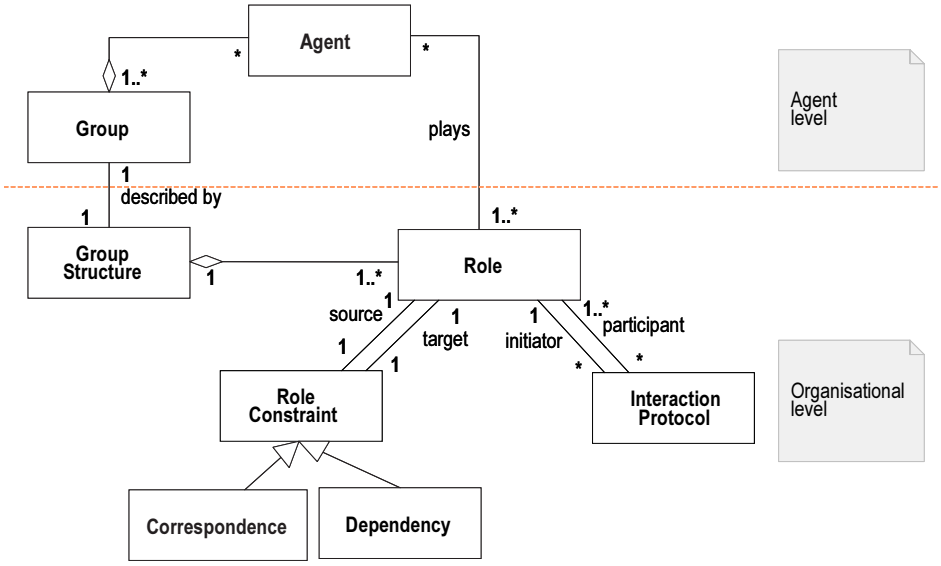

Figure 2.4 — Le métamodèle UML d'AGR

2.2.4 Modèles sociaux et organisationnels

Sont apparues comme pertinentes, dans la problématique de la conception de systèmes multi-agents, les notions organisationnelles de groupe, de rôle ou de collaboration issues de l'observation de sociétés d'individus. Ainsi, lors de la conception d'un système, l'un des points clés est de définir l'organisation du système en terme de rôles à remplir dans des groupes [Ferber et Gutknecht, 1998]. Ici, l'organisation n'est pas un résultat émergent de l'activité du système, mais un moyen pour lui d'atteindre son but. Le modèle AGR (Agent/-Groupe/Rôle) est défini comme dans la figure 2.4 [Ferber *et al.*, 2003]. Un agent fait partie d'un groupe et y joue un certain rôle qui représente sa fonction abstraite. Dans la méthode Gaia, l'organisation est vue comme une collection de rôles. Chaque rôle possède des permissions et des responsabilités en fonction de sa place dans l'organisation, afin d'assurer certaines propriétés de sécurité et de bon fonctionnement [Wooldridge *et al.*, 2000]. Les agents, qui sont dans Gaia des agents de forte granularité, doivent remplir un rôle dans l'organisation qui est défini *a priori*. De la même manière, DESIRE définit des tâches à remplir, qui peuvent être assimilées à des rôles dans Gaia ou Aalaadin [Brazier *et al.*, 2000]. La méthode Voyelles de Demazeau identifie cinq axes de modélisation multi-agent, dont l'organisation qui est vue comme la contrainte structurante du système multi-agent [Demazeau, 2001b].

Les approches centrées organisation permettent de fixer des règles, des normes aux groupes ou des contraintes aux rôles, et ainsi garantir le bon fonctionnement des organisations. Le contrecoup d'une telle rigidité organisationnelle est que la gestion de collectifs auto-organisés dans lesquels les fonctions et notions de groupes sont émergentes est impossible. Il faut alors permettre la focalisation sur d'autres axes de modélisation comme dans Voyelles par exemple.

Un autre domaine d'inspiration est la simulation sociale qui concerne la modélisation et la simulation sur ordinateur de phénomènes sociaux. De nombreux modèles sociaux de comportements auto-organisateurs ont été explorés par cette discipline. Parmi ces modèles

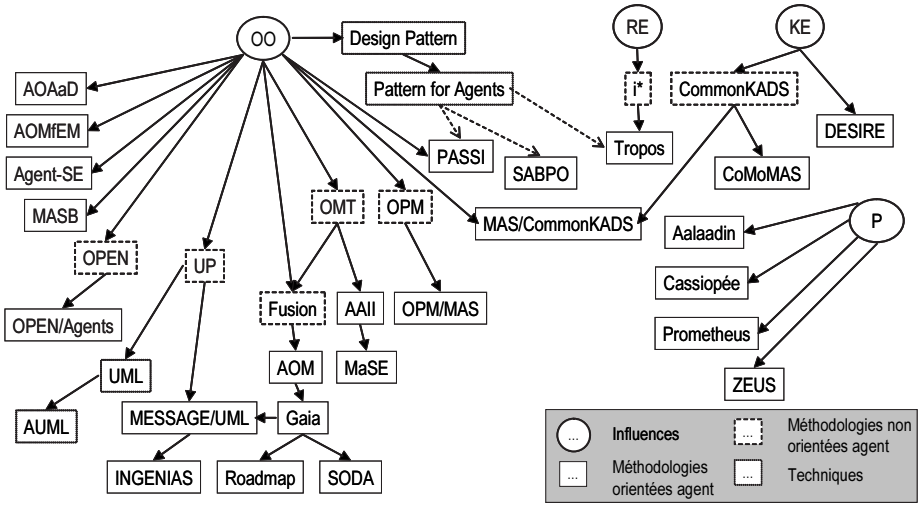

Figure 2.5 — Généalogie des méthodes orientées agent et les différentes influences

il y a, par exemple, les modèles basés sur les étiquettes (ou *tags*) qui furent d'abord initiés par Holland puis poursuivis par Hales et Edmonds qui présentent les modèles basés sur les étiquettes comme de réels outils de conception de systèmes multi-agents auto-organisateurs [Holland, 1993; Hales et Edmonds, 2003a]. "Les étiquettes sont des marquages ou des indicateurs attachés aux agents qui peuvent être observés par les autres agents. Les étiquettes peuvent évoluer ou changer dans le même sens que les comportements peuvent évoluer ou changer (pour des agents adaptatifs)". Les étiquettes sont en fait des données sociales communes comme, par exemple, l'expression d'un contentement lors de transactions. Ces informations permettent d'augmenter le taux de coopération dans des systèmes de distribution de tâches, sans planification globale. Hales et Edmonds définissent alors un principe de rationalité sociale : "Si un agent socialement rationnel peut exécuter une action dont le bénéfice joint est plus grand que la perte jointe, alors il sélectionnera cette action" [Hales et Edmonds, 2003b]. Ceci implique de définir, lors de la conception, une mesure du bénéfice des agents lors d'interaction, ce qui n'est pas forcément chose facile. C'est pour cela que la plupart des applications de ces techniques sont axées sur le commerce *peer-to-peer*, ou la gestion des marchés électroniques, dans lesquels la mesure du bénéfice est directe.

2.3 Alternatives et autres sources d'influences

Comme le montre la figure 2.5, l'ingénierie objet (OO) est la principale influence pour les méthodes orientées agent. Cependant, trois autres courants existent : l'ingénierie des connaissance (KE), l'ingénierie des besoins (RE) et une approche orientée plate-forme (P).

2.3.1 Ingénierie des connaissances

Plusieurs méthodes pour le développement des systèmes multi-agents ont été proposées en partant des celles qui ont été dédiées à la modélisation des systèmes à base de connaissances et experts, appelée *ingénierie des connaissances*. Par exemple, la méthode MAS-

CommonKADS de Iglesias *et al.* est une extension de CommonKADS qui ajoute des techniques venant des méthodes orientées objet à des techniques de conception des protocoles dans le but de modéliser les agents et les interactions entre agents [Iglesias *et al.*, 1998]. L'extension à CommonKADS comprend les modèles suivants :

- Le *modèle d'agent* qui décrit les caractéristiques principales des agents, y compris les capacités de raisonnement, les habiletés, les détecteurs et les effecteurs, les services, les buts ;
- Le *modèle de tâche* qui décrit les tâches (les buts) devant être exécutés par les agents, et la décomposition des tâches, en utilisant les spécimens textuels et les diagrammes ;
- Le *modèle d'expertise* qui décrit les connaissances nécessaires aux agents pour exécuter les tâches. La structure de connaissances suit l'approche de KADS, et distingue les connaissances de domaine, les connaissances de la tâche, les inférences et les connaissances spécifiques au problème à résoudre ;
- Le *modèle de coordination* qui décrit les conversations entre les agents, c'est-à-dire leurs interactions, leurs protocoles et les capacités d'interaction exigées. Les interactions sont modélisées en utilisant les techniques de description formelle MSC (Message Sequence Carts) et SDL (Specifications and Description Language) ;
- Le *modèle d'organisation* qui décrit l'organisation dans laquelle le système multi-agent sera introduit ainsi que l'organisation de la société d'agents. La description de la société d'agents utilise une extension du modèle d'objets d'OMT (*Object Modeling Technique*), et décrit la hiérarchie des agents, le rapport entre les agents et leur environnement, et la structure de la société d'agents ;
- Le *modèle de communication* qui détaille les interactions entre l'agent humain et le logiciel, et les facteurs humains nécessaires au développement de ces interfaces utilisateur ;
- Le *modèle de conception* qui recueille les modèles précédents et est subdivisé en trois sous-modèles : la conception de l'application, la conception de l'architecture et la sélection de la plate-forme de développement pour chaque architecture d'agent.

MAS-CommonKADS a été plusieurs fois exploitée dans la conception de systèmes de gestion de réseaux ou de systèmes multi-experts. Souvent, les agents déployés sont à forte granularité compte-tenu de l'approche adoptée orientée connaissances.

2.3.2 Ingénierie des besoins

L'*ingénierie des besoins*, quant à elle, fournit des méthodes, des techniques, et des outils permettant de développer et d'implanter des systèmes informatiques fournissant les services et les informations attendus par leurs utilisateurs, exigés par leurs acquéreurs, et qui soient compatibles avec leur environnement de fonctionnement. Bien que cette problématique soit non directement connectée à la notion de système multi-agent, certains travaux se focalisant sur des agents logiciels communicants fournisseurs de service pour des utilisateurs humains se sont fortement inspirés de cette branche de l'ingénierie des systèmes.

Par exemple, la notation $i^\star$ de Yu (voir figure 2.6) se focalise sur l'ingénierie des besoins centrée sur les caractéristiques intentionnelles des agents (leurs buts) [Yu et Mylopoulos, 1994]. Cette notation a notamment été adoptée par la méthode TROPOS [Castro *et al.*, 2001].

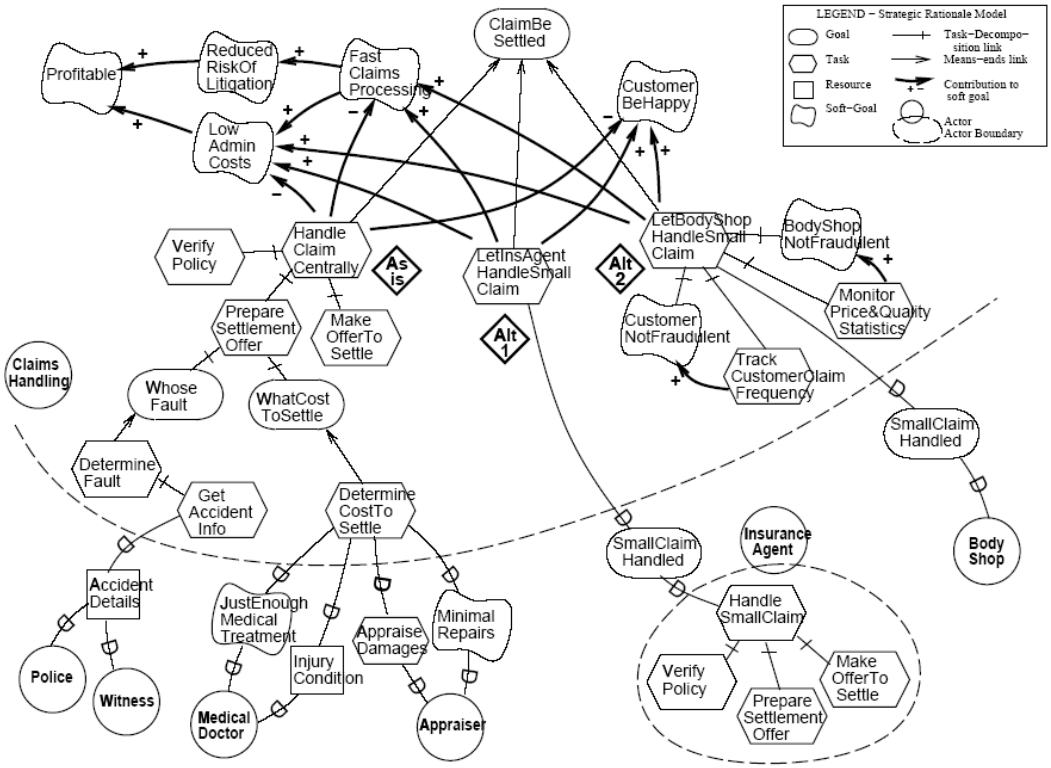

Figure 2.6 — Un exemple de modélisation orientée intention avec *i** [Castro *et al.*, 2001].

Les principaux éléments utilisés par *i** sont :

- l'*acteur* qui est un agent social organisationnel, humain ou logiciel ;
- le *rôle* qui une caractérisation abstraite du comportement d'un acteur dans un contexte particulier ;
- la *position* qui est un ensemble de rôles joués par un acteur ;
- le *but* qui est l'intérêt stratégique d'un acteur qui peut être *soft* (secondaire) ou *hard* (primordial ou de fonctionnement) ;
- le *plan* qui est une façon de satisfaire un but ;
- la *ressource* qui est une entité physique ou informationnelle dont a besoin un acteur pour fonctionner ;
- la *dépendance* sociale entres acteurs où un acteur (*depender*) dépend d'un autre (*dependee*) pour atteindre un but, exécuter un plan ou obtenir une ressource.

Ce langage, malgré les nombreux concepts manipulés, est graphique et facilement compréhensible ; bien que les schémas puissent devenir facilement illisibles pour des applications réelles comme le montre la figure 2.6. Malgré tout, ce langage reste seulement un langage d'expression des besoins – non forcément fonctionnels – et c'est notamment pour cela que TROPOS utilise d'autres notations comme UML pour décrire d'autres aspects des systèmes en construction.

2.3.3 Les plates-formes multi-agents et de simulation

Plusieurs types de plates-formes multi-agents existent : les plates-formes de simulation, les plates-formes de développement et les plates-formes d'exécution [Arlabosse *et al.*, 2004]. Elles ont été un courant notable d'influence pour les méthodes orientées agent. Souvent, les enjeux de conception sont la conséquence des techniques utilisées par les plate-formes. Par exemple, Zeus qui est principalement une plate-forme de simulation, a souvent été citée comme source d'inspiration [Nwana *et al.*, 1999], mais reste pourtant très limitée par les choix de modélisation portant uniquement sur des systèmes économiques purs.

D'autres méthodes se sont adaptées aux plates-formes d'exécution. Par exemple, Prometheus, de Padgham et Winikoff, est une méthode de développement d'agents BDI[8], dont les concepteurs affichent clairement leur souhait d'utiliser la plate-forme Jack[9] comme environnement d'exécution [Padgham et Winikoff, 2003]. Ainsi, les choix de modélisation et de spécification sont directement influencés par les contraintes techniques imposées par Jack.

Drogoul et Collinot ont développé la méthode Cassiopée afin de modéliser des systèmes multi-agents indépendamment de tout choix d'architecture d'agent ou de plate-forme d'exécution. Néanmoins, cette méthode est fortement inspirée du domaine de la simulation orientée agent/multi-agent et est le fruit d'une certain expertise dans ce domaine [Collinot et Drogoul, 1996, 1998; Drogoul, 2000].

2.4 Méthodologie et comparaison de méthodes

Comme les méthodes de développement de systèmes multi-agents sont nombreuses et évoluent rapidement, il est nécessaire de faire une comparaison de l'existant afin d'exprimer nos besoins compte-tenu de nos objectifs : concevoir des systèmes auto-organisateurs dans le cadre de la théorie des AMAS, intégrer la méthode dans un contexte connu des ingénieurs, et fournir des outils d'aide à la conception et à la décision.

2.4.1 Les cadres de comparaisons existants et problématique

La problématique de la comparaison de méthodes a été souvent soulevée dans la littérature pour aider le concepteur à choisir la méthode la plus adaptée au problème donné ou pour exhiber les différences essentielles entre les différentes méthodes.

Shehory et Sturm ou bien Dam et Winikoff comparent les méthodes suivant quatre ou cinq axes : les concepts manipulés, les notations utilisées (et les techniques de modélisation), le processus de développement et la pragmatique [Shehory et Sturm, 2001; Sturm et Shehory, 2003; Dam et Winikoff, 2003]. Dam et Winikoff ont même fait tester trois méthodes (MaSE, Prometheus et Tropos) par des stagiaires afin de les comparer suivant ce schéma. Cernuzzi et Rossi se focalisent seulement que sur les concepts de modélisation des agents (les rôles, les buts, l'organisation, etc.), mais proposent une évaluation chiffrée par une somme pondérée des différentes prises en compte des concepts par les méthodes [Cernuzzi et Rossi, 2002].

[8] Bien qu'aujourd'hui Prometheus s'ouvre à d'autres architectures et revendique son caractère générique.
[9] `http ://www.agent-software.com/shared/products/index.html`

Cette évaluation nécessite, par contre, une grande expertise dans toutes les méthodes et un grand nombre d'évaluateurs afin de déterminer au mieux les coefficients. Dans [Bernon *et al.*, 2002b], nous avions comparé des méthodes suivant huit axes : l'étendu de la couverture du processus, la spécialisation de la méthode à une application, l'architecture d'agent sous-jacente, l'utilisation de notations existantes, le domaine d'application (dynamique ou non), le modèle de rôle, le modèle de l'environnement et l'identification des agents.

Compte-tenu du large éventail de méthodes à comparer et de notre volonté de s'intégrer dans des processus existant, la comparaison qui va suivre sera faite suivant les quatre axes de Dam et Winikoff, mais en gardant en sous-catégorie les points de Bernon *et al.* qui rejoignent nos objectifs.

2.4.1.1 Les concepts clés et propriétés

Afin d'évaluer une méthode, il est intéressant de vérifier quels concepts agents et multi-agents sont pris en compte [Shehory et Sturm, 2001; Sturm et Shehory, 2003; Dam et Winikoff, 2003; Cernuzzi et Rossi, 2002]. Nous avons vu quels étaient ces concepts, et les propriétés des systèmes multi-agents dans le paragraphe 1.2. Pour plus d'information, Boissier *et al.* recensent de manière exhaustive les caractéristiques propres aux systèmes multi-agents et à leurs applications [Boissier *et al.*, 2004]. Citons par exemple adaptation, autonomie, but, concurrence, croyance, désir, distribution, environnement, groupe, intention, interaction, norme, organisation, ouverture, pro-activité, protocole, réactivité, rôle ou tâche. Cet axe de comparaison va donc aussi s'attacher à évaluer les problèmes auxquels la méthode peut répondre.

2.4.1.2 Les notations et langages de modélisation

Comme nous l'avons vu dans les paragraphes précédents, les langages de modélisation agents sont souvent graphiques, inspirés de l'ingénierie objet, des besoins ou des connaissances. Ils comportent des symboles, une syntaxe et une sémantique propres. Là encore, l'évaluation s'établit sur l'analyse des propriétés des langages utilisés [Shehory et Sturm, 2001; Dam et Winikoff, 2003; Cernuzzi et Rossi, 2002; Arlabosse *et al.*, 2004] :

- *Accessibilité* : le langage est-il facile à apprendre et à utiliser ? Demande-t-il un long temps d'apprentissage ?
- *Complexité* : y a-t-il une possibilité de visualiser la complexité du système à plusieurs niveaux d'abstraction ?
- *Consistance* : y a-t-il possibilité de vérifier la consistance d'une spécification ?
- *Exécution* : y a-t-il une fonctionnalité de prototypage permettant des exécutions préliminaires ?
- *Expressivité* : est-ce que tous les aspects de la conception de l'application sont pris en compte par la notation ?
- *Modularité* : est-ce que la modification des spécifications en amont implique une remise en cause de la totalité des spécifications ?
- *Portabilité* : est-ce que le langage est indépendant de quelconques choix de développement (langage de programmation par exemple, ou architecture) ?

– *Précision* : le langage ne doit pas être ambigu. Y a-t-il des définitions précises établies ?
– *Raffinage* : le concepteur est-il guidé pour raffiner les modèles utilisés ?
– *Traçabilité* : est-il facile de modifier des spécifications passées, de tracer ses changements sans avoir de répercussion sur les spécifications en cours ?

Tous ces critères sont à prendre en compte lors du choix d'une méthode utilisant un langage dédié ou bien lorsqu'un développeur veut lui-même définir son propre langage de modélisation.

2.4.1.3 Le processus

Compte tenu de l'influence qu'a pu avoir le monde de l'objet, les cycles de vie et processus utilisés dans la conception de systèmes multi-agents sont de types classiquement utilisés en conception modulaire : cascades, itératifs, en V, en spirales. Comme nous l'avons précisé dans le paragraphe 2.1, les principales phases d'un processus de développement sont : l'analyse des besoins, la conception de l'architecture (ou analyse), la conception détaillée, le développement (ou implémentation) et le déploiement. Toutes les méthodes ne couvrent pas tout ce cycle, et il est nécessaire de le prendre en compte lors de l'évaluation d'une méthode.

Pour les méthodes proposées nous allons aussi analyser :
– *Le contexte de développement*, c.-à-d. est-ce que le processus est adéquat pour le développement de nouvelles applications, la ré-ingénierie (*reengineering*), le prototypage, la réutilisation, pour faire appel à des schémas de conception pré-existants, utiliser des bibliothèques dédiées, faire du *reverse engineering* ?
– *Les délivrables* sont-ils bien identifiés et associés à des étapes précises ?
– *La vérification et la validation* sont-elles faciles et rapides, voire automatisables ?
– *La gestion de la qualité* est-elle prise en compte ?
– *Les directives de conduite de projet* sont-elles claires ?

2.4.1.4 La pragmatique

Shehory et Sturm ou Dam et Winikoff proposent de s'intéresser à l'aspect pragmatique de la méthode, c.-à.d. la gestion et les questions techniques. Ceci inclut les supports mis en place pour l'utilisation de la méthode (livres, textes en ligne, ou logiciel d'aide au suivi), le prix à payer pour choisir une nouvelle méthode, ou le besoin d'expertise. Le critère technique porte plutôt sur la spécialisation des méthodes à des domaines particuliers (gestion de collectifs humains, données distribuées, résolution de problèmes, simulations, etc).

2.4.2 Analyse de méthodes existantes

Dans ce paragraphe, nous allons dresser une comparaison des principales méthodes orientées agent/multi-agent. Chaque méthode sera décrite brièvement puis nous examinerons les quatre axes de comparaison. Elles ne sont pas ordonnées de manière chronologique mais alphabétique.

2.4.2.1 AAII

AAII signifie *Australian Artificial Intelligence Institute Methodology* et a été notamment développée par Kinny *et al.* pour la gestion du trafic aérien [Kinny *et al.*, 1996].

Les concepts clés. AAII est dédiée à la conception d'agents BDI impliqués dans une organisation et occupant des rôles précis. Ainsi, AAII axe la conception suivant deux points de vue, tout en s'intégrant dans un cadre orienté objet :

- le point de vue *externe* aux agents qui définit l'*organisation*, les *rôles*, les *services* et les *responsabilités* ;
- le point de vue *interne* aux agents qui est une instanciation de l'architecture BDI au problème.

AAII propose plusieurs modèles à instancier au cours du développement de la vue externe, qui sont totalement indépendants de l'architecture BDI :

- le *modèle d'agent* qui décrit la hiérarchie entre agents (*modèle de classes d'agents*) et identifie les multiples instances possibles (*modèle d'instances d'agent*) ;
- le *modèle d'interaction* qui décrit les responsabilités, les services et les interactions entre agents.

Compte tenu du choix de l'architecture BDI, la vue interne est décrite suivant trois modèles correspondants :

- le *modèle de croyances* (*beliefs*) qui décrit les informations à propos de l'environnement, l'état interne des agents, et leurs actions possibles ;
- le *modèle de buts* (*goals*) qui décrit les *buts* que l'agent peut adopter et les évènements auxquels il peut répondre ;
- le *modèle de plans* qui décrit sous forme de *plan* les moyens pour un agent d'atteindre ses buts.

AAII ne prend pas en compte les systèmes ouverts (en terme de nombre variable d'agents), et impose une hiérarchie au sens objet du terme. De plus, le potentiel des agents BDI est contraint par l'utilisation de simples graphes ET/OU pour modéliser les buts, et ni les normes, ni les règles sociales ne sont définies. La notion d'environnement n'est pas discutée lors du développement ; du moins de manière séparée des autres notions.

Les notations. Se voulant être une extension des méthodes de conception orientées objet classiques, AAII repose principalement sur des notations de types UML. Les modèles d'agents et d'instances sont regroupés dans des diagrammes type diagrammes de classes ou de collaboration. La hiérarchie entre agents et l'organisation sont définies grâce aux relations de composition, d'agrégation ou d'héritage. Les classes peuvent être abstraites (marquées avec un Ⓐ) et les instances peuvent être considérées comme des ensembles statiques d'agents (marquées avec un Ⓢ). La figure 2.7 montre un exemple de diagramme de classes (boîtes anguleuses) et d'instances (boîtes arrondies) d'agents pour l'application au trafic aérien.

Le modèle de croyances consiste en un ensemble de croyances et un ou plusieurs états de croyances. Un ensemble de croyances est spécifié par des diagrammes d'objets sur le domaine de croyances d'une classe d'agents. Formellement, un ensemble de croyances est

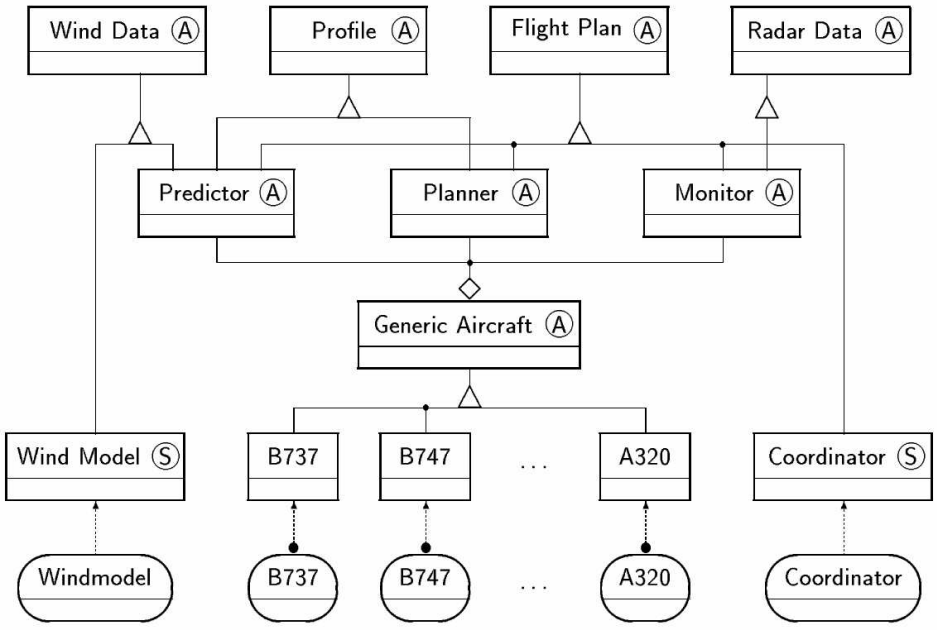

Figure 2.7 — Un exemple de diagramme de classes et d'instances pour le problème du trafic aérien dans AAII.

spécifié comme un ensemble de prédicats et de fonctions dérivées du domaine. Un état de croyance est un ensemble de diagrammes d'instances définissant une instance particulière de l'état de croyance. Enfin, le modèle de plans est un ensemble de plans qui sont spécifiés grâce à des machines à états/transitions avec possibilité d'échec à l'exécution.

Le processus. Le processus de AAII suit les deux axes de conception : externe et interne. Une première phase consiste en l'élaboration de la vision externe en quatre étapes :

1. **Identifier les rôles du domaine d'application** : avec la définition de l'organisation structurelle et fonctionnelle ainsi que les premières classes d'agents (souvent abstraites) de granularité indifférente ;
2. **Identifier les responsabilités et les services** nécessaires pour remplir ces responsabilités pour chaque rôle d'agent ;
3. **Identifier les interactions** induites par les responsabilités pour chaque service, et définir les actes de langage nécessaires à la communication ainsi que leur contenu . C'est à ce niveau que la définition interne de chaque agent peut commencer ;
4. **Raffiner la hiérarchie d'agents** en introduisant de nouvelles classes ou sous-classes d'agents, en composant des classes d'agents. Enfin, établir les instances d'agents nécessaires.

A partir de l'identification des interactions, il est possible de commencer à modéliser la vue interne des agents, en deux points, à itérer autant de fois qu'il y a d'agents :

1. **Analyser les moyens d'atteindre les buts** : ceci revient à décomposer les buts en sous-buts ou en actions puis à les ordonnancer et à les conditionner, et enfin à identifier les possibilités d'échec ;

2. **Construire les croyances du système** : à partir des conditions d'atteinte de buts et de sous-buts dans des contextes particuliers, définir les croyances nécessaires aux agents pour raisonner.

Malgré la clarté de ce processus, il reste néanmoins limité aux phases préliminaires de développement (l'analyse en particulier), sans toutefois proposer d'aide à l'identification des agents. Aucun support de conception ou de développement n'est spécifié, c'est aux utilisateurs de travailler. Par contre, les délivrables sont clairement identifiés tout au long du processus, mais la gestion de la qualité est inexistante.

La pragmatique. AAII est une méthode assez ancienne, et peu de ressources sont disponibles pour des concepteurs novices désirant tenter de l'utiliser réellement. Compte tenu du langage graphique utilisé, modéliser avec AAII n'est pas difficile si l'on connaît un peu la conception objet. Par contre, l'utilisation de logique pour définir les états internes des agents ne peut être facilement accessible à tous. Les domaines d'application de AAII sont assez restreints, car les systèmes spécifiés sont fermés. Elle a notamment (et visiblement uniquement) été utilisée pour le contrôle de trafic aérien [Kinny *et al.*, 1996]. L'avantage de se limiter à l'analyse du système, n'implique l'utilisation d'aucun langage spécifique de développement et permet potentiellement un passage à l'échelle.

2.4.2.2 Aalaadin

Aalaadin est à la base un projet de recherche, plus qu'une réelle méthode. Néanmoins, c'est un cadre de développement notable de systèmes multi-agents, fournissant des indications méthodologiques, certes partielles, mais surtout un environnement de prototypage et d'exécution idéal pour des agents reposant sur les notions de groupe et de rôle, grâce au modèle AGR [Ferber et Gutknecht, 1998].

Les concepts clés. Aalaadin va de pair avec le modèle AGR (*Agent/Groupe/Rôle*). Le métamodèle UML d'AGR est présenté dans le paragraphe 2.2.4. Un *agent* joue des *rôles* au sein de *groupes*. Un agent peut avoir plusieurs rôles et appartenir à plusieurs groupes. Le rôle est une notion fonctionnelle abstraite, spécifique à un groupe et pouvant être joué par plusieurs agents. Un rôle est caractérisé par son unicité (vraie ou fausse), ses *compétences* (ou *services*), et ses *capacités* afin de remplir son rôle. Un groupe est un ensemble d'agents ayant des points communs et qui peuvent communiquer. La communication est interdite, pour des raisons de sécurité entre agents de groupes différents.

Les organisations obtenues peuvent être hiérarchisées par composition de groupes, grâce à des agents frontières appartenant à deux groupes par exemple. Aalaadin propose une vision centrée organisation plutôt que centrée agent pour des raisons de sécurité, de normes et de responsabilités [Ferber *et al.*, 2003]. Une organisation peut être vue à deux niveaux (voir figure 2.8) :

- *La structure organisationnelle* représente les relations qui font d'une agrégation d'individus un tout identifiable, c.-à-d. un invariant organisationnel ;
- *l'organisation concrète* est une instance de structure organisationnelle avec des agents, des rôles et des groupes particuliers.

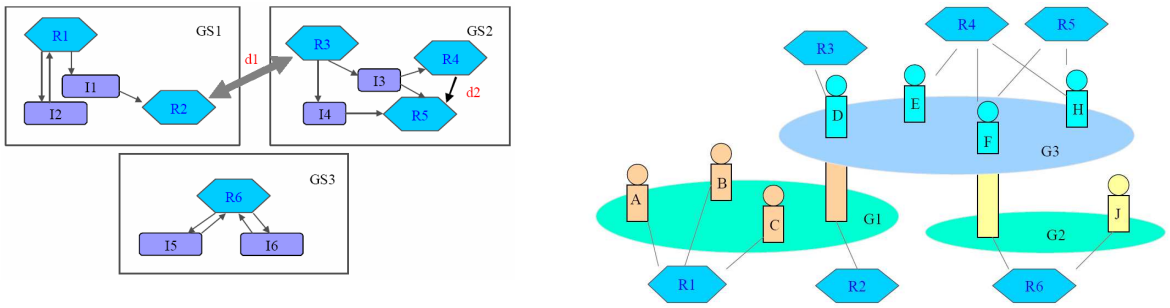

Figure 2.8 — Les diagrammes de structure organisationnelle (à gauche) et d'organisation concrète (à droite) dans Aalaadin.

Aalaadin ne fait aucun présupposé sur les capacités "mentales" des agents. Ils peuvent être totalement réactifs, ou bien pro-actifs à gros grain avec une gestion complexe des états mentaux. Ils ont uniquement besoin de gérer les rôles et l'appartenance aux groupes. Ici, l'autonomie est donc un présupposé, issu de la définition de Ferber [Ferber, 1995].

Les notations. La modélisation graphique d'une organisation suivant AGR peut passer par trois types de diagrammes différents :

Les diagrammes de structures organisationnelles où apparaissent structures de groupes (rectangles), rôles (hexagones), contraintes (flèches), et interactions (rectangles arrondis) entre rôles (voir figure 2.8) ;

Les diagrammes d'organisations concrètes (ou *plateaux à fromage*) permettent de représenter l'appartenance d'agents (quilles) qui jouent des rôles (hexagones) dans des groupes (ovales). L'appartenance d'un agent à plusieurs groupes se fait en étirant la quille représentant l'agent sur tous les ovales des groupes d'appartenance (voir figure 2.8) ;

Les diagrammes organisationnels d'interaction spécifient les interactions entre groupes et/ou rôles. Ce sont des diagrammes de séquences type UML, dans lesquels les lignes de vies sont attribuées à des rôles et non à des instances de classes, comme dans A-UML. Ces diagrammes permettent notamment de représenter les créations ou destructions de groupes.

Ces notations n'exploitent cependant que l'aspect organisationnel du système, et donc sont imprécises. C'est en cela qu'Aalaadin reste incomplète. Il n'y a aucun moyen de spécifier le comportement interne et computationnel des agents. Par contre, les notions de complexité d'organisation sont abordées par le biais de la hiérarchisation de groupes, restant cependant statique. Bien qu'AGR ne soit pas dépendant d'un langage, il est conseillé d'utiliser la plate-forme MadKit[10] pour développer des agents organisationnels. De plus, ceci donne de bonnes directives concernant la structuration des agents.

Le processus. Aalaadin ne propose pas de processus élaboré, mais il peut se résumer en trois phases [Gutknecht et Ferber, 1999] :

[10] `http ://www.madkit.org`

1. **L'analyse** permet d'identifier les fonctions du système et les dépendances au sein de communautés identifiées. Il convient de définir quels sont les mécanismes de coordination et d'interaction entre les entités d'analyse.
2. **La conception** est la phase la mieux décrite. Elle contient l'identification des groupes et des rôles dans des diagrammes de structures organisationnelles. Les protocoles d'interactions alors identifiés entre rôles sont décrits dans des diagrammes organisationnels de séquences ou A-UML. L'identification des objets et des actions sur ces objets permet de commencer à décrire les attributs des agents.
3. **La réalisation** commence par le choix de l'architecture d'agent. Puis, la gestion des entités du domaine permet d'implanter le système à partir d'organisations concrètes. Le mieux étant d'utiliser MadKit comme plate-forme de prototypage et de simulation.

Le modèle AGR peut s'adapter à n'importe quel contexte, bien qu'aucune gestion de projet soit mise en place. Le cycle de développement est assez limité, très peu documenté et les délivrables ne sont pas associés, de manière claire, à des étapes précises. En effet Gutknecht et Ferber n'ont jamais voulu proposer de réel processus pour ne pas réduire le potentiel d'Aalaadin à s'intégrer dans des processus ascendants ou descendants [Gutknecht et Ferber, 1999].

La pragmatique. Les ressources sur Aalaadin ne sont pas très nombreuses, contrairement à MadKit qui bénéficie d'un développement constant (version 4.0 en février 2004). Aucune expertise particulière n'est nécessaire pour se lancer dans la conception d'agents suivant le modèle AGR. MadKit, par contre, demandera un peu de travail, mais comme il est très documenté, le prototypage est assez facile à prendre en main et les possibilités de simulation, de déploiement et d'observation des systèmes construits sont grandes. Comme MadKit est une plate-forme Java, le langage privilégié est Java, mais avec des accès Scheme, Jess ou Python.

2.4.2.3 Cassiopée

Cassiopée, développée par Collinot et Drogoul, est l'une des méthodes les plus originales, car elle est issue du domaine de la simulation multi-agent influencée par l'éthologie [Collinot et Drogoul, 1996, 1998; Drogoul, 2000]. C'est l'une des seules méthodes *bottom-up*, ou ascendantes, qui part des actions nécessaires à l'obtention d'une tâche globale, pour arriver à la définition des rôles organisationnels et structures collectives.

Les concepts clés. Cassiopée repose sur quelques concepts clés : *rôle*, *agent*, *dépendances* et *groupes*. Un agent est vu comme un ensemble de rôles organisés en trois niveaux :

Les rôles individuels, qui sont les comportements que les agents sont individuellement capables de mettre en oeuvre, indépendamment de la façon dont ils le font ;

Les rôles relationnels, c.-à-d. comment ils choisissent d'interagir avec les autres (en activant/-désactivant les rôles individuels), en se basant sur les dépendances mutuelles qu'entretiennent les différents rôles individuels ;

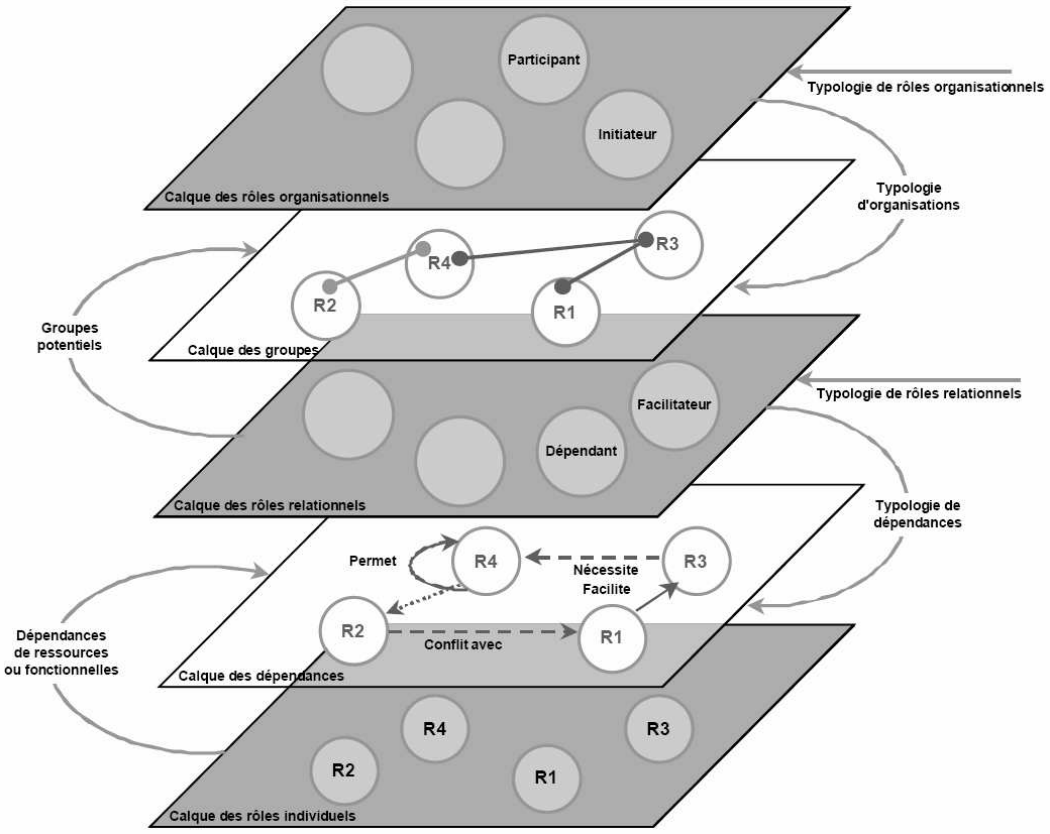

Figure 2.9 — Les cinq calques de Cassiopée (en gris le micro-niveau, en blanc le macro-niveau).

Les rôles organisationnels, ou comment les agents peuvent gérer leurs interactions pour devenir ou rester organisés (en activant/désactivant leurs rôles relationnels).

Les notations. La spécification des comportements des agents repose principalement sur deux types de graphes dans Cassiopée : les graphes de couplage des comportements élémentaires, et les graphes d'influences (voir figure 2.9 [Drogoul, 2000]). Ces graphes sont des graphes de type états/dépendances. Les noeuds représentent les états possibles d'un agent, et les flèches spécifient des dépendances en terme de coopération entre les états par exemple. Les graphes d'influences permettent de dériver des rôles relationnels et des types d'agents par groupement d'états.

Le processus. Le processus de Cassiopée se décompose en cinq phases, ou cinq *calques* (voir figure 2.9), qui alternent entre le point de vue local (agents) et le point de vue global (organisation) :

1. **Le calque des rôles individuels**, qui définit les différents rôles d'agents, ou tâches, nécessaires, et permettent de définir les différents types d'agents ;
2. **Le calque des dépendances**, qui définit les dépendances entre rôles, grâce à des graphes de dépendances ;

3. **Le calque des rôles relationnels**, qui définit la façon dont les agents gèrent ces dépendances, en jouant les rôles relationnels adéquats ;
4. **Le calque des groupes**, qui définit les groupes pouvant apparaître en cours de résolution ;
5. **Le calque des rôles organisationnels**, qui décrit la dynamique de ces groupes, c.-à- d. les rôles organisationnels que les agents peuvent jouer pour les faire apparaître, évoluer ou disparaître.

Ce processus se limite donc aux phases d'analyse, et au début de la conception, si on considère la définition des états internes comme appartenant à la conception (dans les processus classiques orientés objet, c'est le cas). Comme pour AAII, ceci implique une totale liberté d'implémentation. Cependant, l'extension de Cassiopée, appelée Andromède, fournit un outil d'aide à la conception [Drogoul et Zucker, 1998]. Le contexte de développement, par contre, se limite à la création de nouveaux systèmes, et ne propose pas de réutilisation. Les délivrables sont peu nombreux mais très bien définis et spécifiés (les graphes), mais sans gestion de qualité ni de projet.

La pragmatique. Les articles et rapports sur Cassiopée sont peu nombreux mais suffisants pour utiliser Cassiopée de manière efficace. Les principes et les notations de la méthodes étant simples, les nouveaux utilisateurs auront peu de difficulté à analyser des problèmes multi-agents. La principale originalité est que le processus est ascendant, et donc l'analyse en devient simplifiée et claire, ne nécessitant aucune expertise. Les principales applications se sont limitées par contre à des problèmes multi-robots, de type robots footballeurs [Collinot et Drogoul, 1998]. Aucune application à grande échelle n'a été développée pour l'instant.

2.4.2.4 DESIRE

DESIRE, qui signifie *DEsign and Specification of Interacting REasoning framework*, fait figure de pionnier dans le domaine des méthodes orientées agents [Brazier *et al.*, 1997, 1998] et est un exemple de méthodie directement issue de l'ingénierie des connaissances. DESIRE manipule les structures de connaissances, les décompositions de tâches, les échanges d'information, l'ordonnancement et la délégation de tâches.

Les concepts clés. DESIRE repose principalement sur trois concepts clés :

La représentation des connaissances est à la fois graphique et textuelle. La notation graphique est directement inspirée des notation de type *Entité-Relation* dans lesquellse apparaissent différents types de noeuds : les sortes (*sorts*) qui représentent une partie du domaine (ex : un prix, un produit), les objets qui sont des instances de sortes (ex : 100€, le produit A), les fonctions entre des ensembles de sortes (ex : comparaison entre deux produits), les relations nécessaires aux déclarations (ex : la relation "est moins cher"), les méta-descriptions permettant l'association entre un état du monde et une sorte, et enfin les types d'information qui sont des spécification d'ensembles de sortes, de relations, d'objets et de fonctions.

Le modèle générique d'agent adopté par DESIRE est de type BDI, et est caractérisé comme faible (*weak*), dans le sens de [Wooldridge et Jennings, 1995]. Un tel agent possède au moins les quatre types de comportements suivants : autonomie (de contrôle), comportement réactif, comportement pro-actif, comportement social (communiquer, coopérer, ...). Un agent est décomposé en tâches (voir figure 2.10), qui peuvent être par exemple : le contrôle de son propre processus, la gestion des interactions avec les agents, la maintenance des informations sur les agents, la gestion des interactions avec le monde, la maintenance des informations sur le monde, la maintenance de son historique, la gestion de la coopération, et la tâche spécifique de l'agent (en fonction de l'application).

Le modèle de composition de tâches, qui décrit chaque tâche des agents. Ce modèle compositionnel permet de décomposer des tâches en sous-tâches, de spécifier des relations de délégation ou de séquence. Ce modèle est fonctionnel dans le sens où chaque tâche est vue comme un composant ayant des entrées et des sorties d'informations [Brazier *et al.*, 1998]. Ce modèle permet de définir le modèle d'agent générique comme un ensemble de tâches organisées et possède lui aussi un pendant textuel.

Ainsi, dans DESIRE, les propriétés d'autonomie, de réactivité et de pro-activité sont exprimées dans la tâche de contrôle de la connaissance, et les principaux concepts multi-agents manipulés sont :

- l'*agent*, qui est vu comme un composant/tâche ;
- les *croyances*, les *désirs*, et les *intentions* qui sont des objets de connaissance ;
- les *messages* qui sont des liens entre composants/tâches ;
- les *rôles* qui sont des composants/tâches ;
- et les *tâches*.

Par contre, aucune notion de société, d'organisation, de norme ou de groupe n'est soulevée.

Les notations. Dans DESIRE, deux types de notations cohabitent : la notation pour le modèle de connaissances, inspirée de l'ingénierie des connaissances, et la notation pour les modèles de tâches (dont celui d'agent). Les notations graphiques des connaissances sont de simples graphes noeud/lien avec autant de types de noeuds que de types d'entités : sortes, objets, fonctions, relations, méta-descriptions et types d'information. La notation graphique compositionnelle de DESIRE permet de définir les tâches et leur relation de séquentialité et de délégation par composition. La figure 2.10 montre comment le modèle générique d'agent peut-être décrit par composition de tâches en employant cette notation.

Les deux notations (connaissances et tâches) sont automatiquement traduisibles en version textuelle et inversement. La version textuelle permet de définir des règles plus précises de conditionnement et de séquentialité par l'usage de logique temporelle.

DESIRE possède de nombreux avantages concernant ses notations. En effet, la logique temporelle garantie une réelle précision dans les spécifications. La modélisation compositionnelle est aussi un exemple de modularité, et la hiérarchie potentielle permet de gérer des systèmes complexes en termes de niveaux d'abstraction (possibilité de zoom). DESIRE possède un outil de vérification sur les spécifications formelles, et permet une bonne analyse des modèles proposés. Par contre, DESIRE, possédant un large spectre de capacités de mo-

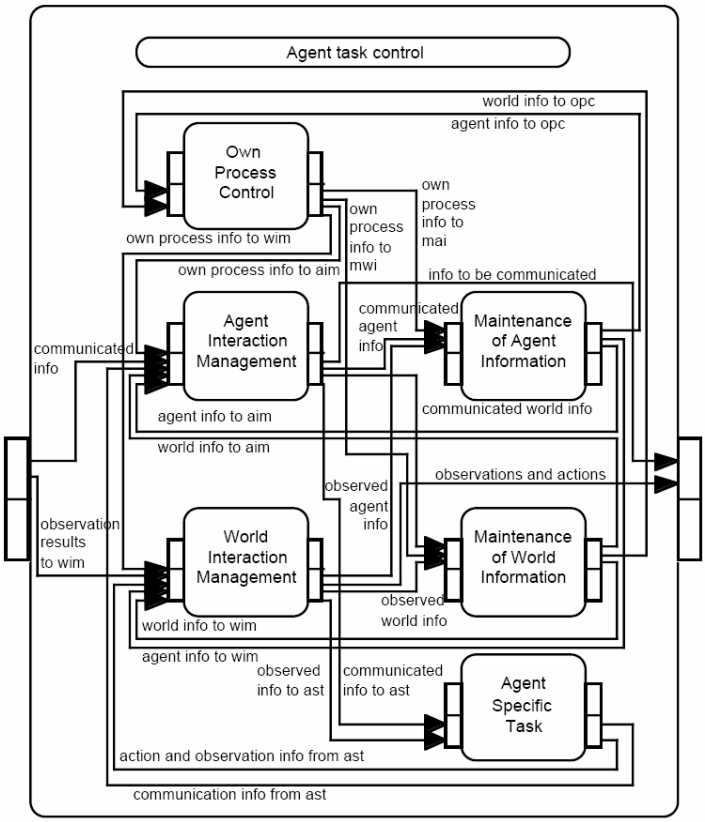

Figure 2.10 — Le modèle d'agent générique de DESIRE – décomposition en tâches.

délisation, peut devenir difficile à appréhender. De même, la modélisation purement compositionnelle ne permet pas de décrire des algorithmes précis – comme peut le permettre A-UML par exemple.

Le processus. Comme le précisent Brazier *et al.*, il n'y a pas de séquence fixée de conception, seulement différents types de connaissances qui sont accessibles à différentes étapes de la conception. Néanmoins, la méthode se décompose en cinq points :

1. **La description du problème** qui inclut les besoins imposés à la conception ;
2. **Les principes de conception** ou *design rationale* qui spécifient les choix faits lors de la conception ;
3. **La conception conceptuelle** ou *conceptual design* qui inclut un modèle conceptuel pour chaque agent, pour le monde extérieur et pour les interactions entre agents ;
4. **La conception détaillée** qui spécifie tous les aspects de connaissance et de comportement ;
5. **La conception opérationnelle** qui spécifie les paramètres nécessaires à l'implémentation.

La couverture du processus de DESIRE est donc assez limitée, car ne faisant référence qu'à la phase de conception. De plus, aucun processus de type pas-à-pas est fourni, seulement des instructions indépendantes, et donc aucune information sur les délivrables à fournir à un instant donné n'est disponible. Côté conduite de projet, aucune gestion de la qualité,

ni des ressources nécessaires n'est explicité. Par contre, le grand avantage de cette approche est bien sûr dû à l'approche formelle et logique qui permet de couvrir pleinement les tâches de vérification et de validation – ce que peu de méthodes permettent.

La pragmatique. Les ressources concernant DESIRE sont limitées (il fallait pouvoir assister aux rares tutoriaux proposés). De plus, une certaine expertise – notamment en logique temporelle – est nécessaire à sa manipulation. Un autre problème majeur est la restriction du langage de programmation : il est propriétaire et unique. Ainsi, DESIRE sera plutôt utilisée pour faire du prototypage. Par contre, le cadre de DESIRE est complet. En effet, il fournit une méthode, un langage de spécification formel, des outils logiciels, un générateur de code et des outils de vérification. Seulement, l'atelier est totalement propriétaire. Enfin, mais ceci est certainement la chose la plus importante, DESIRE est une des seules méthodes à avoir été réellement utilisée dans l'industrie – certes avec ses créateurs aux commandes – pour développer par exemple un logiciel d'aide au diagnostic de maladies cardiaques ou bien de la conception de tâches en entreprise.

2.4.2.5 Gaia

Gaia est une extension des approches d'ingénierie logicielle classique [Wooldridge *et al.*, 2000]. C'est une méthode de la seconde génération, par opposition à la première génération que formaient AAII ou DESIRE. Elle est donc plus complète et bénéficie, de plus, d'une large reconnaissance dans le domaine multi-agent.

Les concepts clés. Gaia se veut être générale et applicable à n'importe quel domaine, et compréhensible par la distinction entre macro-niveau et micro-niveau. Les agents modélisés, pouvant être hétérogènes, sont des systèmes computationnels à gros grain, qui vont essayer de maximiser une mesure de qualité globale. Toutefois, Gaia ne prend pas en compte les systèmes admettant de réels conflits. L'organisation (d'une centaine d'agents environ) est clairement statique dans le temps, de même que les services offerts par les agents.

Gaia manipule six modèles d'analyse et de conception différents :

Le modèle de rôle qui identifie les différents rôles devant être joués par les agents du système. C'est une description abstraite de la fonction attendue pour une entité donnée. Outre sa description textuelle, un rôle possède trois attributs – les *responsabilités*, les *permissions/-droits* et les *protocoles* de communication disponibles pour le rôle – qui sont regroupés dans un schéma de rôle. Les permissions établissent les ressources auxquelles un rôle a le droit d'accéder. Les responsabilités déterminent la fonctionnalité du rôle en terme de sûreté (*safety*) et de vivacité (*liveness*) ;

Le modèle d'interaction qui définit les protocoles de communication entre agents. Dans Gaia un protocole est défini comme par nom, un expéditeur, un récepteur, une description, des entrées et des sorties. Les protocoles définissent les dépendances et les relations entre rôles ;

Le modèle d'agent qui attribue les rôles aux différents agents du système. Il identifie les différents types d'agents et leurs instances ;

Le modèle de service qui définit, comme son nom l'indique, les différents *services* offerts par le système, et les agents tributaires. Un service est vu comme une fonction d'un agent et possède les caractéristiques suivantes : des entrées, des sorties, des pré-conditions d'exécution et des post-conditions.

Le modèle d'organisation ou d'accointances qui définit la structure de l'*organisation* grâce à des graphes orientés représentant les voies de communication entre agents. Différent types de relations existent : contrôle, pair (*peer*), dépendance, et d'autres pouvant être définis ;

Le modèle environnemental[11] qui décrit les différentes ressources accessibles caractérisées par les types d'actions que les agents peuvent entreprendre pour les modifier.

Gaia rend facile la manipulation d'un grand nombre de concepts multi-agents, grâce à ces modèles, ce qui a certainement fait sa popularité. La propriété d'autonomie de contrôle ou de rôle des agents est exprimée par le fait qu'un rôle encapsule sa fonctionnalité, qui est interne et non affectée par l'environnement. La réactivité et la pro-activité sont, elles, plus ou moins exprimées grâce aux propriétés de vivacité des responsabilités. On peut regretter que l'environnement ne soit défini que comme une liste de variables caractérisées en lecture ou écriture. Enfin, les notions sociales sont abordées dans le modèle organisationnel d'accointances.

Les notations. Gaia ne fournit pas de notation graphique à proprement parler. En effet, le modèle de rôles et les protocoles ne sont décrits que par des tables. Ceci n'enlève en rien la précision obtenue grâce, notamment, aux propriétés de sûreté ou de vivacité. Les autres modèles sont aussi clairement accessibles. Les rôles étant clairement distincts, la modularité fait partie des avantages de cette approche. Par contre, comme aucune présentation hiérarchique (à la DESIRE par exemple) n'est disponible, la gestion de la complexité organisationnelle ou fonctionnelle n'est pas prise en compte et limite donc les systèmes développés grâce à Gaia, à des organisations simples et rigides. Enfin, Gaia n'aborde pas le problème de l'exécutabilité.

Le processus. La figure 2.11 montre le processus de développement proposé dans Gaia. Trois phases sont principalement abordées : l'analyse, la conception architecturale et la conception détaillée.

Lors de l'analyse, le système se voit divisé en sous-organisations, dont vont découler les modèles d'environnement, de rôle et d'interaction (préliminaires). A partir de ces modèles, l'analyse doit spécifier les règles organisationnelles (permissions, vivacité et sûreté) de dépendances entre rôles. La conception architecturale doit aboutir au raffinage des modèles de rôle et d'interaction par l'analyse des structures organisationnelles. Enfin, lors de la conception détaillée, les modèles d'agent (détermination des types et instances d'agents) et de services sont implantés.

Ce processus est donc très limité, et se focalise uniquement sur les premières phases de conception. Il n'y a donc pas de phase de vérification/validation. La gestion de projet et de

[11] Les auteurs ont introduit ce modèle que très tardivement.

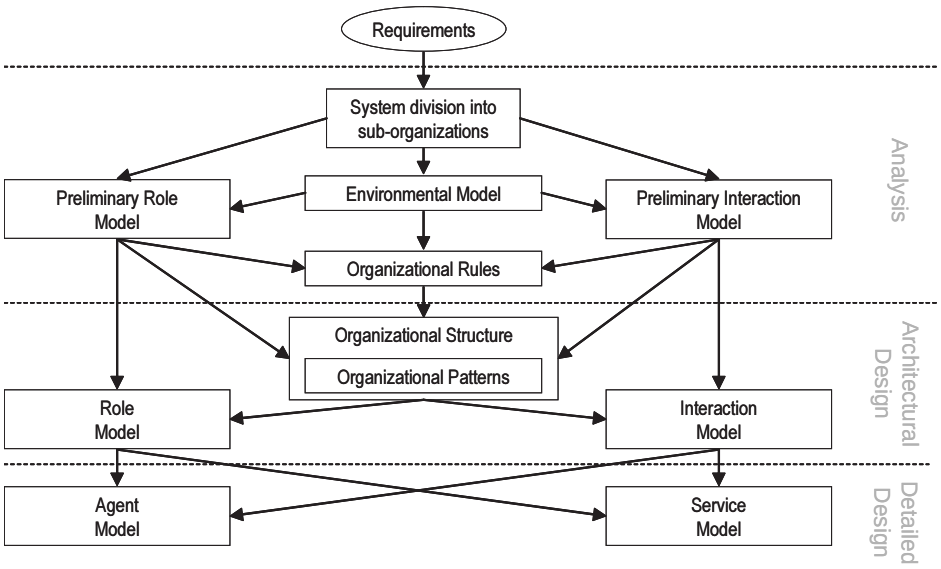

Figure 2.11 — Le processus de développement de Gaia et les modèles manipulés.

qualité n'est pas abordée. De plus, aucune séquence d'activités à mener est réellement expliquée, seule des directives sont données. Néanmoins, les délivrables et différents modèles à fournir au cours du processus sont très clairement définis.

La pragmatique. Les ressources disponibles sur Gaia sont assez limitées. De plus, Gaia nécessite de posséder une solide maîtrise de la logique temporelle, ce qui est aussi le cas de DESIRE. Ceci la rend plutôt difficile à adopter pour des ingénieurs. Comme précisé dans les concepts, Gaia est limitée aux applications à agents à forte granularité, peu nombreux, et avec une organisation statique, ce qui rend le passage à l'échelle difficile. Malgré quelques travaux sur une spécification de directives de passage à la plate-forme JADE, Gaia ne propose aucune étude d'implémentation, ce qui a, bien sûr, l'avantage de laisser le développeur libre d'utiliser n'importe quel langage de programmation [Moraitis *et al.*, 2002].

2.4.2.6 MaSE

MaSE, développée par Deloach *et al.*, signifie *Multiagent Software Engineering* et est un exemple d'approche complète de développement de systèmes multi-agents de l'analyse au déploiement, avec de nombreux modèles graphiques et une approche logique [Deloach, 1999; Deloach et Wood, 2000; Deloach *et al.*, 2001].

Les concepts clés. MaSE manipule neufs modèles très proches de ceux utilisés en conception orientée objet :

La hiérarchie de buts est obtenue en décomposant le *but* principal du système en sous-buts à partir de l'expression des besoins ;

Les cas d'utilisation qui définissent une séquence d'événements pouvant apparaître dans le système. Ce modèle est souvent retrouvé dans les processus de conception objet afin

de décrire à la fois la fonctionnalité du système et son environnement en terme d'utilisateurs et d'acteurs. Dans MaSE, il permet notamment d'identifier les *rôles* joués par le système (et donc ses agents) ;

Les diagrammes de séquences sont définis pour chaque cas d'utilisation identifié. Dans ce modèle, les interactions entre rôles sont décrites comme des envois de *messages*.

Les rôles sont vus comme le moyen d'atteindre les buts fixés au système. Généralement, un but est associé à un rôle unique, mais ceci n'est pas une règle systématique ;

Les tâches concurrentes décrivent les moyens pour les rôles d'atteindre les buts. Ils sont définis par des automates à états finis (ou bien des réseaux de Pétri au début de MaSE) dans lesquelles apparaissent les tâches conditionnées et ordonnancées ;

Les classes d'agents sont créées à partir des rôles identifiés. Les architectures d'agents utilisées ne sont pas imposées (BDI ou autres). Une classe d'agent est une extension d'une classe d'objet qui peut jouer plusieurs rôles, et qui possède des attributs et des méthodes ;

Les conversations définissent les protocoles de coordination entre deux agents. Ils consistent en deux automates à états finis, un pour l'initiateur et un pour le récepteur ;

L'architecture consiste en un ensemble de composants qui peuvent être des classes ou des ensembles de classes. Elle détaille les différents type d'agents ou objets présents dans le système.

Les diagrammes de déploiement permettent de spécifier la place des agents sur les plates-formes utilisées et les liens de discussion nécessaires entre les plates-formes.

Dans MaSE, l'autonomie des agents est, comme dans Gaia, assurée par l'encapsulation des fonctionnalités dans les rôles. De même, la pro-activité est exprimée par les tâches, spécifiées par des automates, qui sont attribuées aux rôles. Par contre, la réactivité n'est pas clairement abordée. En effet, il n'y a pas de lien direct entre les événements et les actions ; mais il est possible de le définir en utilisant des machines à états. Les propriétés sociales (groupe, organisation) ne sont pas non plus clairement définies, contrairement aux normes (règles organisationnelles) ou protocoles (conversations).

Les notations. Les nombreux modèles de MaSE utilisent des notations directement inspirées de la conception objet. La hiérarchie de but est simplement définie grâce à un arbre dont les noeuds sont les buts et les liens représentent les décompositions. Les diagrammes de séquences sont quasi-similaires aux diagrammes UML du même nom, avec la différence que ce sont des rôles et non des classes qui communiquent. Les cas d'utilisations apparaissent au même niveau que les buts. Les tâches sont modélisées grâce à des automates à états finis, ainsi que les protocoles. Les diagrammes d'agents sont des diagrammes de classes UML. Enfin, les diagrammes de déploiement sont aussi très proches de la notation UML.

S'inspirant d'UML, MaSE en retire tous les avantages d'une telle notation graphique : accessibilité (mais avec le désavantage de la synchronisation nécessaire lors de la transformation de modèles), analysabilité, gestion de la complexité par décomposition des buts et des classes (mais pas pour les tâches et les rôles), expressivité (mis à part les flot de données et la représentation des connaissances comme dans DESIRE). Les diagrammes d'agents de

MaSE permettent une certaine réutilisabilité, mais pas pour les tâches, les rôles ou les protocoles. Là où l'on pourrait s'attendre à un défaut d'exécutabilité, dû à la filiation UML, MaSE fournit un outil, *agentTool*, qui permet une génération de code partielle [Deloach et Wood, 2000].

Le processus. Le processus de MaSE est très clair, comme le montre la figure 2.12, et se décompose en deux phases. La première phase, l'analyse, se décompose en trois étapes :

1. **Identifier les rôles** à partir des besoins de l'utilisateur et les structurer dans un diagramme de hiérarchie de rôles ;
2. **Identifier les cas d'utilisations et créer les diagrammes de séquences** associés pour aider à l'identification d'un ensemble initial de rôles et de voies de communication ;
3. **Transformer les buts en ensembles de rôles** :
   (a) **Créer un modèle de rôle** pour saisir les rôles et les tâches associées ;
   (b) **Définir un modèle de tâches concurrentes** pour chaque tâche pour définir les comportements des rôles.

L'analyse se poursuit par la conception, qui se développe jusqu'à la problématique du déploiement en quatre étapes :

4. **Assigner les rôles** à des classes d'agents spécifiques, et identifier les conversations en examinant les modèles de tâches concurrentes en se basant sur les rôles joués par chaque classe ;
5. **Construire les conversations** en extrayant les messages et les états définis pour chaque voie de communication dans les modèles de tâches concurrentes, en ajoutant des messages et des états pour plus de robustesse ;
6. **Définir les classes internes d'agents** grâce à une architecture de chaque classe d'agent en utilisant des composants et des connecteurs. Il faut alors s'assurer que chaque action apparaissant dans une conversation est bien implémentée comme une méthode au sein de l'architecture d'agent ;
7. **Définir la structure finale du système** en utilisant des diagrammes de déploiement.

Ce processus est valide pour n'importe quel contexte de développement (nouvelles applications, *re-engineering*, ou conception avec réutilisation). MaSE a le net avantage de couvrir la totalité du cycle de vie du système, malgré le manque de gestion de qualité. La définition des étapes est très précise et les différents modèles à produire et délivrables sont clairement identifiables.

La pragmatique. De nombreux articles et rapports sont dédiés à MaSE, ainsi, et grâce à agentTool, il devient facile de l'utiliser pour développer un système multi-agent. De plus, malgré la large couverture du processus, MaSE n'impose l'utilisation d'aucun langage de programmation particulier – bien que le cadre d'implémentation *agentMom* soit écrit en Java – et s'applique à n'importe quel domaine. Cependant, le concepteur devra certainement maîtriser la logique temporelle afin d'exploiter au mieux le potentiel des modèles proposés. De plus, les architectures développées sont statiques, fermées et ne permettent pas de définir des sociétés à grand nombre d'agents.

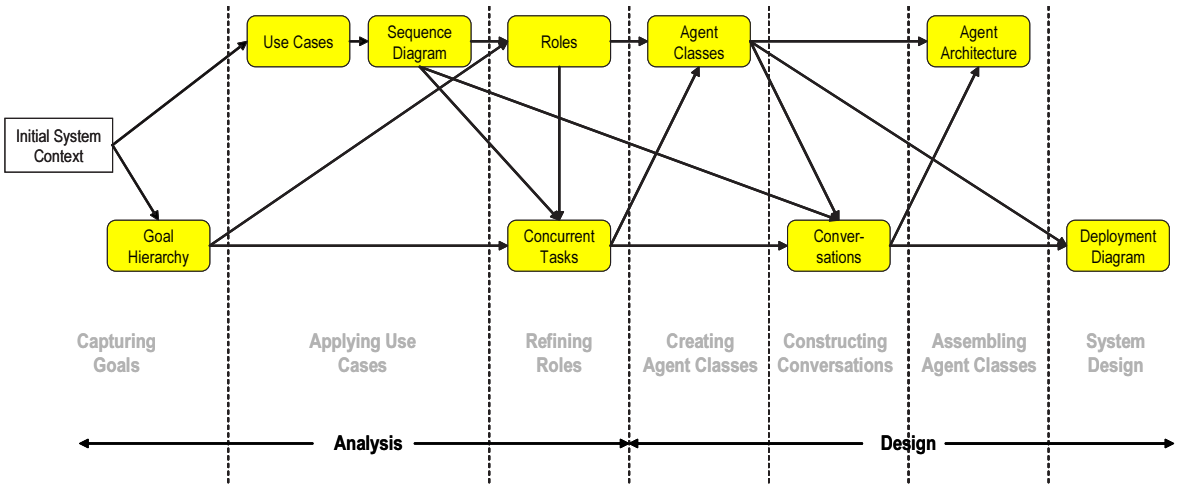

Figure 2.12 — Les différentes étapes et les modèles de MaSE.

2.4.2.7 MASSIVE

Lind propose une vision très différente avec la méthode MASSIVE, pour *Multi Agent SystemS Iterative View Engineering* [Lind, 2001]. Il s'oppose aux visions classiques d'ingénierie multi-agent en pointant la nécessité de se munir de méthodes basées sur les besoins plutôt que sur la technologie, sur les vues du le système plutôt que sur les modèles et proposant un processus itératif plutôt que séquentiel.

Les concepts clés. MASSIVE repose sur le concept de *vues*. Lind définit une vue comme un ensemble de caractéristiques liées conceptuellement au système cible, c.-à-d. une projection du modèle complet sur un sujet particulier. Dans MASSIVE, un système consiste en sept vues.

La *vue de l'environnement* rassemble les deux dimensions environnementales dans les systèmes multi-agents : l'environnement du point de vue du concepteur (connaissance globale et déterminisme) et l'environnement du point de vue du système (connaissance locale et non-déterminisme).

La *vue des tâches* présente une hiérarchie des fonctionnalités du système et définit les besoins non-fonctionnels du système indépendamment de l'approche multi-agent.

La *vue des rôles* détermine l'agrégation fonctionnelle pour résoudre le problème en fonction des contraintes physiques. Un *rôle* est défini comme une abstraction qui lie la partie dépendante du domaine de l'application à la technologie agent utilisée pour résoudre le problème. Un *agent* représente une ou plusieurs descriptions de rôles et une architecture capable de les exécuter.

La *vue des interactions* présente le système comme un ensemble d'entités devant se coordonner pour atteindre leurs *buts* et identifie les interactions essentielles. Une *interaction* est une forme générique et générale de résolution de conflit, non restreinte à de la communication.

La *vue de la société* présente un ensemble d'entités au but commun et nécessite de définir un

modèle de société, en fonction de mesures de qualité et de performance, cohérent avec les rôles et atteignant les buts définis.

La *vue architecturale* se focalise sur l'architecture du système et l'architecture des agents en fonction des besoins du problème, avec la possibilité de réutiliser une architecture existante préférée. Elle permet d'éprouver la frontière entre agents et objets.

Enfin, la *vue du système* gère l'interface avec les utilisateurs : interactions utilisateur/système, gestion des E/S, visualisation de l'activité du système, stratégie de gestion des erreurs, de la performance et du déploiement du système.

L'autonomie des agents est assurée par la séparation des vues concepteur et système. Les agents encapsulent leurs buts et rôles. La gestion des événements procure la réactivité, mais la pro-activité est difficilement spécifiable, si ce n'est en passant par de nombreux diagrammes comportementaux. Les aspects sociaux sont complètement pris en charge dans les vues de la société et des interactions.

Les notations. MASSIVE réutilise de nombreuses notations, et notamment UML. Par exemple, les modèles de la vue des tâches peuvent être décrits grâce à des diagrammes de cas d'utilisation (aspects fonctionnels), des arbres de décomposition de tâches (comme dans MaSE), des diagrammes d'activités pour décrire les tâches et les participants ou des graphes pondérés pour décrire les relations entre tâches. Les protocoles de la vue des interactions sont principalement décrits par des machines à états. Les hiérarchies de la vue de la société sont représentées par des diagrammes de classes ou d'objets simplifiés. L'architecture est décrite par des diagrammes de classes, de composants, de séquence ou de collaboration.

Utilisant UML, et n'y ajoutant que peu de formalisme, les notations de MASSIVE peuvent être caractérisées de peu précises. Par contre, elle gagne en modularité (héritage de la conception OO), en exécutabilité (prototypage rapide), en analysabilité (outils existants) et en accessibilité.

Le processus. Le processus de MASSIVE est très bien décrit dans le livre de Lind [Lind, 2001]. C'est un processus itératif dans sa globalité, comme le montre la figure 2.13, mais par contre, chaque vue est construite séquentiellement. Lind décrit son processus global comme suit :

1. **Sélectionner une vue** parmi les vues proposées ;
2. **Sélectionner un processus de niveau micro** pour assister le développeur dans le raffinage de la vue sélectionnée. Ces processus sont séquentiels et définis différemment pour chacune des vues ;
3. **Vérifier la base d'expériences** afin de réutiliser des modèles ou paquetages pré-définis, s'il y a lieu ;
4. **Appliquer** le processus de micro niveau pour générer une première vue ou raffiner une vue pré-existante (résultant d'une itération précédente ou se trouvant dans la base d'expériences) ;
5. **Evaluer**, grâce à des mesures de qualité, la vue courante. Si la qualité est insuffisante, réappliquer le processus de micro-niveau jusqu'à ce que les mesures de qualités soient correctes ;

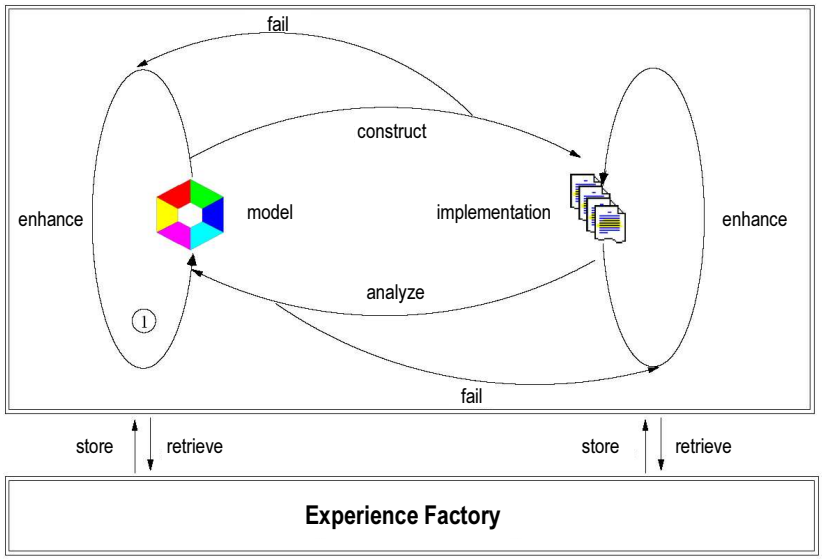

Figure 2.13 — Le processus itératif de la méthode MASSIVE.

6. **Implémenter** la vue si elle a atteint un degré de maturité suffisant, afin de produire du code qui sera mis dans la base d'expériences ;
7. **Tester** afin de voir si le code respecte les spécifications de la vue en trois étapes :
   (a) **sélectionner le cas de test** ;
   (b) **exécuter le test** ;
   (c) **adapter** le code si nécessaire ;
8. **Re-construire** le code si les tests ne sont pas concluants ;
9. **Itérer** le processus dans son intégralité jusqu'à l'obtention d'un système adéquat ;
10. **Mettre à jour la base d'expériences** pour des utilisations futures.

Ce processus, bien que très différent des approches classiques par phases, permet de couvrir l'intégralité du cycle de développement, des besoins au déploiement, en passant par des phases de test et validation. Il s'applique dans n'importe quel contexte et ne dépend d'aucune architecture d'agent précise, ni d'un langage de programmation particulier. Toute les activités et les étapes d'analyse et conception des différentes vues sont très clairement expliquées, mais les délivrables sont trop peu décrits.

La pragmatique. Lind présente deux cas d'étude complets dans son livre (le cas d'un assistant de voyage personnel et une application de recherche de documents distribués), ce qui reste bien suffisant pour appréhender MASSIVE, malgré le peu de ressources disponibles [Lind, 2001]. Aucune expertise particulière n'est nécessaire, même si l'originalité du processus peut freiner l'apprentissage et la compréhension de la méthode. MASSIVE n'est dédiée à aucun langage ou domaine d'application, compte tenu du large éventail de concepts manipulés.

2.4.2.8 MESSAGE

MESSAGE, qui signifie *Methodology for Engineering Systems of Software AGEnts*, est le fruit d'un projet entre compagnies de télécommunication européennes, porté par EURESCOM et développé par Caire *et al.* [EURESCOM, 2000; Caire *et al.*, 2001; Coulier *et al.*, 2004]. Bien que le projet n'ait duré que deux ans, MESSAGE est toujours en cours de développement, notamment avec sa continuité espagnole, INGENIAS, développée par Gomez Sanz et Fuentes [Gomez Sanz et Fuentes, 2002]. MESSAGE étend la notation UML et s'intègre dans le RUP [Jacobson *et al.*, 1999].

Les concepts clés. Comme Gaia, MESSAGE définit des modèles de représentation des concepts orientés agent. MESSAGE distingue les concepts de niveau d'abstraction des données (conception classique) des connaissances (conception de systèmes intelligents). Les concepts de niveau connaissance sont regroupés en trois catégories :

Les entités concrètes qui sont les *agents*, l'*organisation*, les *rôles* et les *ressources*. Les agents doivent fournir des *services*, comme des objets fournissent des méthodes. Un attribut des agents, appelé *objectif* (*purpose* en anglais), similaire à la notion de but, encapsule la notion d'autonomie de contrôle et de décision des agents. Un agent est capable de percevoir des *messages* et des *entités informatives*. L'organisation représente un groupe d'agents reliés par des relations de pouvoir. Dans MESSAGE, le rôle d'un agent est vu comme une interface dans la conception objet, ce qui permet à un agent de jouer plusieurs rôles. Les ressources sont des objets classiques non autonomes et plus ou moins disponibles en fonction des permissions établies ;

Les activités sont soit des *tâches* soit des *interactions*. Une tâche est une activité atomique possédant des pré et des post-conditions, qui sont modélisées par des machines à états. Des tâches composées peuvent être définies en utilisant des relations de composition. Les interactions sont très proches de celles proposées dans Gaia. Une interaction a plusieurs participants devant atteindre un but commun. Des protocoles d'interaction peuvent être définis comme des ensembles de schémas de *messages* permettant de mettre en place une interaction ;

Les entités d'états mentaux sont les *buts* qui associent des agents à des *situations*. Les buts peuvent être activés ou non dans la mémoire de l'agent. Un agent ayant un but actif cherchera à atteindre ce but, c.-à-d. à se retrouver dans une situation spécifiée par le but.

Ces concepts, lors de l'analyse et de la conception, sont manipulés dans des vues (ou modèles), apparaissant dans la figure 2.14 :

La vue de l'organisation montre les entités concrètes dans le système et son environnement, et la relation de haut niveau comme l'agrégation, le pouvoir ou les accointances (qui implique la définition d'interactions) ;

La vue des buts/tâches montre les buts, les tâches, les situations et les relations entre eux. Les dépendances temporelles peuvent être définies, ainsi que les dépendances d'ordre compositionnel ;

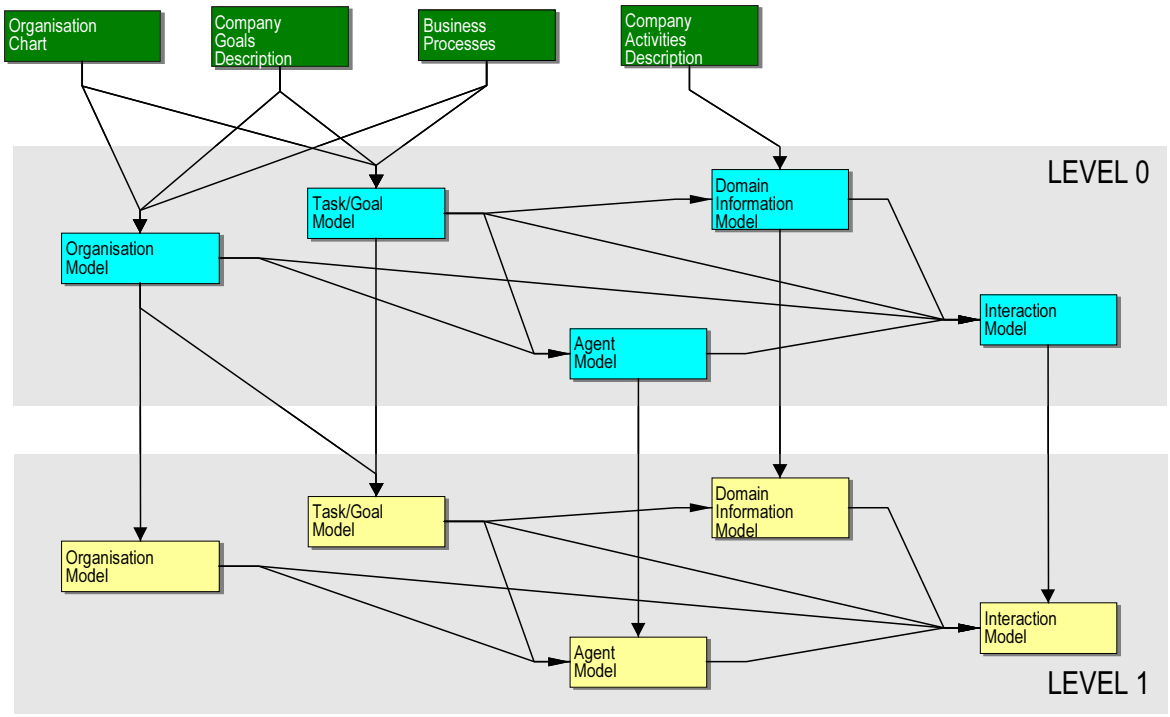

Figure 2.14 — Les deux niveaux d'analyse de MESSAGE et les modèles associés.

La vue des agents/rôles associe les rôles aux agents, en spécifiant les événements déclenchant l'attribution de rôles aux agents et les ressources exploitées ;

La vue des interactions définit, pour chaque interaction agent/rôle, l'initiateur, les collaborateurs, les "motivateurs", les informations échangées, les événements déclenchant les interactions, et les autres effets possibles comme l'attribution de nouveaux rôles par exemple ;

La vue du domaine est une vue classique en analyse orientée objet qui décrit le domaine d'application du système en terme d'entités et de relations entre elles.

Comme dans Gaia, l'autonomie des agents est assurée par l'encapsulation des buts. La réactivité des agents est obtenue grâce à la spécification d'événements et d'actions dans les modèles d'agent/rôle et d'interaction. De même, la pro-activité est reliée aux buts. Les concepts sociaux propres aux systèmes multi-agents sont exprimés, tels, par exemple, les notions d'organisation ou de collaboration.

Les notations. MESSAGE propose une extension d'UML, non pas en utilisant les stéréotypes (comme dans A-UML par exemple), mais en modifiant le métamodèle UML par profil.

Ainsi, la vue de l'organisation est représentée par des diagrammes d'organisation qui sont des diagrammes de classes dans lesquels apparaissent les ressources, les organisations, les rôles, des classes du domaine et où certaines relations sont des accointances. Les décompositions de buts sont représentées dans des diagrammes de classes en utilisant des relations de compositions spécifiques (AND ou OR). La vue d'agent/rôle peut être représentée par des diagrammes de délégation. Les tâches sont représentées par des diagrammes de flots de tâches, qui sont des diagrammes d'activités UML, dans lesquels apparaissent les tâches,

les services, les organisations et les rôles. Enfin, des descriptions textuelles, sous forme de tables, rassemblent les informations sur les rôles. La vue des interactions est représentée par des diagrammes d'interaction (collaboration), des protocoles de type AIP (*Agent Interaction Protocol*) de A-UML, et des descriptions textuelles, comme dans Gaia. Enfin, le domaine, comme en analyse orientée objet, est représenté grâce à des diagrammes de classes.

MESSAGE hérite des avantages et des inconvénients d'UML, qui n'est pas un langage formel et qui perd donc en précision, mais gagne en modularité, exécutabilité (avec A-UML et certains travaux sur le prototypage rapide), en analysabilité (grâce aux outils UML). Par contre, UML n'est pas le plus adéquat pour représenter les connaissances (comme dans DESIRE) ou les raisonnements logiques.

Le processus. MESSAGE s'intègre dans un processus de développement orienté objet connu, le RUP (*Rational Unified Process*). En l'état, MESSAGE ne fournit que des modèles d'analyse. INGENIAS a notamment pour but de continuer le travail sur les activités de conception [Gomez Sanz et Fuentes, 2002].

Ici, l'analyse est séparée en deux niveaux, comme le montre la figure 2.14, ayant chacun pour but de construire des modèles spécifiques ou de les raffiner. Le niveau 0 correspond à la vue externe (macro-niveau) de AAII ou Cassiopée[12], et le niveau 1 à le vue interne (micro-niveau) :

Niveau 0 d'analyse :

1. **le modèle d'organisation** décrit la structure et le comportement d'un groupe d'agents ayant un but commun. Au niveau 0, ce modèle procure une vue du contexte dans lequel le système opère en termes d'agents et de relations de pouvoir ;
2. **le modèle buts/tâches** décrit comment les buts/tâches de haut niveau sont décomposés en sous-buts/sous-tâches de niveau inférieur. Au niveau 0, il contient un ensemble de tâches et de buts de l'organisation dans laquel le système est plongé ;
3. **le modèle d'agent** consiste en une description des objectifs, des relations et des autres attributs des agents et des rôles. Au niveau 0, il modélise le système multi-agent comme s'il était un agent unique ;
4. **le modèle du domaine** encapsule les informations sur le domaine d'application ;
5. **le modèle d'interaction** saisit les moyens d'échange d'information entre les agents, entre eux ou avec leur environnement. Au niveau 0, le modèle définit l'ensemble des interactions possibles entre le système multi-agent et l'organisation externe dans laquelle le système est plongé ;

Niveau 1 d'analyse :

6. **le modèle d'organisation** est raffiné à ce niveau, notamment par l'analyse des autres modèles préliminaires ;
7. **le modèle buts/tâches** est précisé et les tâches sont détaillées, grâce à l'analyse du modèle d'organisation des niveaux 0 et 1 ;

[12]Pourtant, ici, le processus est clairement *top-down*, ou descendant, contrairement à Cassiopée.

8. **le modèle d'agent** est une décomposition du modèle de niveau 0 en un ensemble d'agents capables de fournir le même ensemble de services qu'au niveau 0. Cet ensemble est dérivé de l'organisation de niveau 1 ;
9. **le modèle du domaine** étend le modèle de niveau 0 par l'ajout d'objets et de ressources nécessaires au bon fonctionnement des nouveaux agents identifiés ;
10. **le modèle d'interaction** permet de spécifier les échanges entre les nouveaux agents identifiés et de préciser les accès aux nouvelles ressources.

Comme MESSAGE a fait le choix d'UML, elle gagne à s'intégrer dans le RUP. En effet, ce processus est reconnu et utilisé dans l'industrie car fortement couplé au paradigme objet. Il est adéquat dans n'importe quel contexte et propose des mesures d'assurance qualité et de gestion de projet. De plus, tout le cycle de développement est couvert, des besoins à la maintenance, et ses étapes sont bien définies et couplées aux artefacts et délivrables à produire. Cependant, MESSAGE ne fait que modifier les activités d'analyse pour prendre en compte la problématique agent, sans se soucier des éventuels besoins technologiques nécessaires lors de la conception ou de l'implémentation. L'autre désavantage d'un tel choix et la non prise en compte des concepts agents lors des phases de vérification et de validation.

La pragmatique. Côté pratique, le RUP surclasse les autres approches. En effet, les ressources sont nombreuses (livres, logiciels, sites) et son utilisation ne demande pas une grande expertise. MESSAGE ne propose aucune directive d'implémentation, laissant libre le codeur de choisir son langage, à condition qu'il soit orienté objet. Par contre, INGENIAS est lui fortement dirigé vers Java. Des outils sont disponibles (Rational Rose ou MetaEdit), ce qui facilite les phases d'analyse et conception.

2.4.2.9 PASSI

PASSI de Cossentino, qui signifie *Process for Agent Societies Specification and Implementation*, est une méthode pas-à-pas intégrant des modèles de conception et concepts provenant de l'ingénierie orientée objet et de l'intelligence artificielle en utilisant UML [Cossentino, 2001; Cossentino et Potts, 2002]. Le processus de développement est facilité par PTK qui est composé d'une extension à Rational Rose et d'un outil de ré-utilisation de *patterns* d'agents.

Les concepts clés. Le métamodèle d'agent de PASSI manipule beaucoup de concepts communs aux agents et y ajoute les notions provenant des contraintes FIPA pour être adapté à FIPA-OS ou JADE [Bernon *et al.*, 2003b]. Un agent peut jouer plusieurs *rôles* impliqués dans des *scénarios* et pouvant fournir des *services*. Un rôle remplit au moins une *tâche*, pouvant être une tâche au sens de la FIPA. Les rôles sont aussi des médiums d'information par *message*. Les agents sont définis, par rapport au domaine du problème, par des *ressources* et des *besoins* (fonctionnels ou non).

Ces notions sont encapsulées dans cinq modèles (voir figure 2.15) : le modèle des besoins, le modèle de société d'agent, le modèle d'implémentation d'agent, le modèle de codage et le modèle de déploiement. L'autonomie des agents est liée aux différents rôles qui leur sont

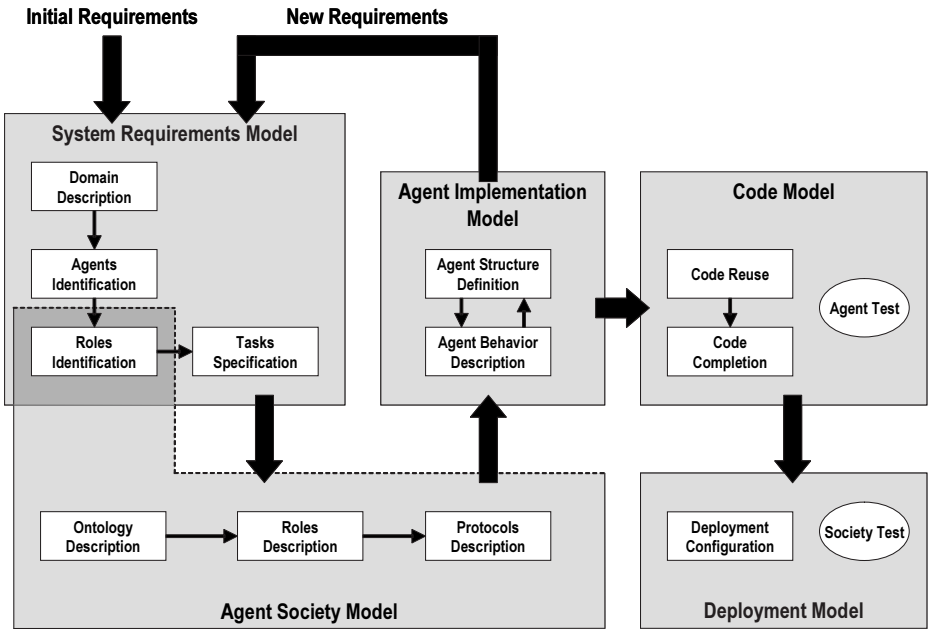

Figure 2.15 — Les cinq modèles/phases de la méthode PASSI.

attribués. La réactivité des agents est définie dans un ensemble de modèles comportementaux, ainsi que la pro-activité. Les aspects sociaux sont intégrés dans le modèle de société d'agents.

Les notations. PASSI réutilise fortement UML. Par exemple, le modèle des besoins est exprimé grâce à des diagrammes de cas d'utilisation, de paquetages stéréotypés, de séquences (associés aux cas d'utilisation) et d'activité. Le modèle de société d'agent utilise des diagrammes de classes et de séquences. Les modèle d'implémentation d'agent et de codage utilise des diagrammes de classes et d'activités classiques. Enfin, le modèle déploiement utilise des diagrammes de déploiement UML.

L'utilisation d'UML, comme pour MESSAGE par exemple, implique le manque de formalisme (et donc de précision), et les connaissances ne peuvent représentées qu'en utilisant des diagrammes de classes (que PASSI nomme ontologies dans son métamodèle).

Le processus. Le processus de PASSI est un processus pas-à-pas, mais réitérable. Il est composé de cinq phases, composées d'étapes, correspondant aux spécifications des cinq modèles présentés ci-dessus (voir figure 2.15) :

Les besoins

1. **la description du domaine** permet d'exprimer les besoins et de décrire le contexte du système grâce à une hiérarchie de cas d'utilisation ;
2. **l'identification des agents** est une étape peu présente dans les méthodes orientées agent, et est effectuée par regroupement en paquetages, qui correspondront aux agents ;

3. **l'identification des rôles** se fait par l'analyse des interactions entre agents dans des diagrammes de séquences. Cette étape est commune aux besoins et à la définition de la société d'agent. En effet, les rôles sont à la fois une description fonctionnelle et comportementale des agents, mais aussi une représentation des relations entre agents ;
4. **la spécification des tâches** permet de définir les activités des agents en les reliant aux fonctionnalités fournies par le système ;

La définition de la société d'agents

5. **l'identification des rôles** (voir plus haut) ;
6. **la description de l'ontologie** permet de décrire la société d'agents d'un point de vue ontologique suivant deux axes : ontologie du domaine (diagramme de classes appelé diagramme d'ontologie du domaine) et l'échange d'information entre agents (diagramme de classes appelé diagramme d'ontologie de communication) ;
7. **la description des rôles** se fait grâce à des diagrammes de classes et de paquetages et permet de décrire les relations entre rôles d'un point de vue hiérarchique et organisationnel grâce à un ensemble de règles sociales.
8. **la description des protocoles** de communication se fait grâce à des diagrammes de séquence ou de protocoles A-UML, suivant les standards FIPA par exemple ;

L'implémentation des agents

9. **la définition de la structure des agents** permet de définir les constituants des agents en termes de ressources internes, de connaissances (attributs) et de tâches (méthodes) ;
10. **la définition du comportement des agents** est obtenue grâce à des diagrammes d'activité où apparaissent les tâches d'un agent et leurs interactions ;

Le codage

11. **la réutilisation du code** peut être possible en cas de développements pré-existants ;
12. **la complétion du code** peut se faire à travers des outils dédiés et permet de rendre exécutable le code des agents ;

Le déploiement

13. **la configuration du déploiement** est nécessaire lorsque l'on spécifie des agents mobile. Cette étape permet de définir la topologie du réseau et l'attribution/permission d'accès aux ressources pour les agents.

Des besoins au déploiement, les étapes de PASSI sont très clairement définies ainsi que les modèles à fournir tout au long du processus. PASSI est une méthode pouvant s'appliquer à n'importe quel domaine, mais est plutôt orientée agent mobile, avec notamment les aspects respectueux des spécifications FIPA. Le manque de formalisme, cependant, rend les phases de tests peu efficaces.

La pragmatique. PASSI est riche en ressources, et possède notamment un site[13] expliquant le processus pas-à-pas ainsi que tous les modèles. Aucune expertise particulière n'est recommandée bien que la connaissance, non obligatoire, des spécifications FIPA soit préférable pour exploiter au mieux la méthode. PASSI n'est reliée à aucun langage particulier, bien que l'outil PTK (*PASSI ToolKit*) soit orienté vers Java afin d'être compatible avec JADE et FIPA-OS. Tous les diagrammes et stéréotypes spécifiques sont intégrés dans une extension au logiciel Rational Rose, afin de réaliser les modèles d'analyse et conception de manière ergonomique.

2.4.2.10 Prometheus

Padgham et Winikoff ont développé Prometheus comme une méthode complète (méthode, notations et outils) pour des agents BDI [Padgham et Winikoff, 2002, 2003], bien qu'aujourd'hui elle ait évolué et se dise indépendante de toute architecture. C'est une méthode de seconde génération, qui profite des avancées des méthodes précédentes, et s'inspire notamment de Gaia ou MaSE. Prometheus est le fruit d'une expérience de programmation avec la plate-forme Jack, ce qui implique certains choix de conception.

Les concepts clés. On retrouve dans Prometheus les même concepts que dans Gaia ou MaSE. Les agents sont des agents à gros grains, BDI et fortement *pro-actifs*. Ils peuvent faire partie de *groupes* (fixés), possèdent des accointances de travail, avec lesquels ils interagissent suivant des *protocoles* spécifiés, fournissent des *services* (ce sont leurs actions) et ont des capacités (des modules). Les agents peuvent envoyer des messages et planifier leurs actions en fonction de *buts*. Bien sûr, ils possèdent des données ou *croyances* sur les autres agents.

L'autonomie des agents est assurée par l'encapsulation des buts et des plans. Bien que la notion de groupe soit avancée, les notions de rôles sont inexistantes, et leur gestion non abordée. Par contre la sûreté est assurée lors de regroupements (en terme de droits d'accès). Une notion intéressante, et peu présente dans les autres méthodes, est la notion d'*incident* pouvant survenir en cours de fonctionnement. Cette notion reste cependant floue, et semble correspondre à la notion d'exception.

Les notations. Prometheus définit des notations dédiées et ré-utilise UML et A-UML. Nous pouvons distinguer sept notations spécifiques. Des *descripteurs textuels* permettent de spécifier des fonctionnalités du système, des scénarios, ou des agents à partir de champs textuels (comme les buts, les actions, les interactions possibles). Les *diagrammes de couplages de données* permettent de regrouper les fonctionnalités du systèmes aux données produites en termes de production, de modification ou d'exploitation. Les *diagrammes d'accointances d'agent* décrivent les interconnexions entre agents de différents types. C'est un diagramme de classes UML simplifié où les agents sont représentés par des classes (sans compartiment) et les connexions sont représentées par des associations binaires avec multiplicité. Les *diagrammes de vue du système* relient les agents, les événements et les objets partagés dans un même modèle en utilisant une notation dédiée. Les *diagrammes d'interaction*

[13] `http ://mozart.csai.unipa.it/passi/`

montrent les interactions entre agents. Prometheus exploite les diagrammes de séquence UML et A-UML pour modéliser les protocoles. Les *diagrammes de vue des agents* sont similaires aux diagrammes de vue du système mais montrent les interactions entre capacités au sein d'un agent. Chaque message, perception ou action identifié au niveau du système doit être pris en compte à ce niveau. Les *diagrammes de vue des capacités* montrent les interactions et contraintes sur les actions et événements entre les plans et les capacités.

Ces notations sont claires, sémantiquement bien définies, et donc faciles à utiliser et à apprendre. Basées sur le paradigme objet, la modularité est prise en compte et la gestion de la complexité est rendue possible par la décomposition des agents en activités et des activités en sous-activités. Comme Prometheus est couplée à Jack, le langage Java est préférable pour l'implémentation.

Le processus. Le processus de Prometheus, ainsi que les modèles produits, sont décrits dans la figure 2.16. Le processus se décompose en trois phases majeures :

1. **La spécification du système** voit l'identification des buts et sous-buts du système (sous forme de listes) par le développement des scénarios de cas d'utilisation. A ce niveau il est alors possible d'identifier les interfaces avec l'environnement (actions et perceptions). Les fonctionnalités du système sont regroupées dans un descripteur préliminaire de capacités, ce qui nécessite, en parallèle, la définition des données de travail associées grâce à des diagrammes de couplage de données.
2. **La conception architecturale** groupe les fonctionnalités identifiées à la phase précédente. Ces fonctionnalités permettent d'identifier des agents auxquels les affecter. Les interactions entre agents, données et perceptions sont alors spécifiées dans un diagramme de vue du système. Un diagramme de couplage est aussi utilisé pour spécifier les relations entre données et fonctionnalités regroupées. Les interactions sont aussi décrites grâce à des diagrammes de séquence.
3. **La conception détaillée** commence par la vue interne des agents, qui est spécifique à l'architecture choisie (principalement BDI, si l'on veut bénéficier des outils proposés). Il convient alors de préciser ou de raffiner toutes les composantes internes aux agents – capacités, plans, croyances et données – dans des diagrammes de vue d'agent et de capacités.

Prometheus couvre une grande partie du cycle de développement, même si les besoins ne sont que peu abordés. Les phases de test, validation, déploiement et maintenance sont totalement absentes, ainsi que la gestion de qualité et de conduite de projet. Néanmoins, Prometheus convient à des applications orientées utilisateurs, comme Tropos, et non à la résolution de problèmes ou à la simulation. Les activités sont assez bien décrites ainsi que les artefacts à produire.

La pragmatique. Les ressources sur Prometheus sont très limitées, les articles présentent bien l'approche, mais il est difficile pour un concepteur de développer sans les créateurs à ses côtés. Seules quelques applications non industrielles ont été d'ailleurs développées.

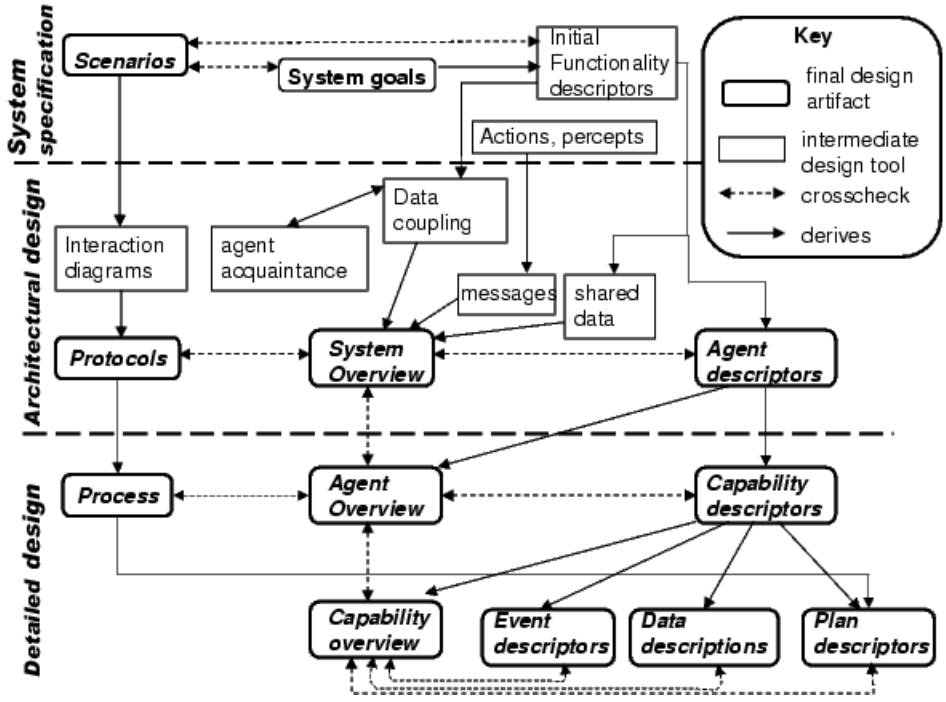

Figure 2.16 — Les trois phases de Prometheus et les modèles manipulés.

Par contre, Prometheus est associé à deux outils : JDE et PDT. JDE (*Jack Development Environment*) permet d'éditer les modèles proposés pour les porter vers la plate-forme Jack. Seulement quelques vérifications de consistance sont faites sur certains modèles. PDT (*Prometheus Design Tool*) manipule les descripteurs et vérifie partiellement les interactions entre objets du modèle [Padgham et Winikoff, 2002].

2.4.2.11 TROPOS

Tropos[14] est une méthode de développement fondée sur les concepts utilisés en ingénierie des besoins [Castro *et al.*, 2001]. Elle manipule la notion d'agent et de propriétés mentales associées tout au long du processus de développement. Tropos adopte une approche transformationnelle, dans le sens où, à chaque étape, les modèles vont être raffinés de manière itérative par ajout ou suppression d'éléments ou relations dans les modèles.

Les concepts clés. Les concepts manipulés par Tropos sont les même que dans $i^\star$ (voir paragraphe 2.3.2), à savoir : l'*acteur* (BDI de préférence), le *rôle*, la *position*, le *but* (*soft* ou *hard*), le *plan*, la *ressource* et la *dépendance* (avec *depender* et *dependee*). D'autres notions sont manipulées comme les *capacités* d'un acteur à définir, choisir et exécuter un plan pour remplir un but et les *croyances* qui sont une représentation de la connaissance d'un acteur sur le monde. Le but est de définir les obligations des acteurs envers les autres acteurs.

Tropos définit une modélisation selon cinq points de vues complémentaires :

[14]Tropos est déviré du terme grec *tropé* signifiant "facilement modifiable ou adaptable".

- *Social* : quels sont les différents acteurs et que font-ils ? Quelles sont leurs obligations et leurs capacités ?
- *Intentionnel* : quels sont les buts adéquats et comment sont-ils reliés ? Comment sont-ils remplis et par qui des dépendances sont-elles demandées ?
- *Communicationnel* : comment les acteurs dialoguent-ils et comment interagissent-ils ?
- *Orienté procédé (processus)* : quels sont les processus (business/computer) adéquats ?
- *Orienté objet* : quels sont les classes et objets adéquats et quels sont leurs liens ?

Dans Tropos, l'autonomie est exprimée grâce aux buts des acteurs. La réactivité et la proactivité sont prises en compte en modélisant les plans, les activités et les événements. Les notions sociales d'organisation ou de groupe ne sont pas directement abordées.

Les notations. Tropos utilise principalement $i^\star$ pour modéliser les systèmes. Cette notation été introduite dans le paragraphe 2.3.2. L'utilisation de diagrammes d'états/transitions est nécessaire à la modélisation des capacités (diagrammes de capacités) et des plans (diagrammes de plan). Les interactions entre agents sont spécifiées grâce à des diagrammes de protocole A-UML.

Une approche plus formelle, appelée *Formal Tropos*, propose des description textuelles et logique d'expression des besoins et des possibilités de traduction automatique vers $i^\star$ [Perini *et al.*, 2003]. Ceci ajoute de la précision aux notations et langages. Les modèles de Tropos sont simples, accessibles et très expressifs. La gestion de la complexité passe néanmoins par plusieurs modèles (et non pas par des zooms comme avec DESIRE). Tropos a développé un outil de génération de prototypes vers la plate-forme Jack, ce qui ne réduit pas ses capacités de portabilité, bien que l'architecture agent soit imposée (agents BDI). Enfin, de nombreux outils permettent d'analyser les spécifications.

Le processus. Tropos, comme MESSAGE, a l'avantage de couvrir une grande partie du cycle de développement logiciel multi-agent. Cinq phases sont clairement définies :

L'analyse des besoins initiaux permet une compréhension du problème par l'étude d'une organisation existante et produit un modèle organisationnel (acteurs et leurs dépendances). La modélisation des buts et des dépendances entre acteurs s'effectue via des diagrammes d'acteurs où un nœud représente un acteur, et un lien représente une dépendance. Les buts sont précisés comme étant *hard* (dont les conditions de satisfaction sont clairement identifiées) ou *soft* (dont la satisfaction est sujet à interprétation). $i^\star$ est utilisé pour exprimer les dépendances entre acteurs et buts ainsi qu'une description formelle. Les sorties de cette phase sont un modèle de dépendances stratégiques (acteurs, buts et dépendances) et un modèle de "rationale" stratégique (moyen d'atteindre les buts collectivement) ;

L'analyse des besoins finals fournit une description du système étudié dans son environnement opérationnel et modélise le système en tant qu'ensemble d'acteurs possédant des dépendances. Ces dépendances définissent les besoins fonctionnels ou non fonctionnels du système. Cette phase fournit les modèles précédents révisés en incluant en premier le système en tant qu'acteur. Un nouveau modèle "rationale" est produit pour cet acteur ;

La conception architecturale définit l'architecture globale du système en termes de sous-systèmes interconnectés par des flots de données et de contrôle. Ces sous-systèmes sont représentés comme des acteurs et les interconnexions comme des dépendances entre acteurs. Plusieurs styles d'architectures (structure plate, pyramidale, ...) sont donnés et évalués par rapport à certains critères (adaptabilité, intégrité, coopération, ...) identifiés au niveau des besoins finals. Ceci conduit à la production d'un diagramme des besoins non fonctionnels et à une révision des précédents modèles ;

La conception détaillée définit chaque composant architectural en termes d'entrées, de sorties, de contrôle, etc. Cette phase permet de détailler la communication entre acteurs et le comportement des acteurs en différents niveaux. Un niveau détaille un dialogue intra-acteur ou inter-acteurs. Le niveau le plus bas décrit le traitement interne d'un acteur via des diagrammes de plans. Chaque plan identifié est décrit dans un diagramme d'états-transitions. Lors de cette phase, l'utilisation d'UML est possible, notamment par la définition de stéréotypes spécifiques à Tropos. L'utilisation d'A-UML, de KQML et des FIPA-ACL permet de spécifier les interactions entre agents.

L'implantation fournit une architecture d'agents BDI sous la plate-forme Jack.

Tropos est adéquat dans n'importe quel contexte et couvre la totalité du cycle de développement, même la vérification et la validation sont abordées de manière formelle. Par contre, les activités et les délivrables associés au sein des phases ne sont pas si clairement définis (la lecture de nombreux articles est nécessaire). Contrairement à MESSAGE, et au RUP en général, Tropos ne touche pas aux aspects qualité et gestion de projet.

La pragmatique. Malgré les nombreux articles sur Tropos, qui est développée par une équipe très productive, il est difficile de rassembler les morceaux, et peu d'outils existent. Ici encore – architecture BDI oblige – des notions de logique temporelle peuvent être nécessaires. Tropos est plutôt conçue pour des systèmes de *e*-Business ou de gestion partagée des connaissances. En effet, aucune réelle identification des agents n'est précisée. Ainsi, les acteurs de l'environnement seront toujours représentés par des agents. Des applications sans acteurs humains – du type simulation ou résolution de problèmes – sont proscrites.

2.4.2.12 Voyelles

L'approche Voyelles de Demazeau est une méthode de haut niveau (peu de directives techniques sont données) mais très souvent citée car reposant sur des principes purement multi-agents [Demazeau, 1995, 2001a,b, 2003]. Elle repose sur la décomposition de la vue d'un système suivant quatre[15] dimensions (ou lettres) : **A**gent, **E**nvironnement, **I**nteraction et **O**rganisation [Boissier et Demazeau, 1996; Ricordel et Demazeau, 2000, 2002; da Silva et Demazeau, 2002].

Les concepts clés. L'approche Voyelles repose principalement sur trois déclarations [Demazeau, 1995] :

[15] Récemment, une cinquième voyelles a été ajoutée pour l'**U**tilisateur [Demazeau, 2003].

Equation déclarative :	$SMA = A + E + I + O$
Equation fonctionnelle :	$F_{SMA} = \sum_i F_{agent_i} + F_{Collective}\ (+F_{Emergence})$
Principe de récursion :	$Entité = SMA \mid Entité\ basique$

L'équation déclarative exprime la décomposition d'un système multi-agent suivant les quatre axes de Voyelles. Le ***A*** pour *agents* correspond aux éléments pour définir les entités actives du système (architectures internes, représentation des connaissances, etc.). Dans Voyelles, ces agents peuvent être de nature et de granularité quelconques. Le ***E***, pour *environnement*, dans le sens des domaines de simulation ou de la physique, contient les éléments de description de l'entité commune partagée par les agents. Le ***I*** pour *interactions* contient les éléments pour structurer les interactions entre les agents (ACL, AIP A-UML, etc.). Le ***O*** pour *organisation* prend ces sources dans la sociologie ou la psychologie sociale, et contient les éléments pour structurer, contraindre les agents avec, par exemple, des structures organisationnelles, des *normes*, ou des *lois*. Demazeau précise aussi que l'*autonomie* (naturelle ou artificielle) et la délégation de contrôle doivent être affectées à au moins une des Voyelles (pas forcément la même) [Demazeau, 2003].

L'équation fonctionnelle exprime le fait que la fonctionnalité d'un système multi-agent n'est pas seulement la somme (ou la composition comme dans §1.1.2), mais que la composante collective due aux interactions et la composante émergente du système en sont indissociables (bien que le choix du symbole $+$ reste discutable, car permettant cette dissociation).

Le principe de récursion permet de concevoir les systèmes multi-agents comme des agents, et inversement. Ainsi, ceci permet d'analyser un système multi-agent suivant deux visions : centrée système et centrée agent. L'équation déclarative peut alors s'interpréter au niveau agent : $Agent = a + e + i + o$. La composante ***a*** de l'agent correspond aux compétences, capacités et raisonnements. La composante ***e*** spécifie les actions et perceptions que l'agent peut avoir sur son environnement. La composante ***i*** désigne la gestion des interactions de l'agent, avec la compréhension des discussions et la connaissance des protocoles. Enfin ***o*** peu s'interpréter comme le raisonnement social de l'agent, c.-à-d. la représentation que l'agent a sur les normes, les organisations et les moyens de les gérer.

Cette approche permet un jeu d'écriture par associativité, afin d'exprimer le type d'approche choisie pour le développement d'un système, ou bien un domaine d'application. Par exemple, Demazeau attribue à l'informatique une approche $(((A + E) + I) + O)$ dans laquelle les agents et l'environnement sont les priorités de développement (approche centrée agent/environnement), alors que les sciences de l'économie se caractériseraient plutôt comme $(((O + I) + E) + A)$.

Voyelle est une méthode de haut niveau d'abstraction, et donc les concepts comme comme l'autonomie, la réactivité, la pro-activité ou les aspect sociaux ne sont pas "techniquement" expliqués, mais pré-supposés.

Les notations. Voyelles laisse les concepteurs libres d'utiliser les formalismes, les notations ou langages de leur choix pour spécifier chaque lettre du système. Cependant quelques

exemples permettent de les guider. Pour modéliser les agents, on peut par exemple utiliser des logiques formelles, comme avec le système de vision ASIC [Boissier et Demazeau, 1996], les arbres ou les graphes [Van Aeken, 1999], des automates, etc. La modélisation de l'environnement peut elle aussi passer par des formalismes très différents. Si l'on reprend le travail de Van Aeken, elle passe par l'expression de différentes configurations d'arbres, telle que l'organisation. Comme dans de nombreuses autres méthodes, les protocoles exprimés par des diagrammes de séquences ou des machines à états sont un moyen d'exprimer des interactions.

Ainsi Voyelles n'est couplée à aucune notation particulière, et les critères de comparaison que nous nous sommes donnés concernant cet aspect méthodologique ne sont pas applicables.

Le processus. Voyelles suit un processus de développement logiciel classique, comme le montre la figure 2.17, dans lequel les concepts Voyelles se greffent à tout moment.

L'**analyse** porte sur le domaine d'application et le type du problème, et consiste en la décomposition du problème en voyelles, c.-à-d. les différentes composantes du système. A un plus haut niveau d'abstraction, les entités identifiées devront répondre aux principes de récursion ou d'émergence. A la fin de l'analyse, le problème est donc "Voyellé".

La **conception**, dans Voyelles correspond à l'identification des modèles à utiliser pour chacune des voyelles. Ici encore, les principes de haut niveau (récursion et émergence) doivent être utilisés. Dans des travaux antérieurs, les modèles proposés sont [Occello *et al.*, 1998, 2001] :

- ASTRO pour les agents hybrides ou PACORG pour les agents réactifs ;
- les langages à protocoles (IL) ou les forces de PACO pour les interactions ;
- les organisations statiques de RESO.

La **programmation** (ou implémentation) consiste en l'instanciation des modèles, en utilisant des plates-formes et des langages choisis. Le résultat, le système implémenté, peut alors être exécuté, évalué (test et validation) et repensé en cas d'inadéquation avec les besoins exprimés par le type de problème et le domaine d'application.

Ce processus est complet, mais comme son niveau d'abstraction est élevé, aucune des phases n'est réellement détaillée. Par conséquent, les délivrables ne sont pas décrits, car leur nature peut différer d'une application à une autre. La vérification et la validation sont prises en compte, avec par exemple l'outil AGIP de vérification de protocoles. La gestion de projet et de qualité sont absentes dans Voyelles.

La pragmatique. Voyelles est certainement la méthode qui se distingue le plus des autres, et elle est difficilement comparable aux autres, du moins à partir des critères choisis. Ici, la priorité est la liberté totale de choix – ce qui est toutefois problématique lorsque l'on veut enseigner les moyens de concevoir des systèmes, ce qui est le but, au final, d'une méthode de développement. Le niveau de détail est trop faible pour l'utiliser sans connaissance approfondie de l'approche. De plus, les ressources sont quasi-inexistantes. Le seul moyen de savoir quels modèles utiliser, comment décomposer son problème, ou quel outils choisir est de parcourir les travaux (la plupart étant des thèses) ayant utilisé la méthode. La plate-

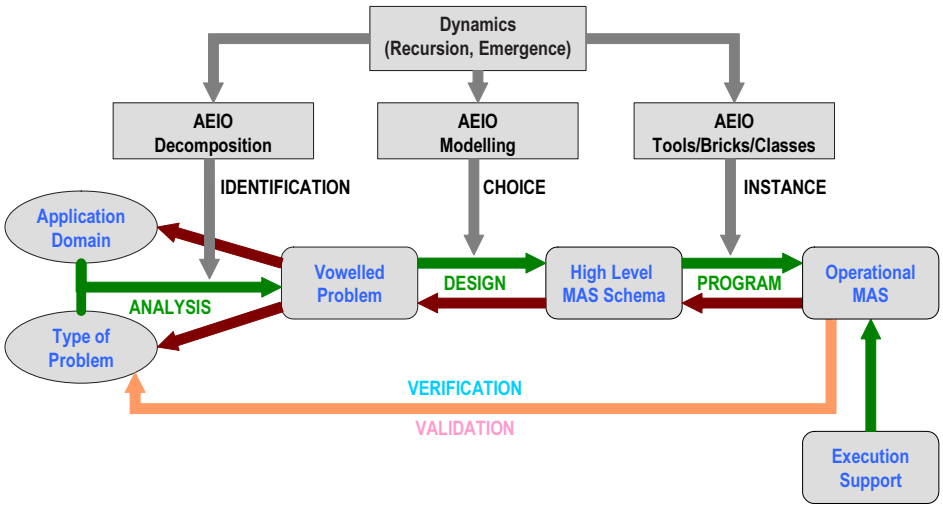

Figure 2.17 — L'approche générale de la méthode Voyelles.

forme MASK (*Multi-Agent System Kernel*) est développée pour fournir des outils de développement Voyelles et permet l'ajout de modèles. Elle fonctionne sous UNIX et les librairies pour les boîtes à outils sont écrits en C, C++ et Java. Les agents sont distribués en utilisant les communications UNIX (TCP/IP), le système Xenoops ou le WEB. Cependant elle reste à l'état de prototype et son portage reste difficile. Enfin, le développement et le déploiement des systèmes peuvent être assurés par la plate-forme Volcano [Ricordel, 2001].

2.5 Analyse de la comparaison et expression des besoins

Le tableau 2.2 montre une synthèse de la comparaison des méthodes présentées dans le paragraphe précédent (§2.4).

2.5.1 Lecture suivant l'axe des méthodes

Lire la table suivant l'axe des méthodes (de gauche à droite) nous indique, si une notion méthodologique donnée est largement prise en compte par les méthodes orientées agents existantes. On peut constater que les interactions ou les notions de rôles sont communément abordées. Par contre, des notions comme l'adaptation, l'ouverture ou l'environnement sont très souvent absentes. L'autonomie reste à "une moyenne de +" seulement, compte tenu de la partialité de cette propriété, qui ne repose souvent que sur des présupposés ou des encapsulations (artificielles) de capacités cognitives.

Concernant les notations, cette lecture reste difficile car aucune notation ne semble universelle. Seules les méthodes proposant plusieurs types de notations ou langages, sont polyvalentes, comme MaSE ou Tropos. Voyelles se plaçant à un très haut niveau d'analyse, et ne préconisant aucune notation, n'est pas analysable par rapport à cette rubrique. Deux courants principaux se dégagent : les notations de type UML qui gagnent en modularité, analysabilité et complexité, et les notations et langages formels plus précis et vérifiables.

Toutes les méthodes affichées se concentrent principalement sur les phases d'analyse et

		AAII	Aaladin	Cassiopée	DESIRE	Gaia	MaSE	MASSIVE	MESSAGE	PASSI	Prometheus	Tropos	Voyelles
Concepts & Propriétés	*Adaptation*	-	+	+	-	-	-	±	±	±	-	-	+
	Autonomie	+	+	+	+	+	+	+	+	+	+	+	±
	Buts	+	±	-	+ +	±	+ +	+ +	+ +	±	+ +	+ +	±
	Croyances	+ +	±	-	+ +	±	+	±	-	+	+ +	+ +	+
	Désirs	+	±	-	+	±	+ +	+	+	+ +	+ +	+	+
	Environnement	- -	±	-	-	±	-	+	-	±	+	±	+
	Groupe	-	+ +	+	-	±	-	±	±	+	+	±	+
	Intention	+	±	-	+	±	±	±	±	±	+ +	+ +	+
	Interaction/Message	+	+ +	-	+	+	+ +	+ +	+ +	+ +	+ +	+	+ +
	Norme	+	+ +	+	+	+	+ +	+	±	-	±	+	+
	Organisation	+	+ +	+ +	- -	+ +	±	+	+ +	+	+	+	+ +
	Ouverture	- -	-	+	- -	- -	±	±	±	±	-	- -	+
	Pro activité	+	+	±	+	+	+	±	+	+	+	+	±
	Réactivité	+	+	+ +	+	+	-	±	+	+	±	+	±
	Rôle	-	+ +	+ +	+	+ +	+ +	+ +	+ +	+ +	-	+ +	+
	Tâche	-	±	±	+ +	+	+ +	+ +	+ +	+ +	+	±	+
Notations & langage de modélisation	*Accessibilité*	+	+	+	±	±	±	±	+	+	+	+	?
	Analysabilité	+	+ +	±	+	- -	+	+	+	+	+ +	+ +	?
	Complexité	+	+ +	±	+	- -	±	+	+	+	+	-	±
	Consistance	+	±	+	+	+	±	±	±	+	+ +	+ +	?
	Exécution	-	+	+	+	- -	+ +	+	+	+ +	+	+ +	+
	Expressivité	+	+	+	±	±	±	±	±	±	+	±	-
	Modularité	+ +	+	±	+	+ +	±	+	+ +	+ +	+	+	?
	Portabilité	+	+	+ +	- -	±	±	±	±	+	±	-	+
	Précision	+	+	±	+ +	+ +	+ +	-	-	±	+	+ +	- -
	Raffinage	+	+	+	+	+	+ +	+ +	+	+	+	+ +	?
	Traçabilité	+	+	+	+	+	-	±	+	+	+	±	?
Processus de développement	*Contexte*	Tous	Tous (Proto.)	Tous (Proto.)	Tous (Proto.)	Tous	Tous	Tous	Tous	Tous	Tous	Tous	Tous
	Besoins	-	-	- -	-	-	- -	+	+	+	-	+ +	-
	Analyse	+ +	+ +	+ +	±	+ +	+ +	+ +	+ +	+ +	+ +	+ +	+ +
	Conception	+	+	+	+ +	+ +	+ +	+ +	+ +	+ +	+ +	+	+ +
	Implémentation	- -	+ +	- -	±	- -	+	+	+	+	+	+	+
	Test	- -	-	- -	+ +	-	±	+	±	±	-	±	+
	Déploiement	- -	+	- -	- -	- -	- -	+ +	±	+ +	- -	- -	+
	Maintenance	- -	-	- -	- -	- -	- -	±	±	±	- -	- -	- -
	Délivrables	+	-	±	-	+ +	+ +	+	+ +	+ +	+	-	- -
	Gestion de la qualité	- -	- -	- -	- -	- -	- -	-	+	- -	- -	- -	- -
	Gestion de projet	- -	- -	- -	- -	- -	- -	-	+	- -	- -	- -	- -
Pragmatique	*Ressources*	-	+	±	-	-	+	+	+	+ +	-	-	- -
	Expertise requise	Oui	Non	Non	Oui	Oui	Non	Non	Non	Non	Oui	Oui	Non
	Langage spécifique	Non	Oui	Non	Oui	Non	Non	Non	Non	Non	Non (Java)	Non	Non
	Domaine d'application	Non	Simulation	Simulation	Connais-sances	Gros Grains	Non	Non	Non	FIPA	Utilisateurs	Utilisateurs	Non
	Scalabilité	Oui	Oui	Non	Non	Oui	Non	Oui	Oui	Oui	Oui	Non	Oui

Notation : (++) pour les propriétés pleinement et explicitement prises en charge ; (+) pour les propriétés prises en charge de manière indirecte ; (±) pour des propriétés potentiellement prises en charge ; (−) pour des propriétés non prises en charge ; (−−) pour des propriétés explicitement non prises en charge.

Tableau 2.2 — Synthèse de la comparaison des différentes méthodes

de conception. Tropos est un cas d'exception concernant les besoins ; ceci s'explique par sa filiation à l'ingénierie des besoins avec $i^\star$. Les autres exceptions sont des méthodes dont le processus s'inspire de processus orientés objets éprouvés, comme le RUP avec MESSAGE, ou le processus itératif de MASSIVE. Cette limitation à ces deux phases peut s'expliquer de deux façons :

- La technologie agent n'en est qu'à ses débuts et certains concepts, comme l'autonomie ou la socialité restent difficiles à implanter ;
- Le concept d'agent n'a peut-être aucune spécificité par rapport à l'objet en ce qui concerne l'implantation.

La première hypothèse est assez convaincante. En effet, ceci implique aussi le manque de formalisme et donc de validation. Peu de définitions communément admises existent. Même le concept d'agent est sujet à polémique. La seconde hypothèse semble par contre discutable. En effet, implanter des agents comme des objets, nécessite de se munir de règles de transformation spécifiques. De plus, le domaine de la programmation orientée agent, qui propose de nouveaux paradigmes de programmation, non forcément inspirés de l'objet, souligne la nécessité de se munir de nouveaux langages de programmation afin de palier les difficultés des concepts agents et multi-agents.

Enfin, les méthodes orientées agents manquent cruellement de ressources. Rares sont celles proposant des tutoriels, des pages internets ou des livres synthétiques.

2.5.2 Lecture suivant l'axe des notions méthodologiques orientées agents

Lire la table 2.2 suivant l'axe des notions méthodologiques orientées agents/multi-agents nous indique si une méthode donnée aborde tous les concepts et propriétés, possède des notations et langages efficaces, propose un cycle de développement complet et clair, et enfin si cette méthode est facilement exploitable. Globalement, un premier constat est qu'aucune méthode n'est "parfaite" dans ce sens. Chaque méthode possède ses spécificités en termes d'architecture, de formalisme ou de modèles. Ces différences sont souvent dues aux filiations des méthodes – ingénierie des connaissances, des besoins, orientée objet, ou simulation.

Toutefois, certaines méthodes sont plus générales – ou inversement plus spécialisées – que d'autres. Par exemple, DESIRE, qui est une des méthodes les plus anciennes, est spécialisée dans la conception de prototypes pour des systèmes de connaissances. De même, Tropos sera plutôt utilisée pour des systèmes orientés ergonomie et utilisateurs ou systèmes d'information, compte tenu de la richesse de son analyse des besoins. Mais ceci implique aussi une architecture BDI d'agents à gros grain. A l'opposé, MESSAGE s'intégrant dans le RUP, est assez générale et convient à tout type de contexte. PASSI, bien qu'ayant fait le choix contraignant de la FIPA, propose un processus et des notations riches et expressives.

2.5.3 Bilan et expression des besoins pour une méthode des AMAS

Nous avons vu dans ce chapitre comment la communauté agent avait adapté des techniques existantes pour la conception de systèmes multi-agents. Les courants majeurs sont l'ingénierie objet, l'ingénierie des connaissances, l'ingénierie des besoins, la simulation et la

reproduction de comportements ou phénomènes naturels.

L'ingénierie objet est l'influence majeure, car elle apporte un paradigme de programmation proche de la notion d'agent. L'agent peut alors être vu comme une extension de l'objet classique – cette extension pouvant être obtenue par stéréotypage ou profil, par exemple. La communauté objet apporte aussi son expérience en terme de développement, en fournissant des processus et des outils de support. De nombreuses méthodes orientées agents ont donc naturellement fait ce choix d'exploitation de l'objet, comme par exemple MaSE, MESSAGE ou PASSI, empruntant notations et directives. Certaines notations ont même été modifiées pour mieux répondre aux besoins agents comme l'illustre A-UML, concernant principalement les interactions entre agents. Cette approche orientée objet a le net avantage d'être connue des développeurs non spécialistes, et est manipulée via un langage maîtrisé.

Cependant, tous les concepts agents ne cadrent pas forcément au monde objet – ou du moins restent difficiles à modéliser de la sorte – notamment en ce qui concerne les capacités cognitives des agents, leurs ontologies, ou leur socialité. DESIRE s'inspire de l'ingénierie des connaissances et propose un méthode fortement orientée vers des agents à gros grains pour la gestion des connaissances. De même, Tropos est dédiée à l'analyse des besoins pour des systèmes orientés utilisateurs. Aalaadin repose sur des notions organisationnelles permettant de spécifier des systèmes par leur structures inter-agents. Enfin, Cassiopée qui est une des seules méthodes ascendantes – qui part des parties – s'attache à décrire les comportements internes des agents avant de définir le comportement collectif. Néanmoins, Cassiopée ne propose aucun processus de développement complet, avec une analyse des besoins, nécessaire lorsque l'on n'a pas l'expérience des créateurs. Ces techniques sont peu connues des développeurs traditionnels et reste donc difficiles à introduire dans l'industrie par exemple.

Les approches inspirées par la nature sont encore plus marginales et ne proposent que des techniques à appliquer au cas par cas. La difficulté réside dans le choix des paramètres et leur corrélation aux variables du système à modéliser.

Chacune de ces approches a ses avantages et ses inconvénients. De plus, l'analyse des méthodes existantes nous montre qu'aucune n'est parfaite. Si nous voulons proposer une méthode dédiée aux AMAS, facilement accessible et enrichie d'outils, plusieurs questions doivent être abordées et leurs réponses formeront notre cahier des charges.

Doit-on s'intégrer dans une des méthodes existantes ou repartir de zéro ? Concernant l'intégration dans une méthode orientée agent existante, replaçons-nous au début de notre investigation, en 2000, où cela semblait difficile de s'intégrer dans l'existant compte tenu de la notion d'agent coopératif spécifique aux AMAS. En effet, la plupart des méthodes de l'époque sont basées sur des architectures d'agents connues (BDI ou FIPA), avec DESIRE ou Gaia (alors à ses débuts). Il n'est pas impossible de faire correspondre les concepts des agents coopératifs à ces architectures, mais ceci risque d'apporter plus de difficultés de compréhension que de simplification, et donc ne pas faciliter l'assimilation des concepts. De plus, ces approches sont complètement antinomiques aux notions d'auto-organisation et d'ouverture car les organisations sont simples, fixées et forcément composées d'agent à gros grains. Le modèle AGR ne semble pas non plus convenir compte tenu de son approche organisationnelle *a priori*. Cassiopée, qui pourtant repose sur des notions proches de nos préoccupations, n'offre qu'un processus de

développement limité.

Il semble donc inutile de s'intégrer dans une méthode orientée agent existante, mais, par contre, il convient de s'interroger sur les méthodes objets. Les méthodes qui s'intègrent dans des processus venus du domaine objet, comme le fait MESSAGE en s'intégrant au RUP, possèdent le net avantage de couvrir tout le cycle de développement de logiciels. La plupart des processus pour la conception de systèmes multi-agents se focalisent sur l'analyse. Cependant, rien ne garantit que le concept d'agent ne doive apparaître qu'en début de cycle. L'assurance apportée par des processus déjà éprouvés et complets nous laisse ainsi le loisir de manipuler les spécificités agents à tout instant du processus. De plus, l'autre intérêt majeur est que de tels processus sont bien connus des développeurs – peut-être pas autant utilisés, du moins tels quels.

Notre choix sera donc de s'intégrer dans une méthode orientée objet connue, le RUP qui est connu et bien défini, afin de proposer plus d'accessibilité et de potentiel quant au développement de la notion d'agent.

Quels notations et langages doit-on adopter/créer/réutiliser ? On distingue majoritairement deux courants dans les méthodes analysées : celles reposant sur des notations graphiques et celles reposant sur des langages formels, principalement des logiques. Bien sûr, certaines proposent des transformations automatiques ou semi-automatiques de modèles graphiques vers des modèles formels et inversement. Cependant, la plupart des méthodes ayant choisi des langages formels ont fait des choix drastiques concernant le concept d'agent. Par exemple, DESIRE est une méthode très efficace pour développer des systèmes consistants mais très limités en nombre d'entités et en capacité calculatoire. De même, Tropos ne concerne que des agents BDI. Or, le concept d'agent coopératif, comme nous l'avons souligné dans le paragraphe 1.3, est peu formalisé et de haut niveau. Imposer des choix alors que le modèle manque de formalisme serait se couper des possibilités de développements futurs. L'utilisation de notations moins formelles que les logiques, comme UML, semble alors la meilleure solution. Mais ceci n'empêche en rien que certaines vues sur le système ou les agents soient formalisées, comme, par exemple, les protocoles d'interactions, ou bien les croyances et leur consistance.

Du point de vue de l'utilisateur de méthodes, le concepteur, faire le choix de notations connues semble être aussi une meilleure solution – le compromis devant se faire entre précision et expressivité. Compte tenu de ces remarques et de celles faites en amont concernant l'agent comme extension du concept d'objet, nous opterons pour l'utilisation d'UML, quitte à en établir une extension spécifique aux AMAS.

Existe-t-il des outils manipulant les notions avancées ou facilement modifiables/réutilisables ? Très peu de méthodes orientées agents proposent des outils d'aide à la conception. Souvent ce sont des outils de simulation ou des plates-formes pré-existantes de développement de systèmes multi-agents comme Jack ou JADE. Ces outils ne sont pas forcément une aide pour le développement de systèmes de A à Z et ne couvrent qu'une partie du processus.

La réponse à la question précédente allant vers le choix UML, nous nous intéresserons bien sûr aux outils proposés pour l'analyse et la conception objet en UML. PASSI propose une plate-forme intégrée à Rational Rose[16]. MESSAGE développe son propre outil

[16]PASSI s'intègre dans MetaEdit depuis peu.

appelé INGENIAS. Ici, le choix portera sur la facilité de modification de tels outils pour prendre en compte les agents coopératifs. Cette question reste donc sans réponse jusqu'à la définition des extensions que nous ferons au métamodèle UML pour intégrer les notions agents.

Pour conclure, afin de fournir à des concepteurs néophytes dans le domaine des agents, et plus spécifiquement les agents coopératifs, une méthode de développement assez complète, nous nous intégrerons dans le RUP, en y ajoutant des activités supplémentaires, comme l'identification des agents par exemple, qui fait souvent défaut aux autres méthodes. Cette intégration ainsi que la proximité du concept agent à l'objet, établit le langage UML comme le meilleur compromis notationnel. Enfin, nous chercherons à modifier des outils existants lorsque ce sera possible, voire à en créer le cas échéant, afin de rendre la manipulation des notations et le suivi du processus les plus simples possibles.

Deuxième partie

ADELFE : méthode de développement de systèmes multi-agents adaptatifs

3 Le processus d'ADELFE

PROPOSER une méthode de développement relève principalement d'une démarche à suivre pour atteindre des objectifs donnés. Le processus fourni doit permettre à tout concepteur d'effectuer un suivi pas-à-pas des étapes de développement de logiciels et des méthodes mises en jeu. Définir un processus consiste seulement en la définition de ses étapes (ou phases), mais aussi en leur ordonnancement et en l'expression des modèles et artefacts produits et nécessaires. Le processus d'ADELFE ne déroge pas à la règle, et nous l'avons ainsi défini [Gleizes *et al.*, 2003].

En l'état, nous nous sommes focalisés sur les premières phases de développement. Le processus se divise en quatre *définitions de travaux* : les besoins préliminaires, les besoins finals, l'analyse et la conception. Ces travaux s'intègrent dans le RUP et ont été étendus afin de répondre au métier agent, comme MESSAGE a pu le faire [Jacobson *et al.*, 1999; Coulier *et al.*, 2004][1]. Bien sûr, rien ne montre que le concept agent ne soit qu'une notion d'analyse et de conception. En s'intégrant dans le RUP, rien n'interdit la définition future de travaux d'implémentation dans lesquels la notion d'agent pourra prendre place, en utilisant, par exemple, la transformation de modèles selon MDA.

Chaque ensemble de travaux, ainsi que ses artefacts, est défini en utilisant la notation issue de SPEM (pour *Software Process Engineering Meta-model*) permettant de définir des processus de manière simple et compréhensible pour les utilisateurs d'UML [OMG, 2002]. Ce profil UML est brièvement exposé dans le paragraphe 3.1. Les ensembles de travaux du processus d'ADELFE sont développés dans les paragraphes suivants (§3.3 à §3.6).

[1] Voir le paragraphe 2.5 pour justification.

3.1 Le SPEM

Le SPEM (*Software Process Engineering Meta-model*) est un profil UML, et donc basé sur une approche objet, pour définir les processus et leurs composants [OMG, 2002]. Il fournit un ensemble minimal d'éléments de modélisation de processus, nécessaire pour décrire n'importe quel processus de développement logiciel, sans ajouter de modèles ou de contraintes spécifiques à des domaines ou disciplines, comme la gestion de projet ou l'analyse. Il utilise la notation UML.

3.1.1 Le SPEM, UML et le MOF

Les éléments de modélisation de SPEM sont décrits comme des concepts UML – bien que tous n'aient pas forcément été retenus comme pertinents. Cet ensemble de concepts constitue le noyau du MOF, ou *Meta Object Facility*. Ce dernier donne une définition circulaire à ce langage de modélisation. Le MOF a été adopté et standardisé par l'OMG et est annoncé comme un métalangage universel capable de décrire des langages comme UML, SPEM, ou d'autres modèles relationnels. Le MOF se divise en quatre niveaux standardisés, notés M_0, M_1, M_2 et M_3, comme le montre le tableau 3.1.

L'application réelle d'un processus se situe au niveau M_0. La définition générique du processus, comme par exemple le RUP, ou toute autre méthode ou modification spécifique de ce processus utilisée dans un projet donné se situe au niveau M_1. Ici, nous nous focalisons sur le métamodèle, au niveau M_2 servant de modèle au niveau M_1. Afin d'éviter un nombre infini de niveaux, le niveau M_3 est réflexif. Par conséquent, le MOF peut être défini par lui-même.

3.1.2 Définition de processus avec le SPEM

Les éléments de définition d'un processus aident à décrire comment un processus doit être exécuté. Ils décrivent le comportement général du processus en fonctionnement, et sont utilisés pour sa planification, son exécution et son observation. Un processus peut être vu comme une collaboration entre rôles pour atteindre un but ou un objectif. Pour guider sa représentation, l'ordre des activités peut être contraint. Le processus peut aussi être défini par sa "forme" au cours du temps, c.-à-d. la structure de son cycle de vie en terme de phases et d'itérations.

<table>
<tr><td>M3</td><td colspan="6">MOF</td></tr>
<tr><td>M2</td><td colspan="3">SPEM</td><td colspan="3">UML</td></tr>
<tr><td>M1</td><td>RUP</td><td>SI Method</td><td>…</td><td colspan="2">User model 1</td><td>…</td></tr>
<tr><td>M0</td><td colspan="3">Processes as really enacted on a given project</td><td>User data 1</td><td>User data 2</td><td>…</td><td>…</td></tr>
</table>

Tableau 3.1 — Les quatre niveaux du MOF.

Un Process est un ProcessComponent voué à être un processus complet en terme de couverture du développement logiciel. Un ProcessComponent est une partie de la définition interne du processus et pouvant être réutilisée avec d'autres ProcessComponents pour assembler un processus complet. Un ProcessComponent importe un ensemble non-arbitraire d'éléments de définition appelés ModelElements.

Une WorkDefinition (ou définition de travaux) est une Operation décrivant des travaux à accomplir dans le processus. Ses sous-classes sont Activity, Phase, Iteration et Lifecycle. Une WorkDefinition peut être composée d'autres WorkDefinitions et est reliée aux WorkProducts d'entrée ou de sortie par la classe ActivityParameter. Les travaux décrits dans une WorkDefinition utilisent les WorkProducts d'entrée et créent ou modifient les WorkProducts de sortie. Les dépendances entre deux WorkDefinitions sont exprimées par le lien de dépendance temporelle Precedes.

Une Phase est une spécialisation de WorkDefinition dans laquelle une précondition définit le critère d'entrée dans la phase et les buts à atteindre pour en sortir. Les Phases sont définies avec une contrainte supplémentaire de séquentialité ; i.e. elles sont exécutées suivant une série de dates limites étalées au cours du temps et avec un recouvrement minimal de leurs activités. Une Iteration est une WorkDefinition composée avec une date limite. Il faut noter que ces éléments ne décrivent en rien l'exécution en tant que telle : ce sont des éléments de la description du processus utilisés pour aider la planification et l'exécution de cette description.

Un WorkProduct, ou artefact, est toute chose produite, consommée ou modifiée lors du processus, comme une partie de document (textuel ou non), un modèle (UML par exemple) ou du code source. La portée des types de produits dépend du processus à modéliser.

A une WorkDefinition est attribué un ProcessPerformer représentant le rôle principal qui exécute ces travaux. Un ProcessRole est responsable d'un ensemble de WorkProducts et est une sous-classe de ProcessPerformer. Une dépendance de type Impacts agit d'un WorkProduct à un autre WorkProduct pour indiquer que la modification du premier peut invalider le second. Le ProcessPerformer représente une des abstractions du processus dans son intégralité, ou en partie, et est utilisé pour les WorkDefinitions qui n'ont pas de "propriétaire" spécifique. Un ProcessRole définit les responsabilités sur des WorkProducts et définit les rôles qui mettent en œuvre ou assistent à des activités spécifiques. Un ProcessPerformer agit sur des WorkDefinitions de haut niveau qui ne peuvent être attribuées à des ProcessRoles. A chaque WorkDefinition peuvent être associés une Precondition et un Goal qui sont des Constraints dans lesquelles les contraintes sont exprimées grâce à des expressions booléennes comme en UML avec les conditions de garde. Ces expressions booléennes observent les états des WorkProducts paramètres de la WorkDefinition ou d'une WorkDefinition englobante.

Une Activity est la principale sous-classe de WorkDefinition et décrit une partie de travail exécutée par un ProcessRole. Une Activity appartient à un seul ProcessRole mais d'autres peuvent venir l'assister. Une Activity peut être composée d'éléments atomiques appelés Steps qui sont décrits dans le contexte de l'Activity englobante, en termes de ProcessRoles et de WorkProducts qui l'utilisent. Dans le cas d'activités supportées par un individu ou un petit groupe, ce sera unProcessRole. Comme pour les WorkDefinitions, la dépendance entre une activité et une autre est établie grâce à la dépendance Precedes.

Après avoir identifiés les activités, il est possible de les regrouper en Disciplines qui sont des spécialisations de la classe Package d'UML, qui partitionnent les activités dans un processus en fonction d'une thématique. Partitionner les activités de cette manière implique que les guides, ou Guidances (voir 3.1.3), et WorkProducts en sortie soient eux aussi catégorisés. L'inclusion d'une Activity dans une Discipline est obtenue en utilisant la dépendance Categorizes avec la contrainte supplémentaire que toute Activity est catégorisée par exactement une Discipline.

3.1.3 Les guides

Les guides, ou Guidances, peuvent être associés à tout élément de modélisation SPEM dans le but de fournir aux utilisateurs (de la méthode) des informations plus détaillées. Les types possibles de Guidances dépendent de la famille de processus et peuvent être par exemple : des lignes de conduite, des techniques, des métriques, des exemples, des profils UML, des tutoriels, des formulaires ou des archétypes. Chaque Guidance est associé à une GuidanceKind, dont le nom indique son type. Une Technique est un algorithme précis et détaillé utilisé pour créer un WorkProduct. Les Techniques sont utiles aussi pour définir les compétences nécessaires à l'exécution d'activités. Un UMLProfile fournit des mécanismes qui spécialisent UML pour un langage cible comme C++, Java et Corba ou pour un domaine particulier, comme l'analyse ou la conception. Un ToolMentor montre comment utiliser un outil spécifique pour accomplir une tâche. Chaque ToolMentor est associé à un seul outil et hérite de l'association avec l'Activity dans laquelle il est utilisé. Une Guideline est une autre sorte de Guidance qui est un ensemble de règles et de recommandations établissant le comment un WorkProduct donné doit être présenté et organisé.

3.1.4 Pourquoi modéliser le processus d'ADELFE en SPEM ?

Au début de nos travaux, rien ne nous guidait vers l'utilisation de SPEM. Pourtant, ses notations nous ont semblé très claires et donc efficaces pour la diffusion de la méthode. Nous avons vu, dans le chapitre précédent, qu'un grand nombre de méthodes orientées agent ne se souciaient guère de ces considérations, ce qui peut être, en partie, un des freins à l'utilisation des systèmes multi-agents par les non-spécialistes.

La grande difficulté a été de lire entre les lignes des spécifications du SPEM [OMG, 2002] qui jusqu'alors n'avait été aucunement utilisé. ADELFE a représenté, alors, un des seuls exemples d'application de SPEM. Nous avons notamment proposé à d'autres équipes travaillant sur les méthodes orientées agent, comme PASSI, de modéliser leur processus grâce à SPEM dans le but de fournir un cadre de description commun dans le cadre du groupe de travail sur les méthodes de FIPA[2] [Cossentino *et al.*, 2003].

[2]FIPA Methodology Technical Committee – `http://www.fipa.org/activities/methodology.html`

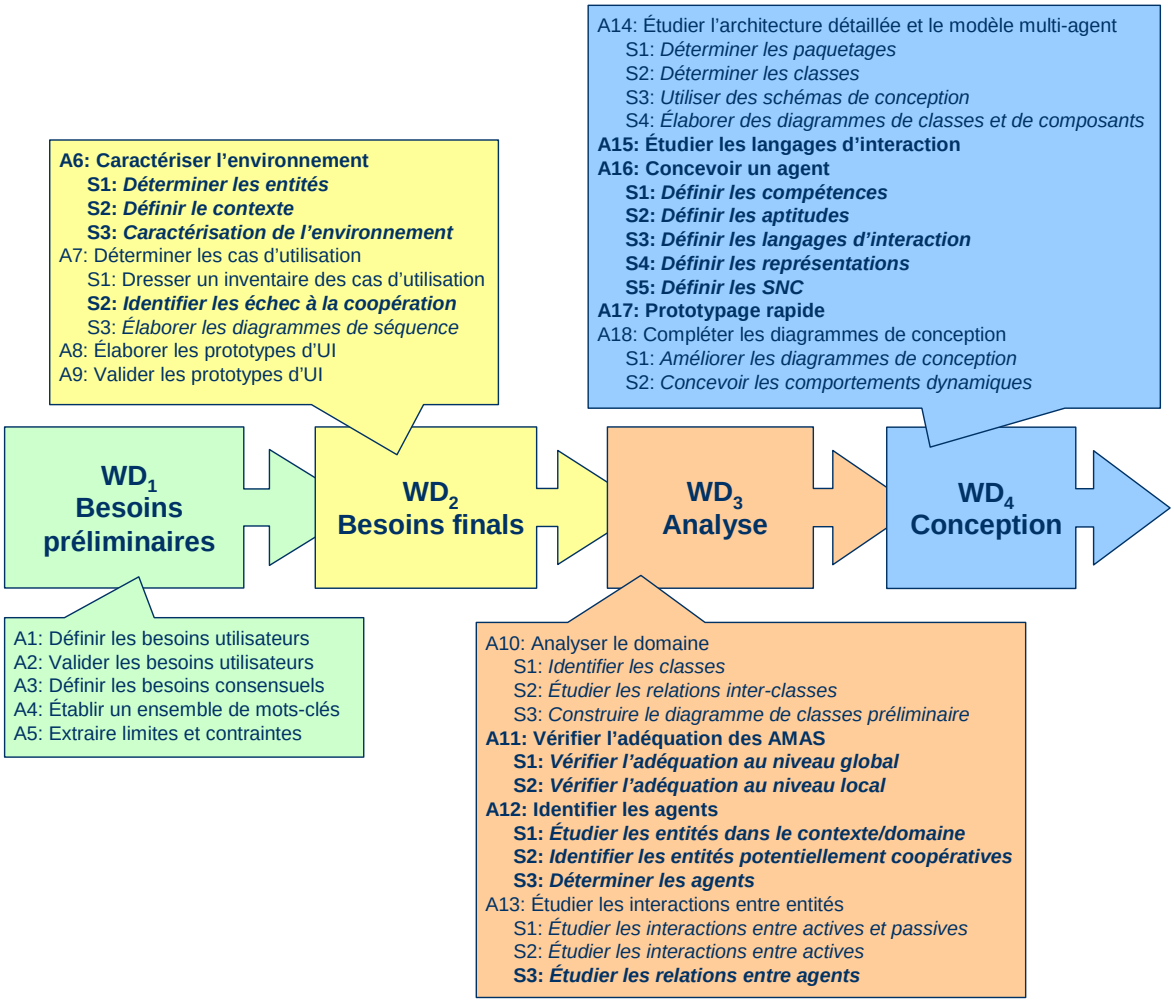

Figure 3.1 — Les quatre premières définitions de travaux du processus d'ADELFE.

3.2 Survol du processus

ADELFE suit donc le RUP (modifié selon le projet Neptune[3]) et les phases "classiques" : expression des besoins (préliminaires et finals), analyse et conception qui doivent être étudiées afin de concevoir un système. Des phases supplémentaires telles que l'implantation et le déploiement n'ont pas encore été étudiées, mais le seront certainement dans une version ultérieure d'ADELFE afin de permettre une étude complète du cycle de vie logiciel.

Le processus ADELFE a été exprimé en SPEM et le vocabulaire du SPEM a été adopté [Gleizes *et al.*, 2003]. Nous parlerons ainsi de :

- Définitions de travail (WorkDefinition ou WD_i) : pour les phases du RUP (Expression des besoins, Analyse et Conception) ;
- Activités (Activity ou A_j) : décomposition d'une WD ;
- Étapes (Step ou S_k) : étapes d'une activité.

[3] `http ://neptune.irit.fr/index1.html`

3.2.1 Spécificités d'ADELFE

ADELFE est spécifique à la conception d'AMAS ; aussi, certaines activités (marquées en gras dans la figure 3.1) ont été ajoutées au RUP pour l'adapter à cette technologie :

Durant l'expression des besoins finals (WD_2) :

- A_6 : *Caractérisation de l'environnement*. En effet, il semble nécessaire de se concentrer, dès les premières analyse des interactions entre le système et son environnement, sur la nature de ces interactions car ce sont elles qui guideront l'auto-organisation du système et qui influeront sur la définition des règles de coopération ;
- A_7-S_2 : *Identification des échecs à la coopération*. Cette étape est une suite logique du point précédent. Une fois que les propriétés de l'environnement ont été identifiées, il faut définir quelles sont les interactions entre acteurs (environnement) et cas d'utilisations (système) par lesquelles les problèmes d'inadéquation entre système et environnement passeront.

Durant l'analyse (WD_3) :

- A_{11} : *Vérification de l'adéquation aux AMAS*. Certes, nous proposons de développer des systèmes dont le fonctionnement adaptatif repose sur l'auto-organisation par coopération, mais tous les problèmes n'ont pas forcément besoin de les utiliser pour être résolus. De plus, même si leur utilisation semble utile, l'analyse ne sait pas forcément où placer des agents coopératifs, ou bien quelles sont les entités du domaine qui devront/pourront être décomposées en agents ;
- A_{12} : *Identification des agents impliqués dans le système à construire*. Les agents ne sont par forcément évident à identifier dans des systèmes non orientés utilisateurs. En effet, dans des systèmes de courtage en ligne, par exemple, les agents sont directement associés aux utilisateur. Mais dans un problème comme la prévision de crue, la présence d'agents horaires, devant produire des prévision pour une heure, n'est pas une identification directement extractible de l'analyse des entités du domaine. Il nous a donc semblé nécessaire d'ajouter une telle activité d'identification des agents à partir de l'analyse des interactions inter et intra système ;
- A_{13}-S_3 : *Étude des relations entre ces agents*. Comme pour les objets, avec les diagrammes de séquences ou de collaboration, les interactions entre agents doivent être étudiées. Cependant, les agents peuvent faire preuve de plus de richesse dans leur protocoles de communication, et donc cela nécessite l'ajout d'une activité spécifique devant produire des modèle d'interaction entre agents (protocoles A-UML) ;

Durant la conception (WD_4) :

- A_{15} : *Étude des langages d'interaction* qui permettent aux agents d'échanger de l'information. Tous les agents ne sont par forcément capables de communiquer avec tous les autres. Il faut définir des langages d'interaction entre agent portant sur des aspect de résolution de problème particulier. Ces langages d'interaction peuvent eux aussi être spécifier à partir de diagramme de protocole : un langage correspond alors à un ensemble de méthodes et d'attributs nécessaires pour comprendre les autres agents, et à une ou plusieurs machines à état décrivant la bonne utilisation du langage ;

A_{16} : *Conception complète de ces agents*. Un agent qui intervient dans un AMAS est composé de différentes parties qui forment son comportement : des compétences, des aptitudes, un langage d'interaction, des représentations du monde et des Situations Non Coopératives (SNC). L'ajout de cette activité est essentiel à la conception de système multi-agent adaptatifs, puisque cela est l'attribution d'un comportement nominal, pour résoudre une tâche courante, et d'un comportement coopératif aux agents afin de résoudre le problème voulu. C'est donc dans cette activité que les règles d'auto-organisation coopérative vont être définies et attribuées aux agents ;

A_{17} : *Prototypage rapide*. Éventuellement, afin de vérifier que le comportement des agents est bien celui désiré, cette activité permet de simuler les agents, non finalisés. Cette étape peut notamment servir de pré-validation ou de trouver des SNC non encore identifiées par l'observation et l'établissement de post-théories.

Le processus ADELFE peut se résumer par le schéma de la figure 3.1 (qui n'est pas exprimé en SPEM). Les paragraphes suivants détaillent chacune des WD_i et le SPEM est utilisé pour exprimer les séquentialités des activités et les productions/modifications d'artefacts grâce à des diagrammes d'activités.

3.2.2 Un exemple tutoriel : ETTO

Dans les paragraphes suivants, nous allons illustrer les différentes activités et étapes grâce à un exemple tutoriel : ETTO, pour *Emergent TimeTabling Organization* [Bernon *et al.*, 2002a; Peyruqueou, 2002; Picard *et al.*, 2004; Picard et Gleizes, 2004]. Le problème choisi est l'établissement d'emploi du temps dans lequel les créneaux horaires et les lieux (les salles) doivent être assignés à des enseignants et à des étudiants afin de construire un emploi du temps. Habituellement, les solutions à de tels problèmes utilisent différentes techniques comme les résolutions par satisfaction de contraintes ou les techniques métaheuristiques (recuit simulé, recherche avec tabous, coloration de graphes, etc.) et plus récemment les réseaux de neurones formels, les algorithmes évolutionnaires ou colonies de fourmis [Dréo *et al.*, 2003; Burke et Petrovic, 2002; Socha *et al.*, 2002, 2003]. Cependant, aucune technique, solution réelle, n'existe lorsque les contraintes peuvent évoluer dynamiquement et lorsque le système a besoin de s'adapter à l'environnement, si ce n'est en relançant la résolution à chaque changement. Comme ce problème n'est pas un problème de simulation, comme il est complexe (appartenant à la classe des problèmes NP-complets) et n'a aucune solution universelle, il s'avère représenter un bon exemple d'application des AMAS. Le but est de faire émerger la solution, au macro-niveau, à partir des interactions des parties indépendantes au micro-niveau.

Pour illustrer cet exemple, nous présenterons des diagrammes utilisant les notations d'ADELFE, qui s'inscrivent dans une approche UML. Le chapitre 4 fournit plus de détails concernant les spécificités de ces notations.

3.3 Les besoins préliminaires (WD_1)

Les besoins préliminaires représentent un travail d'intercompréhension et/ou une description consensuelle du problème du cahier des charges entre clients, utilisateurs et concepteurs sur ce que doit être et ce que doit faire le système, ses limites et ses contraintes. Cette phase doit permettre de transformer le cahier des charges ou des besoins exprimés par le client en un cahier des charges consensuel entre client et fournisseur (cf. figure 3.2). Dans un contexte multi-agent adaptatif, le système à étudier doit s'adapter à des événements imprévus qui peuvent provenir de l'environnement. Par rapport à une méthode orientée objet, ADELFE ajoute donc une caractérisation de cet environnement après ces travaux des besoins préliminaires. Par contre, les travaux des besoins préliminaires ne manipule pas le terme *agent*. En effet, ADELFE se propose d'être une méthode applicable à n'importe quel contexte, et ne peut donc limiter l'expression des besoins en y ajoutant des éléments spécifiques aux agents coopératifs.

Les activités des besoins préliminaires et leur ordonnancement sont résumés dans le diagramme d'activité SPEM de la figure 3.2. Les acteurs de cette activité sont le client, l'utilisateur final et l'analyste des besoins. Une activité, représentée par un "empennage de flèche", est réalisée par un acteur si elle apparaît dans la ligne correspondant aux rôles de participants (par exemple, la définition des besoins de l'utilisateur est faite par le client). Les flèches pointillées indique les flots des produits d'activité en activité. Certaines activités sont conditionnées, grâce à des branchements type UML.

3.3.1 Définir les besoins utilisateur (A_1)

Cette activité représente la première activité du processus des besoins préliminaires, qui s'étend de l'activité A_1 à l'activité A_5. Cette première activité concerne la description du système et de l'environnement dans lequel le système sera déployé. Elle consiste à définir ce qu'il faut construire ou ce que sera le système le plus adapté aux utilisateurs finals.

Les utilisateurs finals, les clients, les analystes et concepteurs doivent lister les besoins potentiels. Le contexte dans lequel le système sera déployé doit être compris. Les besoins fonctionnels et non fonctionnels doivent être établis. Cette information doit être dans le document intitulé *cahier des charges* (ou *requirements set* dans la figure 3.2) préliminaire.

Exemple 3.1. *Le système à réaliser est un gestionnaire d'emplois du temps (d'enseignements). À partir d'informations données par l'utilisateur, il affiche une répartition des cours sur un emploi du temps. On peut donner une description des données qui sont reçues par le système concernant les enseignants, les groupes d'étudiants et les salles dans lesquelles sont donnés les cours :*

- *Une description d'enseignant qui contient les informations suivantes :*
  - *Le nom de l'enseignant ;*
  - *Ses indisponibilités (les jours ou les plages horaires durant lesquels il ne peut pas enseigner, ...) ;*
  - *Ses capacités (les sujets qu'il enseigne, ...) ;*
  - *Les besoins qu'il a à propos d'équipements pédagogiques particuliers (rétro-projecteur, vidéo-projecteur, salle de TP, ...).*

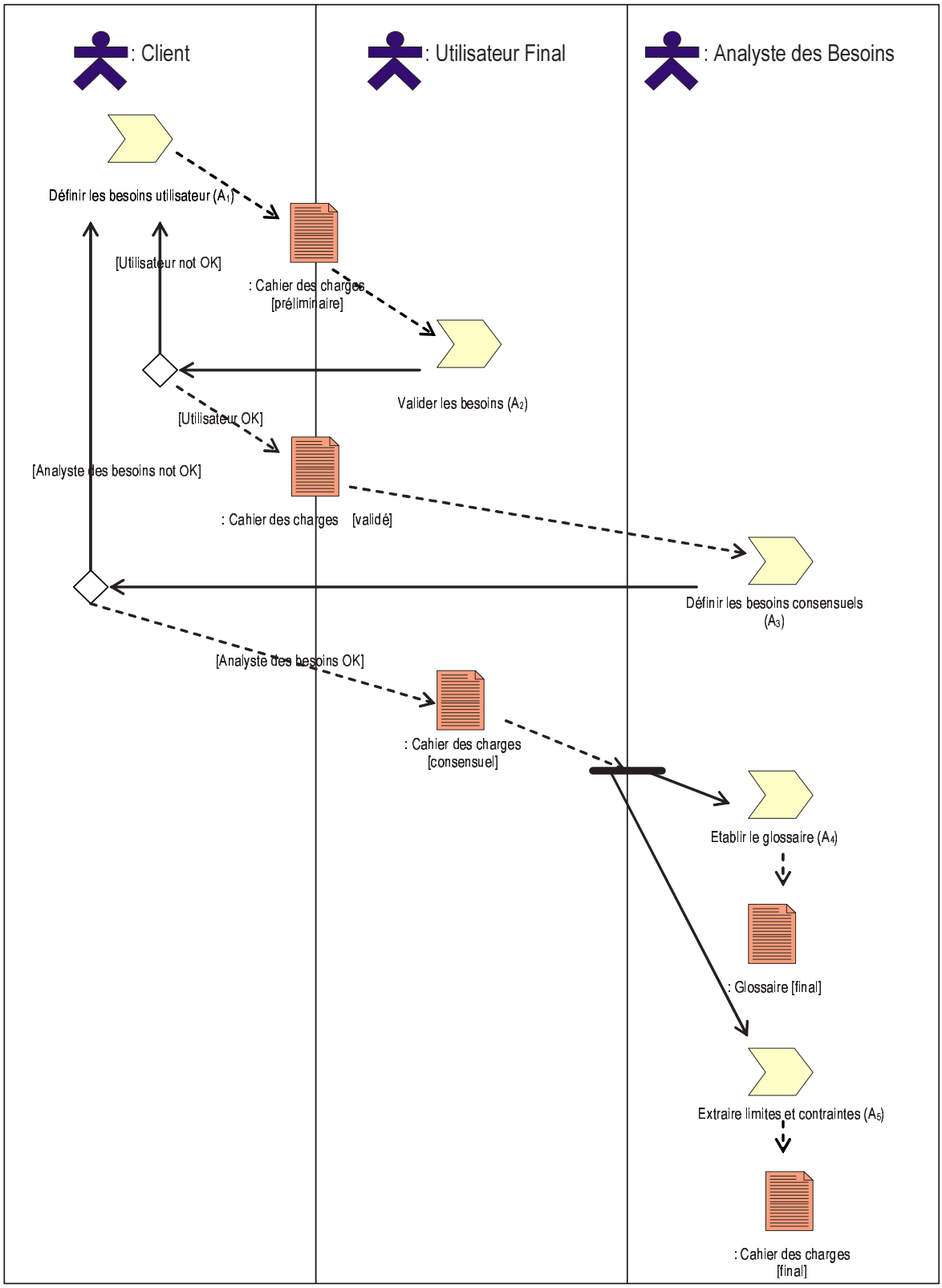

Figure 3.2 — La définition de travaux WD_1 : les besoins préliminaires.

- ▷ *Une description de groupe d'étudiants qui contient les informations suivantes :*
  - *Son identifiant ;*
  - *Les sujets qu'il doit étudier ;*
  - *Le nombre d'heures à suivre dans chaque matière.*
- ▷ *Une définition de salle de cours qui comprend :*
  - *Son numéro ;*
  - *Le matériel spécifique dont elle est équipée (rétro-projecteur, vidéo-projecteur, salle de TP, ...).*

3.3.2 Valider les besoins utilisateur (A_2)

Dans cette activité, il faut vérifier et approuver le contenu du document *cahier des charges* précédemment établi. Si ce document n'est pas approuvé, il est nécessaire de ré-appliquer l'activité précédente. C'est une activité typique dans les méthodes orientées objet qui mènent les projets de bout en bout. Cependant, ces considérations ne soulèvent aucune problématique agent.

3.3.3 Définir les besoins consensuels (A_3)

Dans cette activité, il faut mettre à jour et compléter le document intitulé *cahier des charges* en prenant en compte les besoins consensuels. De même que pour l'étape 2, une non approbation de ce document entraînera un retour à l' activité précédente. Un besoin consensuel est une condition ou une fonctionnalité à laquelle le système doit se conformer et sur laquelle les utilisateurs finals, les concepteurs et les développeurs sont d'accord.

Exemple 3.2. *L'ensemble des besoins utilisateur est totalement réécrit par l'analyste des besoins. Voici la nouvelle version. Dans ce problème, les parties impliquées sont des enseignants, des groupes d'étudiants et des salles de cours.*

Chaque acteur possède individuellement des contraintes qui doivent être respectées au mieux. Un enseignant possède des contraintes concernant ses disponibilités (e.g. les jours ou les créneaux horaires pendant lesquels il peut enseigner), ses compétences (e.g. les matières qu'il enseigne) et ses besoins concernant des équipements pédagogiques particuliers (rétroprojecteur, vidéoprojecteur, salle de TPs, livres, etc.).

Un groupe d'étudiants doit suivre un ensemble d'enseignements constitué d'un certain nombre de créneaux horaires pour un certain nombre de matières (x créneaux pour la matière m_1, y créneaux pour la matière m_2, etc.) ce qui est défini dans le PPN ou Plan Pédagogique National.

Une salle de cours est pourvue ou non d'équipement spécifique (projecteurs, accès pour les personnes handicapées, équipement de TP, etc.) et peut être occupée ou non (e.g. durant un certain créneau ou un certain jour).

Pour chaque acteur, des contraintes doivent être données sous la forme d'une liste ordonnée. L'ordre des contraintes dans la liste donne leur importance relative ; ainsi la première contrainte donnée est celle qui sera, si c'est nécessaire, le plus facilement relâchée.

Résoudre le problème consiste à satisfaire le plus de contraintes possibles pour établir un emploi du temps sur une durée donnée.

3.3.4 Établir la liste des mots-clés (A_4)

Cette étape consiste à lister les principaux concepts utilisés pour décrire l'application et son domaine (le système et son environnement). Une définition de chaque mot-clé sera donnée dans un document intitulé *ensemble de mots-clés*. Cette étape est une première abstraction du système en devenir (i.e. quels seront les concepts manipulés ?) mais il est encore trop tôt pour introduire le concept d'agent. Il est à noter que cette étape peut se dérouler en parallèle avec l'activité A_5 comme le montre la figure 3.2.

Exemple 3.3. *D'après les besoins utilisateur, les mots-clés et concepts suivants sont mis en exergue :*
- ***Planning*** *: l'action de résoudre (établir) un emploi du temps ;*
- ***Salles*** *: lieux où les cours peuvent être donnés ;*
- ***Enseignants*** *: personnes dispensant les cours ;*
- ***Étudiants*** *: personnes assistant au cours ;*
- ***Contraintes*** *: règles que le système doit respecter ;*
- ***Organisation*** *: un état de distribution des cours ;*
- ***Gestion des contraintes*** *: action de vérifier le respect des règles et de changer l'organisation afin de respecter plus de règles.*

3.3.5 Extraire les limites et les contraintes (A_5)

Dans cette activité, il faut définir les limites et les contraintes du système à construire (l'application). Elles peuvent être déduites de l'expression des besoins non fonctionnels et de la définition du contexte dans lequel le système sera déployé. Un besoin non fonctionnel est un besoin qui spécifie les propriétés du système, telles que des contraintes environnementales ou d'implantation, de performance, de dépendance à la plate-forme cible, de maintenance, d'extensibilité et de sûreté. Un besoin qui spécifie des contraintes physiques sur un besoin fonctionnel [Jacobson *et al.*, 1999]. Cette information servira à raffiner le document *ensemble de mots-clés* précédemment établi. Cette activité peut donc se dérouler aussi en parallèle avec l'activité A_4 précédente.

Exemple 3.4. *Le problème est simplifié par le fait que seuls des créneaux horaires de deux heures sont gérés. Sinon, la principale contrainte du système est qu'un enseignant ne peut pas donner un cours à plus d'un groupe d'étudiants à la fois. Cela signifie que le système ne peut associer un enseignant qu'à un groupe d'étudiants et une salle de cours pour une tranche horaire donnée.*

3.4 Les besoins finals (WD_2)

Le but des besoins finals est de transformer la vue du système obtenue grâce aux besoins préliminaires en un modèle de cas d'utilisation, et d'organiser et gérer les besoins (fonctionnels ou non) et leurs priorités. En fait, à ce stade, les concepteurs doivent définir la fonction du système et modéliser son environnement. Le modèle est le résultat de l'étude des différents acteurs mis en jeu et de leurs possibles interactions avec le système. Les flots de données entre le système et les acteurs passifs (comme des bases de données extérieures)

sont représentés par des diagrammes de collaboration d'UML. Les interactions entre le système et les acteurs actifs (comme des acteurs humains) sont résumées en diagrammes de cas d'utilisation et détaillées dans des diagrammes de séquences relatifs à chaque cas d'utilisation. C'est à ce niveau que commence l'introduction d'activités spécifiques aux systèmes multi-agents. Deux activités, A_6 et A_7, ont été modifiées dans ce sens.

Ces travaux doivent exprimer, à partir du cahier des charges consensuel entre client et fournisseur, un modèle d'environnement composé d'un modèle d'interactions entre système et environnement et d'une caractérisation de l'environnement. De plus, cette activité permet de proposer des prototypes d'interfaces graphiques pour ces interactions système-environnement.

Les activités des besoins finals et leur ordonnancement sont résumés dans le diagramme d'activité SPEM de la figure 3.3.

3.4.1 Caractériser l'environnement (A_6)

La caractérisation de l'environnement est la première activité des besoins finals. Le principal objectif de cette activité est de définir l'environnement du système dans un document intitulé *définition de l'environnement* (ou *environment definition*).

Cette définition doit rendre plus facile la définition des cas d'utilisation et des contraintes liées. Ces contraintes et règles peuvent faire apparaître des problèmes environnementaux, comme de l'indéterminisme ou de la discontinuité. Il est important de commencer à tracer ce genre de caractéristique sle plus tôt possible dans le processus, afin de les pointer plus précisément, une fois dans l'analyse.

3.4.1.1 Déterminer les entités (S_1)

Dans cette étape, l'analyste doit identifier les entités actives et passives qui sont en interaction avec le système, ainsi que les contraintes sur ces interactions. Cette étape est classique en conception orientée objet.

Exemple 3.5. *Dans le problème ETTO, on peut déterminer 4 entités actives et 2 passives :*

- ▷ *Les entités actives sont :*
  - *Les **enseignants** car ils modifient eux-mêmes leurs contraintes (dynamicité, autonomie) et ils interagiront avec le système ;*
  - *Les **étudiants** car il interagiront avec le système ;*
  - *La personne en charge de l'enseignement (appelée **gestionnaire des enseignements**) car elle devra choisir les cours à enseigner durant la période durant laquelle l'emploi du temps doit être défini. Il peut aussi apporter des modifications durant le fonctionnement du système ; il est capable d'agir sur le système ;*
  - *la personne qui gère les salles (appelée **gestionnaire des salles**) pour les mêmes raisons ; son rôle est de modifier les contraintes des salles.*
- ▷ *Les entités passives sont :*
  - *Les **salles** car ce sont de simples ressources ; la personne qui gère les salles décide de l'évolution de leurs caractéristiques ;*

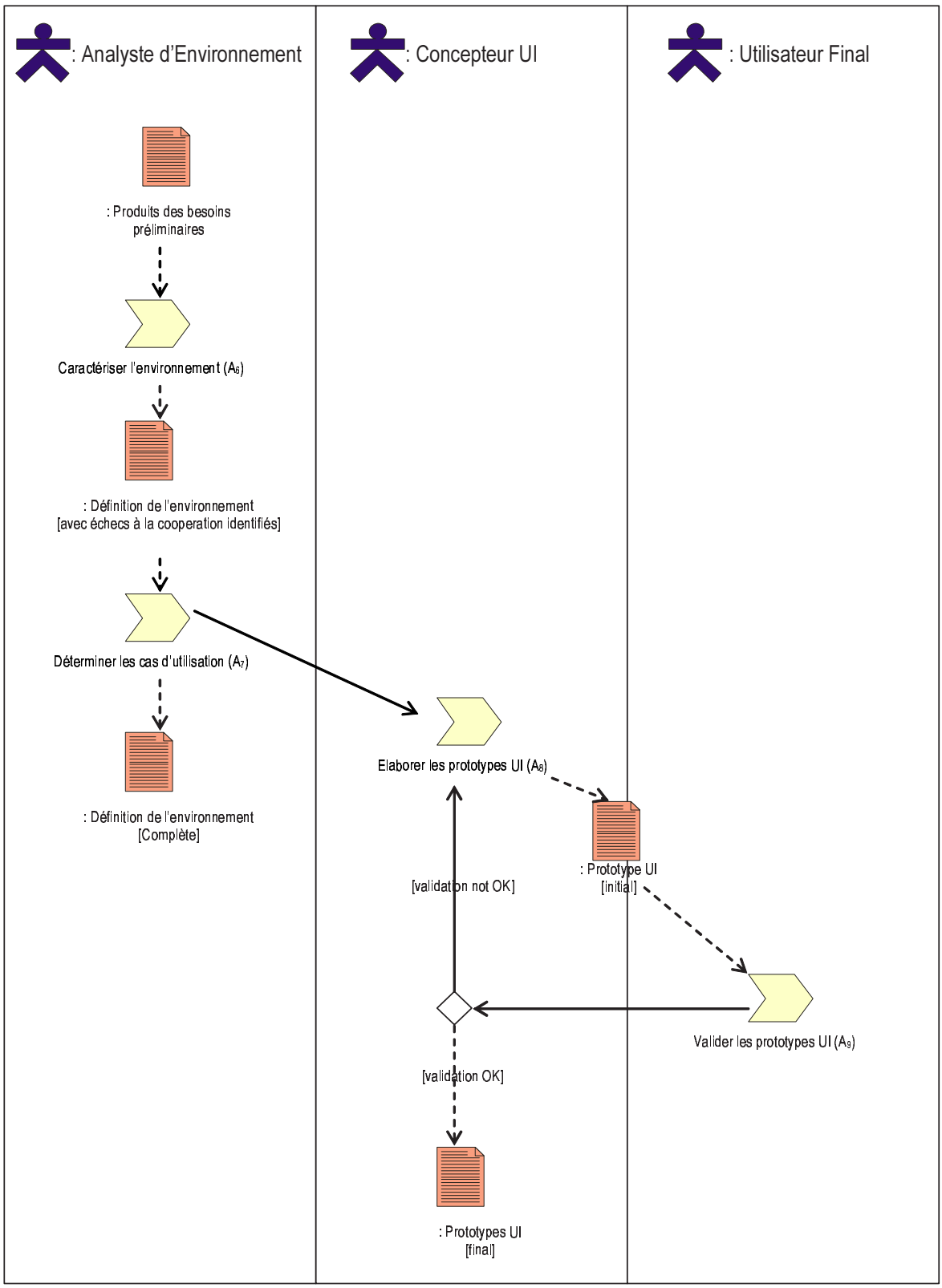

Figure 3.3 — La définition de travaux WD_2 : les besoins finals.

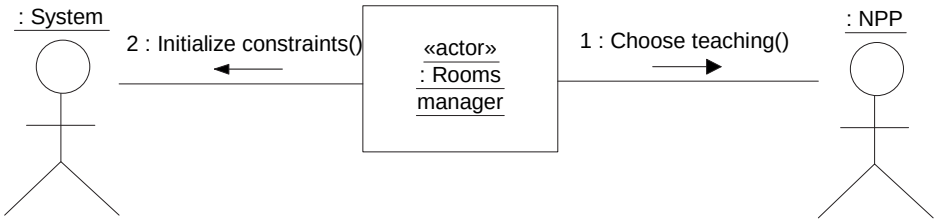

Figure 3.4 — Diagramme de collaboration entre le système et le plan pédagogique national (ou NPP).

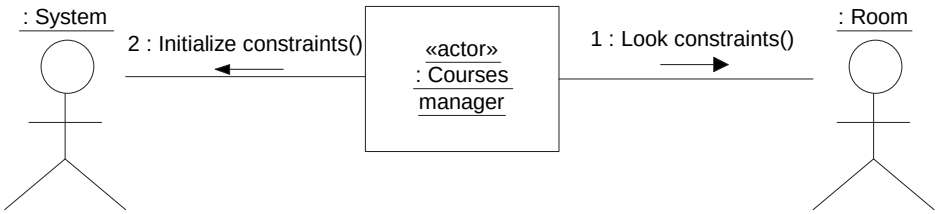

Figure 3.5 — Diagramme de collaboration entre le système et les salles.

– *Le **Plan Pédagogique National** (ou PPN ou NPP). Il inventorie tous les cours à donner durant l'année pour chaque formation. La personne en charge de l'enseignement utilise le PPN.*

3.4.1.2 Définir le contexte (S_2)

Dans cette étape, il faut caractériser les flots de données et les interactions entre les entités identifiées et le système. Les flots de données entre les entités passives et le système sont exprimées grâce à des diagrammes de collaboration. Les interactions entre les entités actives et le système s'expriment via des diagrammes de séquences.

Exemple 3.6. *On peut distinguer deux types de flots de données entre le système et les entités passives.*

Le premier type doit permettre au système d'obtenir les contraintes d'enseignement des enseignants et des groupes d'étudiants depuis le plan pédagogique national (NPP). Ces contraintes sont transmises par le gestionnaire des enseignements. Ceci mène au diagramme de collaboration entre le système et le NPP de la figure 3.4.

Le second type doit permettre au système d'être informé de l'état des contraintes des salles. Cette information est transmise par le gestionnaire des salles. Ceci mène au diagramme de collaboration entre le système et les salles de la figure 3.5.

De plus, il y a des interactions possibles avec chaque entité active :
- *Le gestionnaire des enseignements est capable de consulter le plan pédagogique national (NPP), d'initialiser et de mettre à jour les contraintes d'enseignement des enseignants et des groupes d'étudiants. Il peut aussi lancer la résolution et visualiser le résultat fourni par le système ;*
- *Le gestionnaire des salles est capable de consulter les salles et d'initialiser et mettre à jour les contraintes de ces salles ;*
- *Les enseignants peuvent initialiser et mettre à jour les contraintes concernant leur disponibilité*

et visualiser le résultat fourni par le système ;
– *Les groupes d'étudiants peuvent visualiser le résultat fourni par le système.*

3.4.1.3 Caractériser l'environnement (S_3)

Dans cette étape, il faut caractériser environnement en utilisant les termes proposés par Russel et Norvig, qui représente une bonne caractérisation des problèmes que peuvent rencontrer les système artificiels et qui justifie l'utilisation de système intelligents afin de palier ces difficultés [Russel et Norvig, 1995] (voir §1.2.3) :

Accessible ou non : le système peut, ou non, obtenir une information complète, exacte et à jour sur l'état de son environnement. Il est important d'identifier les entités qui devront faire directement face à cette inaccessibilité, afin de mettre en place des moyens de demande de précision à l'utilisateur par exemple ;

Déterministe ou non : dans un environnement non déterministe, une action n'a pas un effet unique garanti. Il faut identifier les entités de l'environnement fournissant les retours des actions effectuées afin de prévoir des réponses adéquates ;

Statique ou **dynamique** : l'état d'un environnement dynamique dépend des actions du système qui se trouve dans cet environnement mais aussi des actions d'autres processus. Aussi, les changements ne peuvent pas être prédits par le système. Par contre, il convient de clairement identifier les acteurs de la dynamique afin d'en déterminer la nature ;

Discret ou **continu** : Dans un environnement continu, le nombre d'actions et de perceptions possibles dans cet environnement est infini. Par exemple, un système devant prendre en entrée des données textuelles possédant une sémantique forte, comme dans ABROSE, est face à un environnement continu [Gleizes et Glize, 2000]. Ici, l'identification de telles caractéristiques permet, par exemple, la mise en place de structures dédiées à la prise en compte de ces chaînes de caractères par décomposition récursive : une sorte d'ontologie dynamique.

Cette caractérisation correspond à une partie de l'artefact *définition de l'environnement* (ou *environment definition*.

Exemple 3.7. *L'environnement d'ETTO peut se définir en tant que :*

***Dynamique** : Les enseignants, les groupes d'étudiants, le gestionnaire de salles et le gestionnaire des enseignements sont imprévisibles. Ils peuvent ajouter ou modifier des contraintes à n'importe quel moment ;*

***Accessible** : Toutes les salles sont décrites par le gestionnaire de salles et toutes les contraintes sont données par les autres entités actives ;*

***Non déterministe** : Le système n'a pas de représentation de corrélation entre les résultats qu'il fournit et les contraintes ajoutées ou modifiées par les entités actives ;*

***Continu** : Les entités actives sont toujours capables d'ajouter ou de modifier leurs contraintes.*

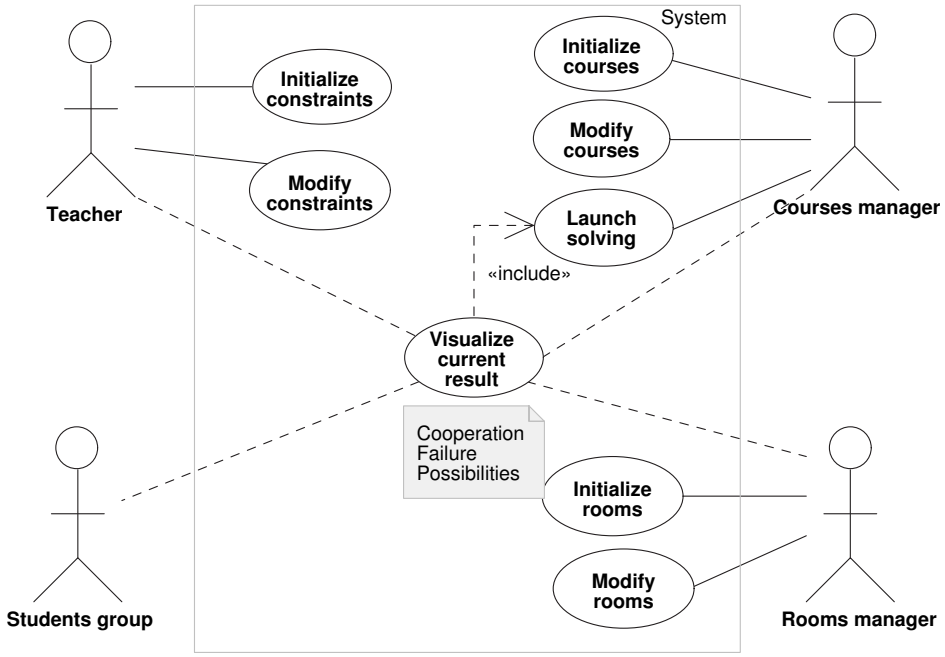

Figure 3.6 — Les cas d'utilisation d'ETTO.

3.4.2 Déterminer les cas d'utilisation (A_7)

L'objectif principal de cette activité est de clarifier les différentes fonctionnalités que le système étudié doit fournir. Par la suite, ces fonctionnalités seront assemblées en un ou plusieurs cas d'utilisation entre les entités actives précédemment identifiées et le système. Un cas d'utilisation est détaillé par une description textuelle et des diagrammes de séquences spécifiques. Si besoin est, un cas d'utilisation peut posséder une boîte "exceptions" dans laquelle les conditions sous lesquelles le service ne peut être rendu doivent être mentionnées.

3.4.2.1 Inventorier les cas d'utilisation (S_1)

Tout d'abord, durant cette étape, il faut identifier tous les différents cas d'utilisation qui peuvent exister pour le système étudié. Il faut donc construire les diagrammes de cas d'utilisation correspondants. Par la suite, il sera possible de mettre à jour les diagrammes ainsi créés et d'ajouter une description textuelle pour chacun d'entre eux. La description textuelle d'un cas d'utilisation peut suivre le schéma ci-dessous, inspiré du plan proposé par Müller et Gaertner [Müller et Gaertner, 2000] : '

- Décrire le rôle du cas d'utilisation ;
- Indiquer quel est le début de ce cas d'utilisation, quand il se produit ;
- Indiquer sa fin, sous quelles conditions il se termine ;
- Donner sa (ses) pré-conditions : à quelle(s) condition(s) se produit-il ?
- Donner sa (ses) post-condition(s) : que se passe-t-il lorsqu'il se termine ? ;
- Préciser les exceptions : quand ne peut-il se produire normalement ?

Exemple 3.8. *L'observation des interactions précédentes amène à la définition des cas d'utilisation suivants (voir figure 3.6) :*

L'entité "gestionnaire des enseignements" (Course manager) peut agir pour :

- *initialiser les enseignements : afin de décrire les contraintes d'enseignement des enseignants et des groupes d'étudiants ;*
- *les modifier : afin de changer les contraintes des enseignants et des groupes d'étudiants concernant leurs enseignements ;*
- *lancer la résolution : afin de faire démarrer la recherche d'une solution au problème prenant en compte les contraintes déjà décrites ;*
- *visualiser le résultat courant : afin d'afficher le résultat fourni par le système au problème qui lui a été posé.*

L'entité "gestionnaire des salles" (Room manager) peut agir pour :

- *initialiser les salles : afin de définir une salle et décrire ses contraintes ;*
- *les modifier : afin de changer les contraintes de disponibilité ;*
- *visualiser le résultat courant : afin d'afficher le résultat fourni par le système au problème qui lui a été posé.*

Les entités "enseignants" (Teacher) peuvent :

- *initialiser les contraintes : afin de définir leurs contraintes de disponibilité ;*
- *les modifier : afin de changer les contraintes de disponibilité ;*
- *visualiser le résultat courant : afin d'afficher le résultat fourni par le système au problème qui lui a été posé.*

et, finalement, les entités "groupes d'étudiants" (Students group) peuvent :

- *visualiser le résultat courant : afin d'afficher le résultat fourni par le système au problème qui lui a été posé.*

3.4.2.2 Identifier les échecs à la coopération (S_2)

Durant cette étape, il faut réfléchir sur les événements qui peuvent mener à des situations qui ne sont pas totalement contrôlées par le concepteur du système et peuvent être néfastes. Le but de cette étape est de pointer du doigt les "mauvaises" interactions pouvant se produire entre les entités et le système. La théorie des AMAS nomme ces événements et situations, des échecs à la coopération (*cooperation failures*). Ces événements peuvent être vus comme des sortes d'"exceptions" au niveau du système. Ces échecs à la coopération doivent être mis en exergue au sein des cas d'utilisation précédemment identifiés et ADELFE fournit une notation spécifique pour cela : des liens pointillés entre l'acteur de l'environnement, source du problème, et le cas d'utilisation devant prendre en charge les problème de coopération.

Exemple 3.9. *Les associations vers le cas d'utilisation de visualisation des résultats (Visualize current result dans la figure 3.6) sont potentiellement non coopératives : le résultat de la résolution est la seule cause d'échec à la coopération entre les utilisateurs et le système, dans le sens ou le système doit satisfaire les contraintes de chacun des participants. Ceci est un indice qui signifie que ce sera certainement cette fonctionnalité qui pourra être développée comme un AMAS.*

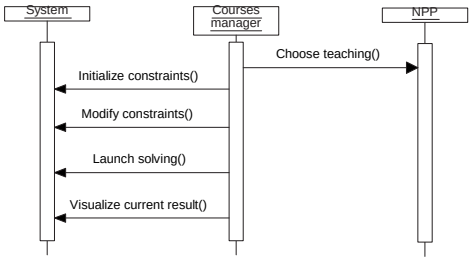

Figure 3.7 — Cas d'utilisation entre le gestionnaire de cours et le système.

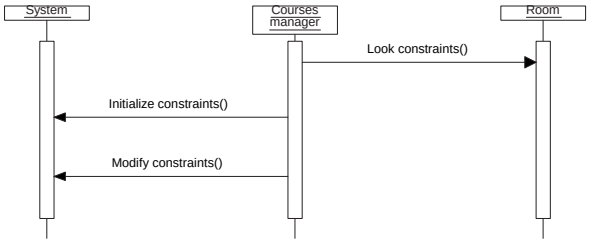

Figure 3.8 — Cas d'utilisation entre le gestionnaire de salles et le système.

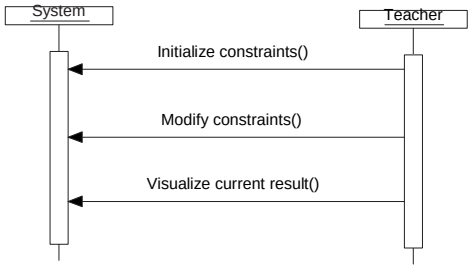

Figure 3.9 — Cas d'utilisation entre un enseignant et le système.

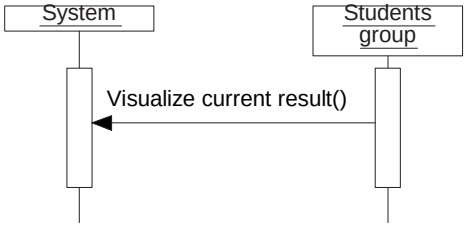

Figure 3.10 — Cas d'utilisation entre un groupe d'étudiants et le système.

3.4.2.3 Élaborer les diagrammes de séquences (S_3)

Pour chacun des cas d'utilisation précédemment établis, un diagramme de séquences correspondant doit être construit. Ceci permet de préciser comment les fonctionnalités du système se mettent en œuvre du point de vue collaboratif et temporel et comment les acteurs y prennent part. Il est à noter que ces interactions ne s'effectuent pour l'instant qu'entre les acteurs et le système en général. Aucune décomposition n'est encore faite.

Concernant notre problématique, c'est aussi et surtout un moyen de tracer, grâce au raffinage futur des diagrammes de séquences, quelles vont être les entités qui prendront en charge les échecs à la coopération.

Exemple 3.10. *Pour tout enseignement choisi dans le NPP par le gestionnaire des enseignements (Course manager), ce dernier doit initialiser les contraintes correspondantes des enseignants et des groupes d'étudiants. Il peut mettre à jour ces contraintes. Quand toutes les contraintes sont définies, il a la possibilité de lancer la résolution et d'afficher alors les résultats à tout moment (figure 3.7).*

Pour chaque salle, le gestionnaire des salles (Room manager) recherche des contraintes sur cette salle et définit alors ces contraintes dans le système. Il peut mettre à jour ces contraintes si elles évoluent (figure 3.8).

Un enseignant peut définir ses contraintes d'indisponibilité et les mettre à jour. Après que le gestionnaire des enseignements a lancé la résolution, un enseignant est aussi capable d'afficher le résultat fourni par le système (figure 3.9).

Un groupe d'étudiants (Student group) est seulement capable d'afficher le résultat fourni par le système, après le lancement de la résolution par le gestionnaire des enseignements (figure 3.10).

3.4.3 Élaborer les prototypes d'interfaces utilisateur (A_8)

Dans cette activité, il faut spécifier – dans le document *prototypes d'interfaces graphiques* (ou *UI prototype*) – les interfaces graphiques (GUI) grâce auxquelles l' utilisateur interagira avec le système ainsi que les relations entre ces GUI. Ceci ne constitue pas le cœur de nos préoccupations, nous ne présenterons aucune interface graphique à ce niveau. Cependant, le chapitre 5 présente le prototype développé pour ETTO.

Exemple 3.11. *Dans le système qui doit être réalisé, l'interface utilisateur sera divisée en trois parties :*

- *L'utilisateur doit pouvoir lancer la recherche de la solution. Afin d'étudier l'évolution de la solution, il a besoin d'une fonctionnalité "pas-à-pas". Cette interface sera composée d'une unique fenêtre comprenant des boutons "démarrer" (start) et "pas-à-pas" (step by step).*
- *L'utilisateur doit définir les contraintes pour les entités. Pour cela, on utilisera un fichier d'initialisation.*
- *L'utilisateur visualisera les résultats via un fichier html généré contenant une représentation de la grille.*

3.4.4 Valider les prototypes d'interfaces utilisateur (A_9)

Les interfaces graphiques utilisateur (GUI), décrites dans le document *prototypes d'interfaces graphiques*, doivent être étudiées et jugées des points de vue fonctionnel et non fonctionnel (ergonomique, lié à la conception, ...). Si la validation des interfaces graphiques échoue, il convient de revenir à l'activité précédente afin de retravailler sur ces interfaces.

Exemple 3.12. *Les interfaces présentées dans le document prototypes d'interfaces graphiques fournissent toutes les fonctionnalités désirées mais l'ergonomie peut être améliorée. Dans une future version, il sera nécessaire de développer une réelle GUI pour définir les contraintes.*

3.5 L'analyse (WD_3)

D'un point de vue multi-agent, l'identification et la définition des agents prennent place à cette étape du processus. L'analyse doit dégager une compréhension du système, de sa structure en termes de composants et repérer si la théorie des AMAS est nécessaire. La spécificité d'ADELFE implique que tout concepteur n'a pas forcément besoin d'ADELFE car toutes les applications ne requièrent pas l'emploi de systèmes multi-agents adaptatifs. L'analyse débute par une étude, ou analyse, du domaine. Un diagramme préliminaire de classes du système et de son environnement est établi afin de construire une première vue statique du système. La dynamique du système est représentée au travers d'un ensemble de diagrammes de séquences entre entités actives. Nous augmentons l'analyse classique d'une nouvelle étape, l'adéquation des AMAS, afin de vérifier si les modèles d'environnement et de système précédemment établis correspondent à un modèle AMAS. Une autre étape supplémentaire est l'identification des agents. Toutes les entités actives ne sont pas forcément des agents. De fait, l'analyste doit identifier les agents en accord avec différents critères (comme l'autonomie, par exemple) et avec les interactions avec l'environnement.

Après les étapes d'expression des besoins, l'objectif de l'analyse est de construire une compréhension du système afin de le structurer en termes de composants et de repérer la nécessité de l'utilisation de la théorie des AMAS. Les activités de l'analyse et leur ordonnancement sont résumés dans le diagramme d'activité SPEM de la figure 3.11.

3.5.1 Analyser le domaine (A_{10})

L'analyse du domaine est une vue statique et une abstraction du monde réel établie depuis les documents *cahier des charges* et *ensemble des mots-clés*. En considérant de manière séparée chaque cas d'utilisation et en définissant des scenarii, le concepteur devra diviser le système en entités. Ces entités peuvent être concrètes (par exemple, un enseignant) ou abstraites (par exemple, une plage horaire). Le résultat de cette étape est un ensemble d'entités dans un diagramme de classes préliminaire décrit dans un document nommé *architecture logicielle*.

3.5.1.1 Identifier les classes (S_1)

Dans cette étape, l'ensemble des cas d'utilisation définis précédemment, les diagrammes de séquences correspondants et le document *ensemble des mots-clés* doivent être analysés pour ainsi identifier les classes nécessaires. Lors de l'identification de ces classes, on peut mettre à jour la liste des entités déjà donnée (voir A_6S_1).

Exemple 3.13. *Dans ETTO, on peut tout d'abord identifier les classes suivantes d'après les entités décrites durant A_6S_1 :*

- *Enseignants, étudiants et salles ;*
- *Gestionnaire des salles ;*
- *Gestionnaire des enseignements.*

Le NPP est une exception car il n'est ni associé à une autre entité, ni en interaction directe avec le système. On ne définit alors pas de classe pour le NPP.

On peut alors identifier les classes qui sont utiles au système :

- *Constraint (Contrainte) : semble être essentielle ;*
- *Constraints manager (Gestionnaire de contraintes) : ne personne en charge des contraintes de chaque entité en possédant ;*
- *Grid (Grille) : une grille pour placer les salles et représenter les résultats. On choisit une grille à trois dimensions : la salle, l'horaire et le jour. Avec une telle représentation, une salle peut apparaître plusieurs fois.*
- *Cell (Cellule) : L'intersection de chaque dimension de la grille.*

3.5.1.2 Étudier les relations entre classes (S_2)

Il faut maintenant étudier les relations entre les différentes classes précédemment identifiées. Cela se fera en étudiant les cas d'utilisation et les diagrammes de séquences existants. Ces relations pourront se modéliser en termes de lien associatif, d'association, d'héritage ou d'agrégation/composition.

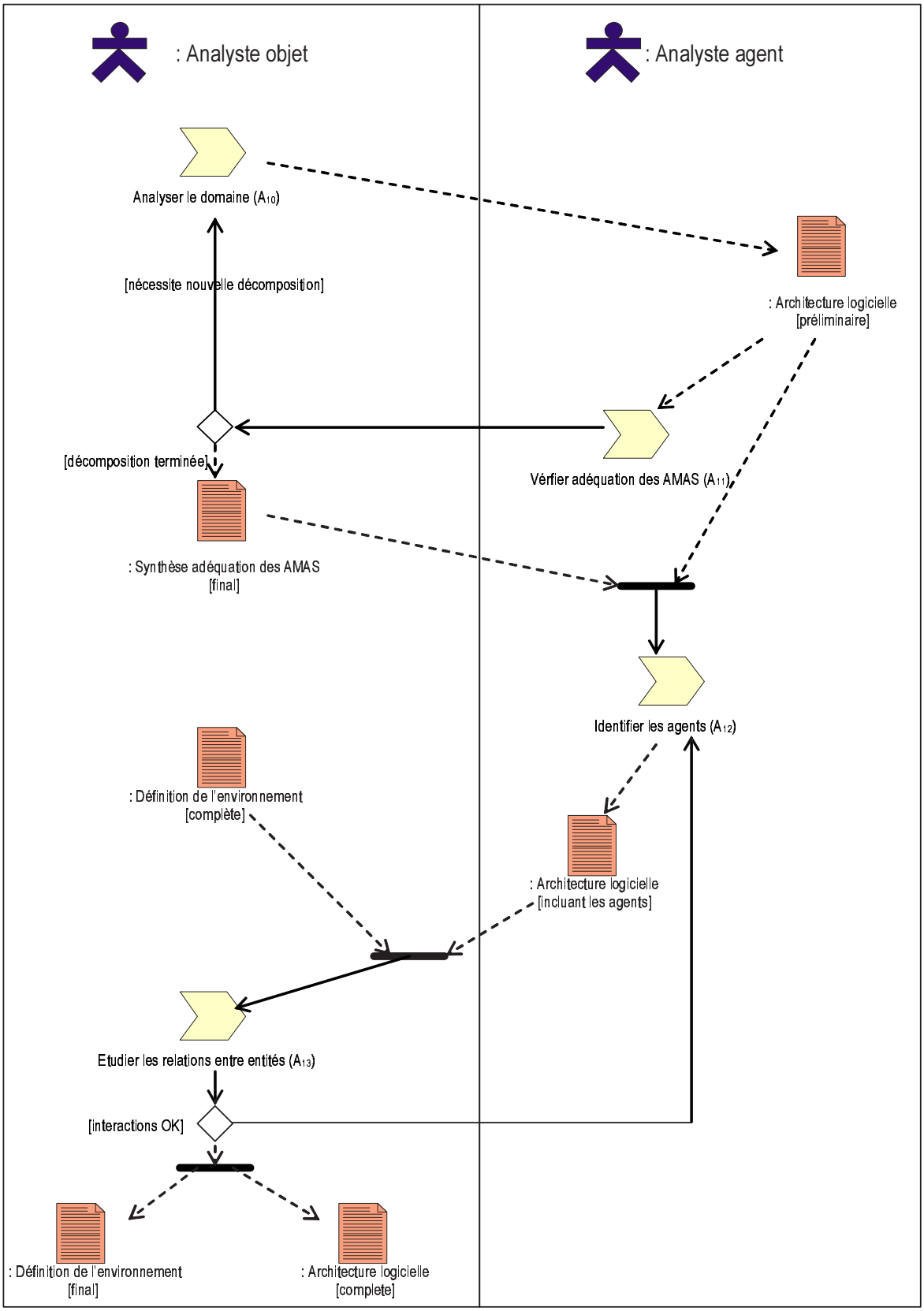

Figure 3.11 — La définition de travaux WD_3 : l'analyse.

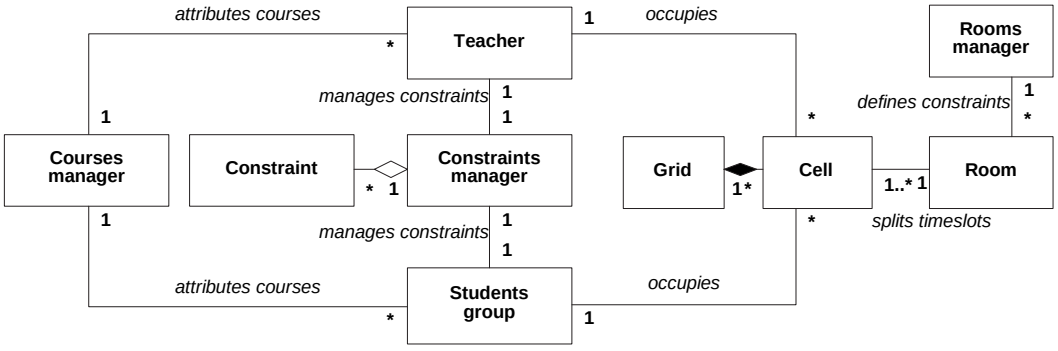

Figure 3.12 — Le diagramme de classes préliminaire d'ETTO.

Exemple 3.14. *Le gestionnaire des enseignements (CourseManager) attribue les enseignements aux groupes d'étudiants (StudentsGroup) et aux enseignants (Teacher). Les enseignants et les groupes d'étudiants reçoivent des cours de la part du gestionnaire des enseignements et les transmettent au gestionnaire des contraintes (ConstraintManager). La grille (Grid) est composée de cellules (Cell) correspondant aux salles (Room) pour des périodes de temps fixes. Les enseignants et les groupes d'étudiants seront placés sur la grille afin de trouver des partenaires et une salle correspondant à leurs contraintes. Un gestionnaire des contraintes est une agrégation des contraintes qu'il gère. Le gestionnaire des salles (RoomManager) définit et modifie les contraintes des salles.*

3.5.1.3 Construire les diagrammes de classes préliminaires (S_3)

Une fois les différentes classes et leurs interactions identifiées, le diagramme de classes préliminaire peut être construit. La version préliminaire des classes doit être stockée dans un document nommé *architecture logicielle*.

Exemple 3.15. *La figure 3.12 montre le diagramme de classes préliminaire d'ETTO, qui correspond à l'analyse faite dans l'exemple 3.14.*

3.5.2 Vérifier l'adéquation aux AMAS (A_{11})

Dans cette activité, il faut vérifier si un (ou plusieurs) système(s) multi-agent(s) adaptatif(s) (AMAS) est(sont) nécessaire(s) pour réaliser l'application à construire. Parfois, ce type de modélisation est complètement inutile. Par exemple, avoir un système qui est capable de s'adapter est complètement inutile si l'algorithme requis pour résoudre la tâche est déjà connu, si la tâche n'est pas complexe ou tout simplement si le système à construire ne possède aucune des propriétés identifiées dans le chapitre 1.

Dans cette activité, il faut donc étudier l'adéquation à deux niveaux :

- au **niveau global**, pour répondre à la question "un AMAS est-il nécessaire pour implanter le système" ?
- au **niveau local**, pour essayer de déterminer si des agents ont besoin eux aussi d'être implantés en tant qu'AMAS, c.-à-d. si un certain degré de décomposition ou de récursivité est requis durant la construction du système.

A la fin de cette activité, l'analyste doit être capable de répondre à ces deux questions.

Pour ceci, il devra donc prendre une décision multi-critère qu'il aura lui-même établie ou bien utiliser l'outil que nous avons développé (voir §4.3). Ce dernier produit le résultat pour les deux niveaux et s'appuie sur l'expérience acquise par l'équipe à travers la mise en œuvre de différents projets utilisant la modélisation AMAS (voir §1.4).

Toutes les informations concernant cette étape sont stockées dans un document nommé *synthèse de l'adéquation* aux AMAS.

Il est possible de reboucler sur l'activité A_{10} précédente si une décomposition à été détectée afin de préciser l'analyse du domaine et obtenir une décomposition suffisante.

3.5.2.1 Vérifier l'adéquation aux AMAS au niveau global (S_1)

Pour décider si une implantation en utilisant un SMA adaptatif (AMAS) est utile au niveau global, on peut étudier les huit questions suivantes :

1. *La tâche globale est-elle incomplètement spécifiée ? Un algorithme est-il inconnu a priori ?* En effet, les AMAS sont adéquats pour des systèmes non complètement spécifiés, pour lesquels la solution pourra être émergente.
2. *Si plusieurs entités sont nécessaires pour résoudre la tâche globale, doivent-elles agir dans un certain ordre ? Une activité corrélée est-elle nécessaire ?* La coopération peut être une bonne solution pour gérer la collaboration entre composantes et leur ordonnacement.
3. *La solution est-elle, de manière générale obtenue par essais successifs, plusieurs essais sont-ils requis avant de trouver une solution ?* Cette question concerne certains problèmes dans lesquels la sortie du système ne peut être prévue en fonction des conditions initiales et pour lesquels une technique par essai/erreur est utilisée afin de modifier les paramètres pas-à-pas. Les AMAS peuvent répondre à cette problématique par leur caractère non-linéaire.
4. *L'environnement du système peut-il évoluer ?* Auquel cas, un système adaptatif par auto-organisation, par exemple, sera une bonne solution.
5. *Le traitement effectué par le système est-il fonctionnellement ou physiquement distribué ? Plusieurs entités physiquement réparties sont-elles utiles à la résolution de la tâche globale ? Une distribution conceptuelle est-elle nécessaire ?* Cette question envisage la pertinence de l'utilisation de systèmes multi-agents comme solution aux problème distribués.
6. *Est-il nécessaire d'avoir un grand nombre d'entités ?* Plus un système possède d'entités, plus les règles de coopération sont nécessaires, et donc les AMAS seront peut-être une solution.
7. *Le système étudié est-il linéaire ?* Si oui, il y a peu de chance que les AMAS soient utiles à la conception, car on peut résoudre le problème en fournissant une organisation hiérarchique fixe des composantes.
8. *Enfin, le système est-il évolutif ou ouvert ? De nouvelles entités peuvent-elles apparaître ou disparaître dynamiquement ?* Les AMAS peuvent répondre à ces problèmes d'ouverture par leur capacité d'auto-organisation.

Toutes les informations à propos de cette étape sont stockées dans le document intitulé *synthèse de l'adéquation aux AMAS*.

Exemple 3.16. *Pour ETTO, on peut répondre à ces questions de la manière suivante (en estimant l'importance du critère sur une échelle allant de 0 à 20) :*

1. ***La tâche globale ne pourra pas être complètement spécifiée** (18/20) : Établir un emploi du temps consiste à pouvoir satisfaire au mieux les contraintes de chacun des intervenants. Le résultat que nous souhaitons obtenir est donc connu mais les moyens et les solutions à mettre en oeuvre pour y arriver ne le sont pas forcément ; a priori, il n'y a pas de solution algorithmique universelle pour résoudre l'ensemble de ces contraintes.*
2. ***La solution impose l'activité corrélée de plusieurs composants** (02/20) : Etant donné le problème posé, il est encore trop tôt pour que nous sachions la façon dont vont interagir les entités du système. Il est fort possible qu'elles interagissent mais nous ne pouvons dire si elles vont avoir une "activité corrélée".*
3. ***La solution est habituellement obtenue par essais successifs** (20/20) : La solution "idéale" à l'emploi du temps n'existe généralement pas, certaines contraintes ne pourront être prises en compte et certains intervenants (au sens large) devront relâcher une ou plusieurs de leurs contraintes. La solution est, en général, atteinte par tâtonnements.*
4. ***L'environnement du système est évolutif, dynamique** (20/20) : Pouvoir ajouter ou modifier des contraintes en temps réel oblige l'environnement du système à être clairement dynamique.*
5. ***Le système est physiquement ou fonctionnellement distribué** (0/20) : Même si les intervenants dans le problème de l'emploi du temps sont physiquement distribués, la résolution du problème n'implique en rien une distribution des données ou des fonctions.*
6. ***Le système comporte un nombre considérable de composants** (10/20) : Le nombre de composants à mettre en oeuvre pour résoudre ce problème ne dépend que de la vision qu'en a l'ingénieur construisant le système ou du problème posé (en termes de professeurs, d'élèves et de salles). Nous ne pouvons, a priori, décider de ce nombre.*
7. ***Le système est potentiellement non linéaire** (15/20) : On peut dire que le fait de tenir compte des contraintes d'un certain intervenant peut avoir un impact sur le fait de conserver ou de relâcher les contraintes d'un autre intervenant. Les intervenants sont donc liés et ce que l'on fait pour l'un peut influencer la manière dont un intervenant va être traité.*
8. ***Le système est ouvert, évolutif** (15/20) : Ajouter des étudiants, des salles, des contraintes en cours de fonctionnement demande que le système soit ouvert. Cependant, ce n'est pas une caractéristique première du système. La réponse est donc positive mais avec un peu de retenue.*

La réponse pour le niveau global est alors qu'en utilisant la fonction de décision que nous avons définie dans l'outil d'adéquation, on peut conclure que certains aspects de l'application justifient un développement à l'aide des AMAS. Il faudra identifier ultérieurement les classes à décomposer en AMAS.

3.5.2.2 Vérifier l'adéquation aux AMAS au niveau local (S_2)

Afin de déterminer l'adéquation aux AMAS au niveau local, on peut étudier trois critères supplémentaires. Une question est associée à chacun d'entre eux et il faut y répondre selon le même principe que dans l'étape précédente.

9. *Une entité n'a-t-elle qu'une rationalité limitée ?* Si les composantes n'ont pas accès à toutes les données de traitement, il sera nécessaire de mettre en place des mécanismes coopératifs d'échange de connaissances afin d'atteindre une solution efficacement.
10. *Une entité est-elle de forte granularité ou non ? Est-elle capable d'effectuer de nombreuses actions, de raisonner beaucoup ? Doit-elle posséder beaucoup de capacités pour effectuer sa propre tâche ?* Plus une entité peut faire d'actions, plus il est difficile de les agencer ou de fournir des modèles de décision. Les AMAS peuvent être une solution par décomposition.
11. *Le comportement d'une entité peut-il évoluer ? L'entité doit-elle s'adapter aux changements de son environnement ?* Comme les AMAS fournissent des fonctions adaptatives, c'est un bon moyen de faire des systèmes apprenants par composition.

Ici encore, une décision multi-critère s'impose. Toutes les informations relatives à cette étape seront aussi stockées dans le document nommé *synthèse de l'adéquation aux AMAS*.

Exemple 3.17. *L'adéquation aux AMAS étudiée au niveau "local" pour l'application ETTO donne le résultat suivant :*

9. ***Les composants du système n'ont qu'une rationalité limitée*** *(10/20) : Pour l'instant, les composants n'ont pas été définis et dépendent (comme à la question 6) de la vision qu'en a l'ingénieur. Il est donc trop tôt pour déterminer ce que ces composants connaissent de l'environnement.*
10. ***En première analyse, les composants ont une forte granularité*** *(17/20) : Les composants chargés d'établir la solution au problème de l'emploi du temps doivent être, a priori, capables d'apporter l'expertise d'un humain résolvant le problème "à la main", leur granularité est donc forte.*
11. ***Le comportement des composants est susceptible d'évolution*** *(0/20) : Pour le problème de l'emploi du temps, tel qu'il a été défini, les composants du système à mettre en oeuvre n'ont pas besoin de s'adapter à un environnement foncièrement différent ou à des intervenants d'un type nouveau, leur comportement devrait rester sensiblement stable.*

La réponse pour le niveau local est alors que certains composants du système justifient le développement avec les AMAS. ADELFE doit donc être appliquée à ces composants.

3.5.3 Identifier les agents (A_{12})

Le but de cette activité est de trouver ce qui sera considéré comme des agents dans le système à construire. Seuls les agents qui permettent de construire des AMAS sont intéressants. En effet, seuls des systèmes à milieu intérieur coopératif fournissent une fonction adéquate. Ces agents seront recherchés parmi les entités définies lors de A_6S_1. Il est ensuite nécessaire, afin de garantir des règles de bonne utilisation, de stéréotyper la classe de chaque entité sélectionnée grâce au stéréotype nommé «cooperative agent» (voir §4.1). Cette activité enrichit le document *architecture logicielle*.

Comme nous avons pu le voir dans le chapitre 2, peu de méthodes proposent de telles activités et considèrent les agents de manière "évidente". Par exemple, Tropos propose une identification des agents par agrégation de buts du système (voir §2.4.2.11).

Cette activité peut se succéder plus ou moins régulièrement avec l'activité A_{13} d'étude des interactions afin de faciliter l'identification, comme il est spécifié par la condition interactions OK de la figure 3.11. En effet, l'identification des agents repose en partie sur l'analyse du parcours des interactions "à risque" provenant de l'environnement.

3.5.3.1 Étudier les entités relativement au domaine (S_1)

Pour chaque entité définie dans A_6S_1, il faut étudier si cette entité :
- est autonome ;
- poursuit un but local ;
- doit interagir avec d'autres entités ;
- possède une vue partielle de l'environnement (ce qui signifie qu'elle ne peut percevoir qu'une partie réduite de son environnement, qu'elle ne possède qu'une connaissance partielle de lui) ;
- possède certaines capacités de négociation.

Les entités qui vérifient les trois premiers critères peuvent être vues comme des agents car elles respectent les caractéristiques minimales attribuées aux agents. Les deux autres critères sont facultatifs. Avant de conclure, des caractéristiques supplémentaires doivent être étudiées, c'est l'objet de l'étape suivante.

Exemple 3.18. *Voici une analyse des entités d'ETTO :*

- ▷ *Les entités actives :*
  - □ *Les enseignants (Teacher) :*
    - *Ils sont autonomes, car leur existence ne dépends pas de celles des autres et qui peuvent émettre de nouvelles contraintes de manière indéterminisme du point de vue des autres ;*
    - *Leur but local est de satisfaire leurs contraintes et de donner des cours ;*
    - *Ils peuvent interagir avec des étudiants (Students group) et des salles (Rooms) pour se mettre d'accord sur des horaires et des lieux d'enseignement ;*
    - *Ils ont une vue locale de leur environnement ;*
    - *Ils ont la capacité de négocier avec des étudiants et des salles.*
  - □ *Les étudiants (StudentsGroup) :*
    - *Ils sont autonomes ;*
    - *Leur but local est de satisfaire leurs contraintes ;*
    - *Ils peuvent interagir avec des enseignants (Teachers) et des salles (Rooms) pour se mettre d'accord sur des horaires et des lieux d'enseignement ;*
    - *Ils ont une vue locale de leur environnement ;*
    - *Ils ont la capacité de négocier avec des enseignants et des salles.*
  - □ *Le gestionnaire des enseignements (CoursesManager) :*
    - *Il est autonome ;*
    - *Il n'a pas de but local, il se contente de répondre aux requêtes et de communiquer les données du NPP aux enseignants et étudiants ;*
    - *Il interagit avec le NPP, les enseignants et les étudiants ;*
    - *Il a une vue locale de son environnement ;*
    - *Il n'a pas la capacité de négocier.*
  - □ *Le gestionnaire des salles (RoomsManager) :*

- *Il n'est pas autonome, il ne fera aucun changement sur l'environnement de son propre chef ;*
- *Il n'a pas de but local, il se contente de répondre aux requêtes et de communiquer les états et contraintes des salles aux enseignants et étudiants ;*
- *Il interagit avec les salles, les enseignants et les étudiants ;*
- *Il a une vue locale de son environnement ;*
- *Il n'a pas la capacité de négocier.*

▷ *Les entités passives :*

□ *Les salles (Rooms) :*

- *Elles ne sont pas autonomes, elles dépendant du gestionnaire de salles (Rooms Manager) ;*
- *Elles n'ont pas de but local ;*
- *Elles interagissent avec le gestionnaire de salles ;*
- *Elles ont une vue locale de leur environnement ;*
- *Elles n'ont pas de capacité de négociation.*

□ *Le plan pédagogique national (NPP, National Pedagogical Plan) :*

- *Il n'est pas autonome ;*
- *Il n'a pas de but local ;*
- *Il interagit avec les gestionnaire des enseignements ;*
- *Il a une vue locale de son environnement ;*
- *Il n'a pas la capacité de négocier.*

3.5.3.2 Identifier les entités potentiellement coopératives (S_2)

Durant ses interactions avec d'autres entités ou avec l'environnement, une entité peut rencontrer des problèmes : le protocole d'interaction peut ne pas être respecté ou l'interaction elle-même être une source d'erreurs ou d'échecs (par exemple, incompréhension entre les participants). Ces échecs peuvent être la résultante d'un environnement dynamique, de l'ouverture du système,etc.

Cette identification des entités potentiellement coopératives correspond à un traçage des interactions de l'environnement vers les composants du système (identifiés dans l'activité A_{10}) puis, de ces composants vers l'environnement. Les interactions sont celles présentes dans l'*architecture logicielle*. Des cas d'utilisations, jusqu'aux classes, en passant par les diagrammes de séquences, l'analyste doit tracer le traitement des échecs à la coopération. Les classes les prenant en compte en fin de parcours d'appel (ou événementiel) devront suivre un comportement coopératif pour les traiter au mieux. Une fois identifiées au niveau des classes, ces situations seront appelées Situations Non Coopératives (ou SNC) (voir §1.3). Les entités qui vérifient, au minimum, le dernier critère doivent être marquées durant l'étape suivante.

Exemple 3.19. *Seules deux entités sont autonomes, ont un but local et interagissent avec les autres entités. Ces deux entités sont les enseignants (Teacher) et les étudiants (StudentsGroup). Leurs contraintes peuvent changer en temps réel, de ce fait leur environnement est dynamique (les utilisateurs changent leurs contraintes de manière imprévisible). Le protocole de communication sera pleinement adapté aux besoins des entités (seules des contraintes portant sur les objets de l'emploi du temps sont exprimables). Elles ne rencontreront alors pas d'échec à la coopération à proprement parler.*

Cependant, ces entités peuvent se trouver dans des états dans lesquels elles ne respectent pas toutes leurs contraintes. Ces états sont vus comme des Situations Non Coopératives (SNC) puisqu'alors le système ne répondra pas aux besoins des utilisateurs.

3.5.3.3 Déterminer les agents (S_3)

Les entités identifiées dans les étapes précédentes et correspondant aux critères de coopération peuvent maintenant être considérées comme des agents. Leurs classes doivent donc être marquées du stéréotype «cooperative agent», qui indique qu'une classe représente un agent coopératif (voir §4.1). Les résultats seront consignés dans le document nommé *architecture logicielle*.

Toutefois, il est possible qu'aucun n'agent n'ait été identifié dans l'étape $A_{12}S_2$. Si l'analyste est arrivé à cette étape, il est passé par l'activité d'adéquation aux AMAS (A_{11}), pour savoir si le système peut être modélisé comme un AMAS, et donc posséder des agents. Trois cas sont possibles : le système est un AMAS (des entités déjà identifiées sont des agents), le système est composé d'un AMAS (des entités identifiées devront elles-même être décomposées en AMAS), ou bien les AMAS sont inutiles. Dans le premier cas, l'analyse devrait avoir identifié les agents dans l'étape précédente, sinon, il faut raffiner les interactions entre classes, grâce à l'activité A_{13}. Dans le deuxième cas, une analyse ultérieure devrait mener à une décomposition d'entités en AMAS (comme nous pourrons l'observer dans l'activité A_{14}).

3.5.4 Étudier les interactions entre entités (A_{13})

Le but de cette activité est de clarifier les interactions entre les entités précédemment identifiées dans l'activité $A_{10}S_1$.

Ces interactions (qui reflètent les cas d'utilisation identifiés durant l'activité A_7S_1) sont diversifiées et doivent être étudiées suivant trois axes :

1. Étudier les relations entre entités actives et passives (S_1) et les modéliser grâce à des diagrammes de séquences ou de collaboration UML ;
2. Étudier les relations entre entités actives (S_2) et les modéliser grâce à des diagrammes de séquences UML ;
3. Étudier les relations entre agents (S_3) et les modéliser grâce à des diagrammes de protocoles AUML (voir §2.1.3.1 et §4.2) [Odell *et al.*, 2001].

Les **entités actives** peuvent faire preuve d'autonomie ; par exemple, en apportant des modifications à leurs contraintes de fonctionnement. Elles sont capables d'agir de manière dynamique avec le système. Parmi ces entités actives se trouvent les agents composant le système. Les **entités passives** peuvent être considérées comme des ressources par le système. Leurs interactions avec le système se réduisent à l'obtention de données pour mener à bien la tâche qu'il doit accomplir. Les entités passives peuvent être utilisées ou modifiées par les entités actives mais ne changent pas de manière autonome.

Afin de tracer les échecs à la coopération (voir activité A_{13}), il est recommandé de marquer les interactions concernées par une notation dédiée (flèche pointillée) et des commen-

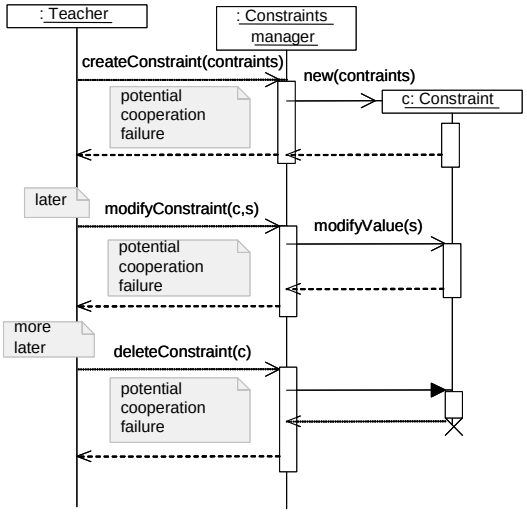

Figure 3.13 — Diagramme de séquences d'ETTO représentant les interactions entre un enseignant et le gestionnaire contraintes.

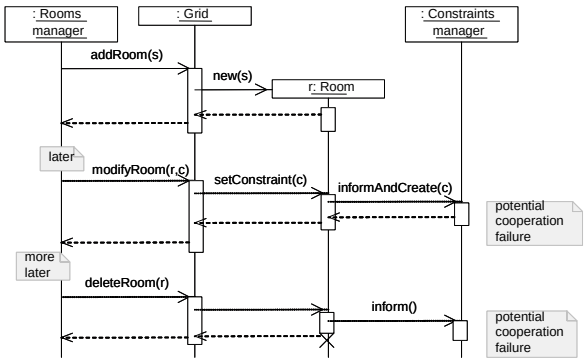

Figure 3.14 — Diagramme de séquences d'ETTO représentant les interactions entre le gestionnaire de salles et les salles.

taires appropriés.

Ces relations sont exprimées en utilisant les différents diagrammes UML ou AUML. Les résultats seront pris en compte pour mettre à jour le document *définition de l'environnement* et pour compléter le document d'*architecture logicielle*.

Exemple 3.20. *Les diagrammes de séquences qui concernent la visualisation de l'emploi du temps courant sont les mêmes pour toutes les entités actives. Le diagramme de séquences de la figure 3.13 concerne la gestion des contraintes. Il y a possibilité d'échec à la coopération avec le ConstraintsManager si les nouvelles contraintes sont incompatibles avec les anciennes ou bien si la destruction des contraintes entraîne une remise en question de l'emploi du temps pour obtenir de meilleurs compromis.*

Le diagramme de séquences pour les cas d'utilisation qui concernent le gestionnaire de salles est représenté dans la figure 3.14. Comme pour l'ajout de contraintes correspondants à une personne, il y a possibilité d'échec à la coopération avec le ConstraintsManager si les nouvelles contraintes d'une salle sont incompatibles avec les anciennes ou bien si la destruction des contraintes entraîne une remise en question de l'emploi du temps pour obtenir de meilleurs compromis.

À ce stade de l'analyse, nous ne sommes pas capables de décrire, en utilisant des diagrammes AUML, les interactions entre les enseignants (Teacher) et les groupes d'étudiants (StudentsGroup). En effet, l'algorithme de communication est trop complexe pour être exprimé ; ces agents semblent de granularité trop forte, comme il a été vu dans l'étape $A_{11}S_2$. Le deuxième niveau de décomposition nous aidera à déterminer ces interactions (voir exemple 3.23).

3.6 La conception (WD_4)

La conception doit définir une architecture détaillée pour le système en termes de paquetages, de sous-systèmes, d'objets et d'agents. Ce sont des activités importantes d'un point de

vue multi-agent du fait qu'une caractérisation récursive du système multi-agent est obtenue à ce stade. Cela implique différents processus de conception pour les différents niveaux du système identifiés.

Une fois les agents identifiés (activité A_{12}) et leurs relations définies (activité A_{13}), les concepteurs doivent étudier les moyens de communication entre agents (activité A_{15}) grâce à des diagramme de protocoles plus complets. ADELFE propose aussi un modèle d'agent coopératif, présenté au paragraphe 3.6.1 à remplir pour chaque agent identifié (activité A_{16}). Une activité de prototypage rapide (activité A_{17}) basée sur les machines à états finies a été aussi ajoutée au processus du RUP, afin de permettre aux concepteurs de vérifier le comportement (principalement social) des agents. Enfin, les concepteurs devront compléter les diagrammes de classes précédemment définis (activité A_{18}). De plus, une machine à état sera affectée à chaque classe afin de décrire son comportement.

Une architecture agent, fournie par ADELFE, permet au concepteur de doter les agents de modèles de compétences, de représentations du monde (représentations de l'agent sur lui-même, des autres et de son environnement), d'aptitudes, d'une attitude sociale et d'un langage d'interaction. À ce stade, une liste des situations non coopératives est établie.

Le modèle de conception est composé de classes d'objets classiques (dans des diagrammes de classes), de classes stéréotypées d'agents et d'interactions. Plus tard, la conception détaillée résulte de diagrammes d'états-transitions (pour modéliser le comportement des objets et des agents) et de diagrammes d'activités. Deux artefacts sont produits : l'*architecture détaillée* et les *langages d'interaction*.

Les activités de conception et leur ordonnancement sont résumés dans le diagramme d'activité SPEM de la figure 3.15.

3.6.1 Un modèle d'agent coopératif

Ce paragraphe expose le modèle d'agent que nous avons défini [Bernon *et al.*, 2003a; Picard *et al.*, 2004], utilisant la coopération comme moteur de l'auto-organisation du système multi-agent afin d'atteindre l'adéquation fonctionnelle. Les agents coopératifs sont composés de différents modules représentant une partition de leurs capacités physiques, cognitives ou sociales (voir figure 3.16). Chaque module représente une ressource spécifique pour l'agent utilisée lors de son cycle de vie "perception-décision-action".

3.6.1.1 Les différents modules

Les interactions entre agents sont régies par deux modules. Le *module de perception* représente les données en entrée que l'agent reçoit de son environnement. Elles peuvent être de natures et de types différents : entiers, booléens ou même des messages structurés de haut niveau (dans le cas d'une boîte aux lettres). Le *module d'action* représente les sorties et le moyen pour l'agent d'agir sur son environnement physique, social ou sur lui-même dans le cas d'une action d'apprentissage. Comme pour les perceptions, les actions peuvent être de granularité variable : activation simple d'effecteurs pour un robot, par exemple, ou envoi de messages à fort contenu sémantique pour des agents sociaux.

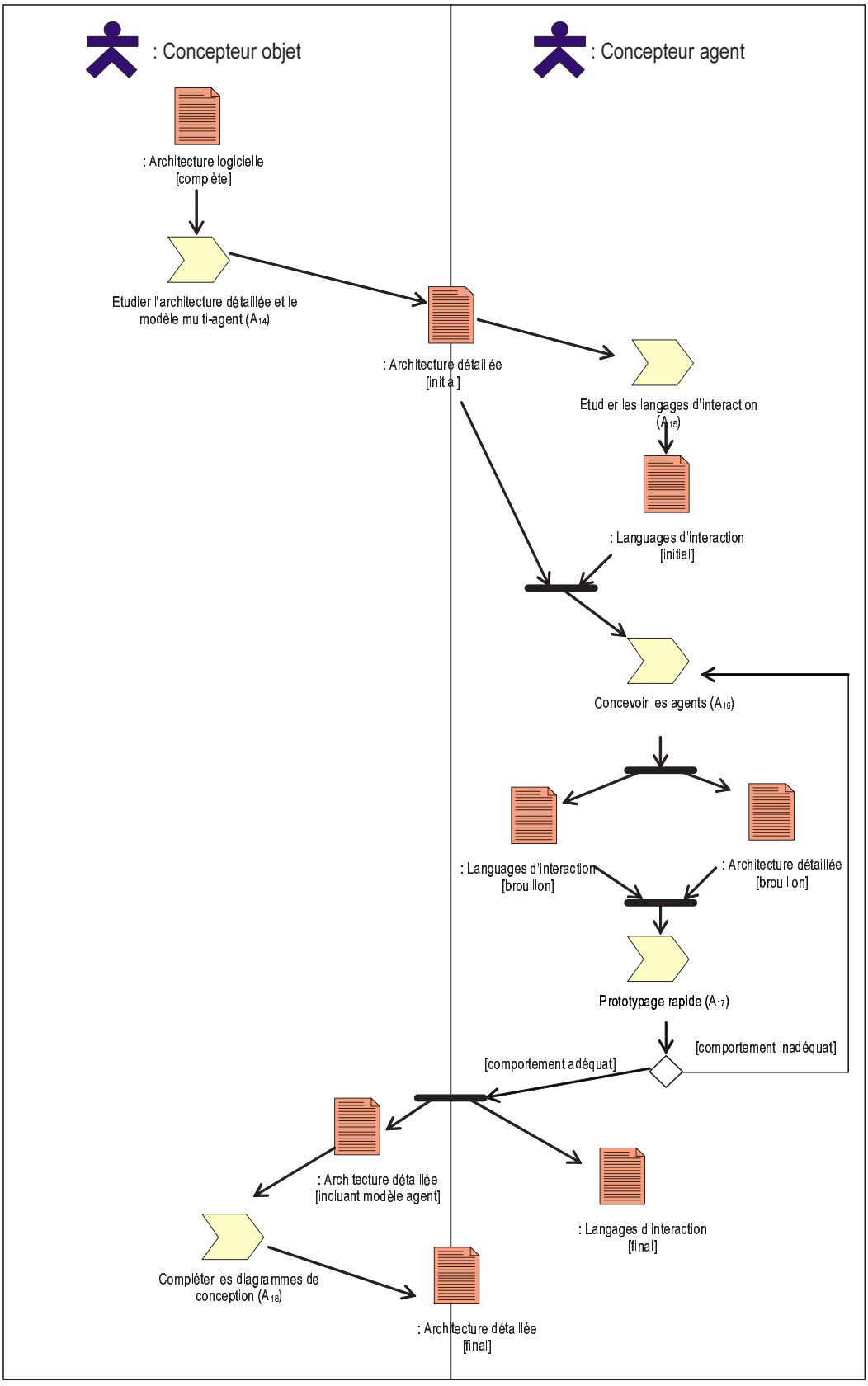

Figure 3.15 — La définition de travaux WD_4 : la conception.

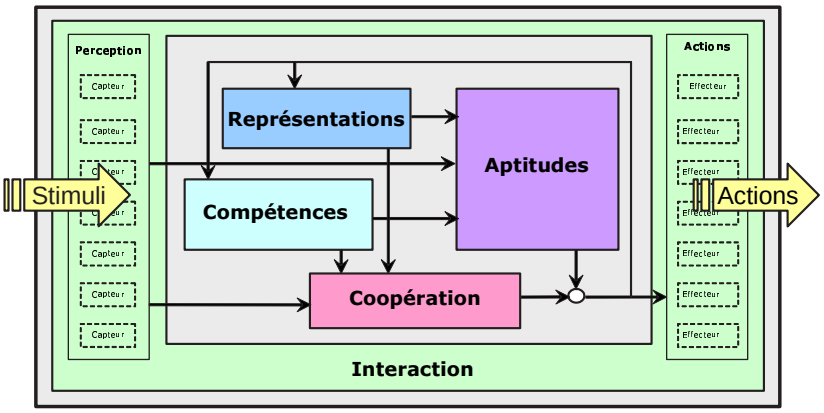

Figure 3.16 — Les différents modules d'un agent coopératif et leurs dépendances

Même si les agents coopératifs essaient principalement d'éviter les situations non coopératives, ils ont une ou plusieurs tâches à remplir en temps normal. Les compétences représentent un champ de connaissances permettant à l'agent de réaliser sa fonction partielle – en tant que partie d'un système réalisant une fonction globale. Aucune contrainte technique n'est requise pour concevoir et développer ce *module de compétences*. Par exemple, les compétences peuvent être implantées comme une base de faits et de règles sur un domaine particulier. Il peut aussi être décomposé en un système multi-agent de niveau inférieur afin de doter le système de capacités d'apprentissage par auto-organisation, comme dans ABROSE, par exemple, où les compétences d'agents négociant des services sont décomposées en réseaux adaptatifs de termes [Gleizes *et al.*, 2000]. Dans ce module, nous pouvons aussi intégrer les caractéristiques intrinsèques aux agents, comme par exemple, leur poids, leur couleur.

Comme pour les compétences, le *module de représentations* peut être implanté sous forme d'une base de faits et de règles, mais il porte sur la connaissance de l'environnement (physique ou social) et sur l'agent lui-même. Les croyances sur les autres agents sont considérées comme des représentations. Les représentations peuvent être décomposées en systèmes multi-agents de plus bas niveau pour obtenir des représentations dynamiques par le biais de l'auto-organisation.

Les aptitudes représentent les capacités de l'agent à raisonner sur ses perceptions, ses compétences et ses représentations – pour interpréter des messages, par exemple. Le *module d'aptitudes* peut être un moteur d'inférences raisonnant sur des bases de compétences ou de représentations. Pour un état donné de compétences, de représentations et de perceptions, ce module doit en déduire une action à exécuter. Les cas pour lesquels aucune ou plusieurs actions sont proposées correspondent à des situations non coopératives et doivent être prises en compte (voir paragraphe suivant).

L'attitude coopérative de l'agent est implantée dans le *module de coopération*. Comme le module d'aptitudes, ce module doit fournir une action en fonction d'un état courant de perceptions, de compétences et de représentations, mais seulement si l'agent est en situation non coopérative. Par conséquent, les agents doivent posséder des règles de détection de situations non coopératives et y associer des actions de résolution ou de prévention. Plusieurs types de situations non coopératives ont été identifiés et peuvent apparaître à n'importe

quel instant du cycle de vie de l'agent (voir paragraphe 1.3).

3.6.1.2 Fonctionnement interne

Lors de la phase de perception, le module de perceptions met à jour les valeurs des entrées. Ces données peuvent impliquer des changements directs dans les compétences ou les représentations. Une fois les connaissances mises à jour, la phase de décision doit conduire au choix d'une action. Durant cette phase, les modules d'aptitudes et de coopération fonctionnent en parallèle. Le premier doit fournir une action correspondant à un comportement nominal. Le second doit détecter si l'agent est en situation non coopérative et s'il doit agir en conséquence. Dans ce cas, l'action coopérative subsume l'action nominale. Si aucune action coopérative n'est proposée, le robot est dans un état coopératif et peut donc agir de façon normale. Une fois une action choisie, l'agent agit lors de la phase d'action afin d'activer ses effecteurs ou changer ses connaissances.

3.6.2 Étudier l'architecture détaillée et le modèle multi-agent (A_{14})

Si le système a besoin d'être conçu comme un système multi-agent récursif (voir activité A_{11}) et est alors composé de plusieurs niveaux (celui du système, celui des composants, ...), le processus d'analyse et de conception doit être appliqué à chacun de ces niveaux.

Le principal objectif de cette activité est de définir l'architecture détaillée du système en termes de paquetages, classes, objets et agents nécessaires ; d'éventuellement raffiner cette architecture en utilisant des patrons de conception (*design patterns*) et/ou des composants réutilisables puis de créer des diagrammes de classes et de composants.

Les résultats de cette activité seront inscrits dans une première version du document intitulé *architecture détaillée*.

Exemple 3.21. *Comme il a été vu dans l'activité A_{11}, le système a besoin d'un second niveau d'AMAS. Ainsi un enseignant (Teacher) doit être considéré comme un système à développer, il en est de même pour les groupes d'étudiants (StudentsGroup). En effet, ces deux types d'entité actives doivent gérer en parallèle plusieurs contraintes et buts : par exemple, un groupe d'étudiants doit suivre plusieurs cours. Aussi, nous appliquons à nouveau le processus ADELFE depuis l'activité A_1 jusqu'à l'activité A_{13}). En arrivant à l'activité A_{14}, le but des enseignants et des groupes d'étudiants sont décomposés en sous-buts correspondant chacun à l'un de leurs enseignements (à donner ou à recevoir). Le résultat est la définition d'un nouvel agent : l'agent de réservation (ou BookingAgent).*

Un enseignant ou un groupe d'étudiants est composé d'agents de réservation (BookingAgent), un par cours qu'il doit donner ou recevoir. Le rôle d'un tel agent est de se déplacer sur la grille afin de trouver un partenaire et une salle pour le cours qui lui a été assigné. Comme les contraintes ne sont plus centralisées, chaque agent de réservation possède un gestionnaire de contraintes (Constraints-Manager) afin de mener la négociation avec des partenaires potentiels, suivant ces propres contraintes (héritées de l'enseignant ou du groupe d'étudiants pour lequel il travaille).

3.6.2.1 Déterminer les paquetages (S_1)

L'objet de cette étape est d'identifier les différents paquetages afin de regrouper les classes par domaine. C'est une étape classique en conception objet, permettant notamment de scinder le développement du système, et d'augmenter les possibilités de réutilisation.

Exemple 3.22. *Nous partageons le système en quatre paquetages :*
- *Un paquetage agent pour gérer tous les BookingAgents, les enseignants (Teacher) et les groupes d'étudiants (StudentsGroups) ;*
- *Un paquetage grid (grille) pour gérer ses trois dimensions (la salle, l'horaire et le jour) ;*
- *Un paquetage constraint (contraintes) qui doit être externe afin de permettre l'accès aux salles et aux agents ;*
- *Un paquetage interface pour permettre à l'utilisateur d'agir sur le système et de jouer le rôle de la personne en charge des salles et de la personne en charge des enseignements.*

3.6.2.2 Déterminer les classes (S_2)

Durant cette étape, les différentes classes apparaissant dans le diagramme de classes préliminaire (voir l'étape $A_{10}S_3$) sont réparties dans chacun des paquetages précédemment identifiés. Il se peut que ces classes aient besoin d'être raffinées ou complétées, notamment à cause d'une décomposition en systèmes multi-agents de certaines d'entre elles.

Exemple 3.23. *Voici l'affectation des classes, et des classes identifiées lors de la décomposition, aux paquetages précédemment définis :*
- ▷ *Le paquetage agent : il contient les trois classes qui correspondent à des agents :*
  - □ *Teacher (enseignant) : représentant les enseignants ;*
  - □ *StudentsGroup (groupe d'étudiants) : qui représente les groupes d'étudiants ;*
  - □ *BookingAgent (agent de réservation) : qui représente un enseignant ou un groupe d'étudiants pour un cours.*
- ▷ *Le paquetage grid : la grille peut avoir plusieurs dimensions (salle, heure, jour). Une classe unique grid n'est alors pas suffisante pour modéliser la grille. Il faut donc ajouter d'autres classes :*
  - □ *Viewer (afficheur) : pour visualiser la grille ;*
  - □ *Grid (grille) : la classe "grid" ;*
  - □ *Cell (cellule) : un élément minimal de la grille ;*
  - □ *Size (taille) : afin de définir le nombre de dimensions de la grille ;*
  - □ *SizingObject (objet de dimensionnement) : pour définir le type d'une dimension. Trois classes peuvent hériter de cette classe :*
    - *Room (salle) ;*
    - *TimeSlot (créneau horaire) ;*
    - *Day (jour).*
  - □ *Coordinates (coordonnées) : afin de définir les coordonnés d'une cellule dans la grille.*
- ▷ *Le paquetage constraint : en partant des scenarii proposés, de nouvelles classes, héritant de la classe Constraint, ont été définies. Les classes du paquetage constraint sont donc :*
  - □ *ConstraintManager (gestionnaire de contraintes) : afin de gérer les contraintes ;*
  - □ *Constraint (contrainte) : la classe dont dérivent les classes suivantes :*

- *Capacity (capacité) : afin de définir la contrainte de capacité d'une salle ;*
- *Unavailability (indisponibilité) : afin de déterminer une indisponibilité ;*
- *Projector (projecteur) : afin de définir une contrainte sur la nécessité d'un (rétro ou vidéo) projecteur.*

▷ *Le paquetage interface : Dans le futur, l'utilisateur configurera l'application via une fenêtre de configuration. Cette fenêtre lui permettra de contrôler les gestionnaires des enseignements et de contraintes. Et un convertisseur permettra la gestion de l'application via des fichiers, au lieu de la fenêtre de configuration, afin d'exécuter une batterie de tests automatisables. À ce moment là, les classes du paquetage interface sont les suivantes :*

- □ *RoomsManager (gesionnaire des salles) ;*
- □ *Converter (convertisseur) ;*
- □ *ConfigurationWindow (fenêtre de configuration) ;*
- □ *CoursesManager (gestionnaire des enseignements).*

3.6.2.3 Utiliser les patrons de conception (S_3)

Dans cette étape, il faut déterminer si l'architecture précédemment identifiée peut être raffinée en utilisant certains patrons de conception (*design patterns*) et/ou tout autre composant réutilisable. C'est une étape très classique en conception orientée objet qui exploite les propriétés de modularité des objets.

Exemple 3.24. *Dans ETTO, nous n'utilisons aucun patron.*

3.6.2.4 Élaborer les diagrammes de composants et de classes (S_4)

Pour chaque paquetage créé, un diagramme de composants et/ou un diagramme de classes est élaboré. Le but de cette étape est de finaliser l'étape précédente en liant les différents composants et paquetages dans des diagrammes UML. Le but est aussi de produire une version préliminaire du document *architecture détaillée*.

Exemple 3.25. *Les diagrammes de classes pour chacun des paquetages sont représentés dans les figures 3.17, 3.18, 3.19 et 3.20.*

3.6.3 Étudier les langages d'interaction (A_{15})

Cette activité consiste à définir la manière dont les agents vont interagir. Il est à noter que définir un langage d'interaction est inutile si aucune communication directe n'est utilisée par les agents (par exemple, s'ils ne communiquent que de manière indirecte via l'environnement).

Si les agents interagissent pour communiquer, pour chaque scénario (défini dans les activités A_7 et A_{13}), il faut décrire les échanges d'informations entre agents. Techniquement, ces protocoles seront spécifiés via des diagrammes de protocoles utilisant la notation AUML (voir §2.1.3.1 et §4.2) [Odell *et al.*, 2001].

Cette étape produit une version initiale du document *langage d'interaction*. Les langages qui permettent les interactions entre agents peuvent s'implanter grâce à un ensemble de

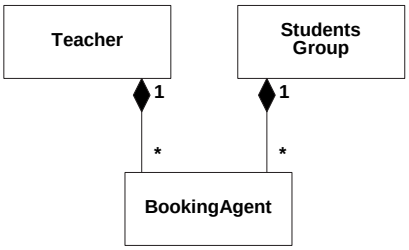

Figure 3.17 — Diagramme de classes du paquetage agent.

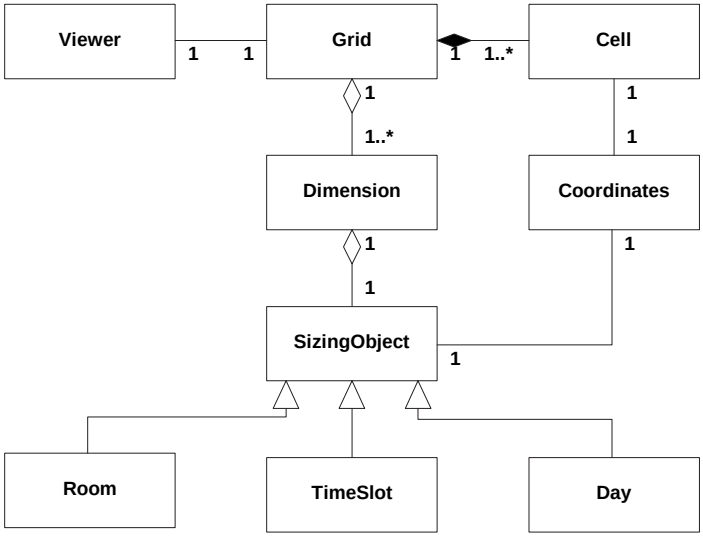

Figure 3.18 — Diagramme de classes du paquetage grid.

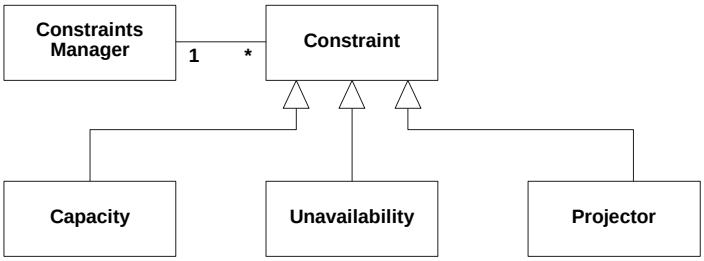

Figure 3.19 — Diagramme de classes du paquetage constraint.

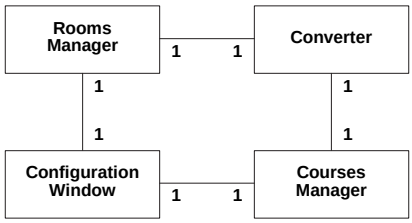

Figure 3.20 — Diagramme de classes du paquetage interface.

classes ou un patron de conception (design pattern), comprenant certains outils de communication spécifiques aux agents tels qu'une implantation des ACL de FIPA. Comme se sont des modèles génériques, les diagrammes de protocoles sont rattachés aux paquetages et non aux classes.

Exemple 3.26. *La figure 3.21 présente le diagramme de protocoles modélisant les interactions entre les BookingAgents de deux enseignants. L'un a réservé une cellule (rôle OccupyingTeacherAgent) et le second explore la même cellule (rôle ExploringTeacherAgent). Ce protocole spécifie la négociation mise en place pour déterminer qui restera dans la cellule. Cette négociation peut aboutir au départ de l'un ou de l'autre des agents, en fonction de leurs contraintes.*

3.6.4 Concevoir les agents (A_{16})

Dans cette activité, il est possible de raffiner les classes stéréotypées par «cooperative agent» définies dans le document intitulé *architecture détaillée* et le document *langages d'interaction*.

Cette activité permet d'instancier le modèle d'agent coopératif (voir §3.6.1) proposé dans ADELFE, c.-à-d. de remplir chacun des modules par des méthodes appropriées représentant les compétences, les aptitudes ou toute autre caractéristique d'un agent. Cette activité est

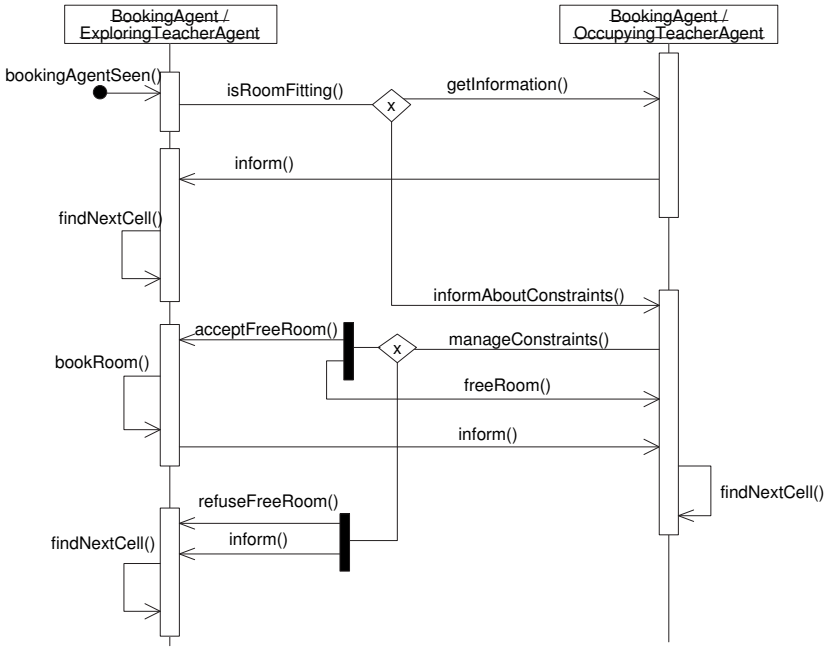

Figure 3.21 — Un exemple de protocole d'interactions entre deux agents de réservation (BookingAgents) d'ETTO.

une étape-clé, puisque c'est celle durant laquelle le concepteur définit les agents, et surtout leur comportement coopératif par l'attribution de règles de coopération.

3.6.4.1 Remplir les modules des agents

Afin de préciser qu'une méthode (ou attribut) appartient à tel ou tel module, et donc qu'elle (ou il) doit respecter certaines règles de bon usage, ADELFE fournit des stéréotypes UML (voir paragraphe 4). Toute classe représentant un agent coopératif, et donc stéréotypée «cooperative agent», doit hériter d'une classe fournie, CooperativeAgent, possédant trois méthodes, perceive, decide et act, simulant le cycle de vie des agents. Ces méthodes sont abstraites et doivent être codées pour chaque classe d'agent.

Il est conseillé de commencer par les compétences qui sont des connaissances sur un domaine permettant à un agent d'exécuter des actions. Les compétence traduisent aussi les buts des agents. Il faut alors donner les méthodes et/ou les attributs qui décrivent les compétences d'un agent et les stéréotyper comme «skill».

Exemple 3.27. *L'objectif individuel d'un Teacher ou d'un StudentGroup est de trouver tous les partenaires et les salles lui permettant d'assurer ou de recevoir tous ses cours, tout en respectant au maximum les contraintes imposées.*

Les compétences d'un agent enseignant (Teacher) ou d'un groupe d'étudiants (StudentsGroup) sont les suivantes :

- *la gestion de contraintes ;*
- *la gestion de ses agents de réservation ;*
- *la gestion de sa boîte aux lettres.*

L'objectif individuel d'un BookingAgent est de trouver un partenaire et une salle à un créneau

donné où s'établir tout en respectant au maximum les contraintes qui lui sont imposées par le groupe d'étudiant ou l'enseignant pour lequel il travaille.

Les compétences pour un agent de réservation (BookingAgent) sont les suivantes :
- *la capacité de se déplacer sur la grille ;*
- *la réservation de cellules ;*
- *la gestion de partenariat entre BookingAgents étudiants et enseignants ;*
- *la gestion de sa boîte aux lettres.*

Un agent possède aussi des connaissances, qui sont appelées représentations, sur l'environnement (social ou physique) ou sur lui-même. Ces représentations peuvent être mises à jour grâce aux perceptions que l'agent possède. Il convient donc de définir, dans un premier temps, ces perceptions, c.-à-d. ce que peut savoir l'agent sur son environnement.

Exemple 3.28. *Les perceptions d'un agent de réservation (BookingAgent) concernent les objets suivants :*
- *les messages qui lui sont adressés ;*
- *la cellule dans laquelle il se situe ;*
- *les informations sur la salle dans laquelle il se situe ;*
- *les agents de réservations proches (dans la même salle).*

Les représentations que possèdent un agent de réservation portent sur :
- *ses propres contraintes ;*
- *l'état de ses réservations ;*
- *l'état de ses partenariats pour établir des cours à des créneaux donnés ;*
- *les contraintes de ses "frères" (agent de réservation travaillant pour le même agent Teacher ou StudentGroup ;*
- *ses rencontres passées.*

Grâce à ses données (compétences, perceptions et représentation), l'agent doit pouvoir produire des raisonnements. Cette production est mise en oeuvre par les aptitudes.

Exemple 3.29. *Les aptitudes que possèdent un agent de réservation (BookingAgent) sont :*
- *la gestion des contraintes ;*
- *la gestion des messages ;*
- *la gestion des réservations ;*
- *la gestion des partenaires.*

En fonction de ses perceptions, de ses représentations et de ses compétences, les aptitudes d'un agent vont décider de l'exécution d'actions afin de modifier son environnement.

Exemple 3.30. *Les actions d'un agent de réservation (BookingAgent) sont les suivantes :*
- *réserver une salle ;*
- *libérer une salle ;*
- *signer un partenariat ;*
- *rompre un partenariat ;*
- *envoyer un message.*

Les interactions entre agents sont exprimées grâce à des protocoles A-UML. Remplir le module d'interaction d'un agent revient à lui attribuer des protocoles génériques définis lors de l'activité A_{15}. Il est possible que l'étude des compétences, ou de tout autre donnée de l'agent, implique des modifications des langages d'interaction, par l'ajout de méthodes de communication, par exemple. Ceci nécessite de travailler en parallèle sur cette activité et l'activité A_{15}.

De plus, travailler sur les agents peut engendrer des modifications de l'architecture objet du système. En effet, par regroupements, il est possible de voir apparaître des relations d'héritage ou de composition. Bien sûr, il est aussi possible qu'une classe d'agent possède des attributs ou des méthodes autres que ceux apparaissant dans les modules. Ces traits (*features* en anglais) sont de préférences stéréotypés «characteristic» lorsqu'ils reflètent des caractéristiques observables par les autres agents (leur couleur, par exemple). Ils peuvent aussi ne pas être stéréotypés s'il s'agit d'attributs ou de méthodes non directement liés au domaine, comme par exemple des attributs d'observation (ou *monitoring*) des agents pour effectuer des statistiques sur le fonctionnement du système.

Exemple 3.31. *Dans ETTO, les agents Teacher et StudentsGroup ont des modules similaires : leurs compétences, aptitudes, perceptions, et représentations sont identiques. Il est donc possible de créer une classe regroupant ces fonctionnalités et dont les deux vont hériter : la classe RepresentativeAgent. Cette classe modélise un agent représentant un acteur humain (enseignant ou étudiant). La principale différence entre un enseignant et un groupe d'étudiants est la capacité de modifier ou non les contraintes (seuls les enseignants peuvent exprimer leurs indisponibilités par exemple). Ceci conduit au diagramme de classes de la figure 3.22.*

3.6.4.2 Définir l'attitude coopérative et les situations non coopératives (SNC)

Le dernier module des agents, mais le plus important, est le module de coopération. Concevoir un agent coopératif consiste essentiellement à donner des règles qui permettent à l'agent d'avoir une attitude coopérative, c.-à-d. comment détecter et faire disparaître les situations non coopératives afin de devenir plus coopératif et donc, pour le système, de donner une fonction plus adéquate.

Une situation non coopérative est définie par :

L'**état** dans lequel l'agent se trouve durant la détection de la SNC. Cet état peut être défini par un ensemble de valeurs d'attributs ou de résultats de méthodes qui peuvent être stéréotypés «perception», «characteristic» ou «representation» ;

La **description** textuelle de la SNC ;

Les **conditions** qui décrivent les différents éléments qui permettent de détecter localement la SNC. Les méthodes et attributs utilisés pour exprimer les conditions doivent être stéréotypés «perception» ou «representation» (voir le fonctionnement interne d'un agent coopératif au paragraphe 3.6.1) ;

Les **actions** liées à la SNC. Les actions décrivent ce que l'agent doit faire pour faire disparaître cette SNC. Les méthodes et attributs utilisés pour exprimer ces actions doivent être stéréotypés «action» ou «skill».

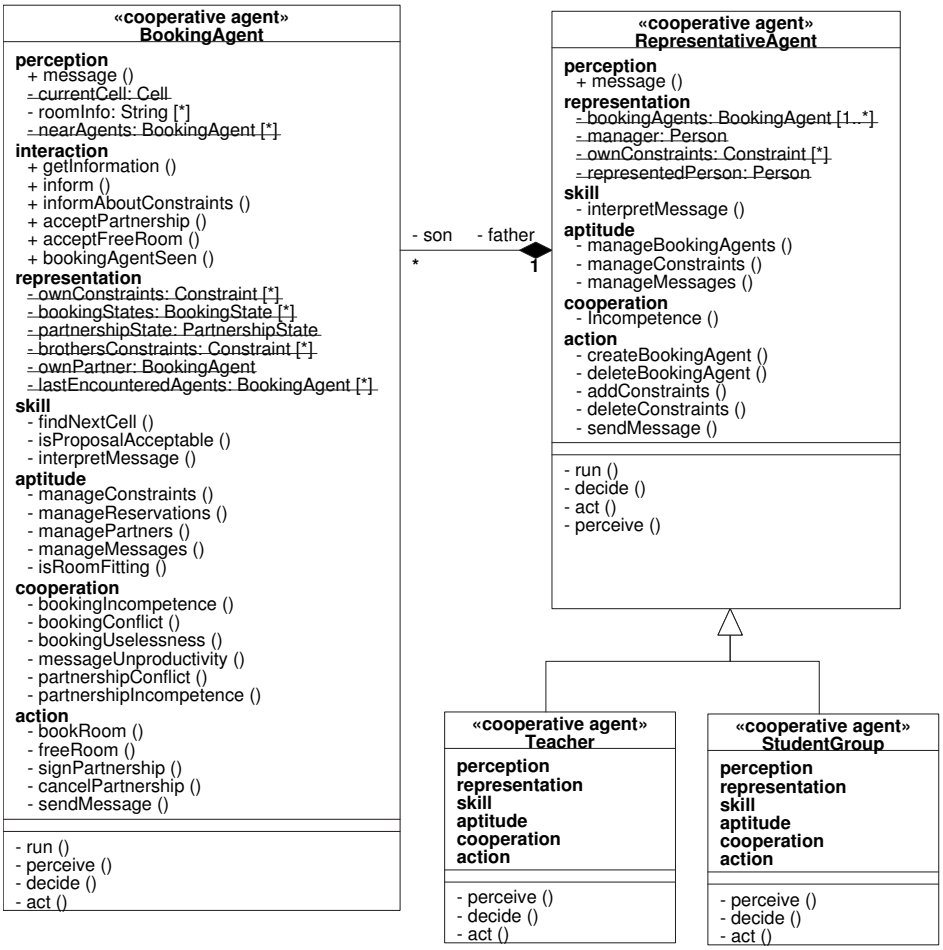

Figure 3.22 — Diagramme de classes d'agents pour ETTO.

Il est possible que la définition d'une SNC implique de raffiner la classe, par l'ajout d'attributs ou de méthodes nécessaires à sa détection ou aux actions à mettre en œuvre pour la faire disparaître.

A chaque SNC sera associée au moins une méthode stéréotypée «cooperation». Cette méthode correspond à la détection de la SNC et sera exprimée en utilisant l'état et les conditions i.e. les méthodes et attributs qui sont stéréotypés «perception», «representation» ou «characteristic».

Si plusieurs actions sont possibles afin de faire disparaître la SNC détectée, il faut définir une autre méthode afin de choisir l'action qui sera effectuée. Cette méthode est stéréotypée «cooperation». Si une seule action est possible, la définition de cette seconde méthode est inutile : cette action unique sera toujours exécutée.

Comme nous l'avons vu dans le paragraphe 1.3, aucune définition formelle de niveau de précision suffisant ne permet de définir de manière automatique les SNC. Cependant, une possibilité pour aider à identifier les SNC de chaque agent est d'analyser l'agent suivant deux dimensions : celle du temps (son cycle de vie) et celle de ses états (valeur pour un attribut donné). La table 3.2 montre un exemple générique d'un tel schéma d'analyse.

Exemple 3.32. *Bien que les RepresentativeAgents soient vus comme des agents AMAS, nous n'ar-*

	Condition 1 non remplie		Condition 1 remplie	
	Condition 2 non remplie	**Condition 2 remplie**	**Condition 2 non remplie**	**Condition 2 remplie**
Perception	• Incompréhension ? • Ambiguïté ?	• Incompréhension ? • Ambiguïté ?	• Incompréhension ? • Ambiguïté ?	• Incompréhension ? • Ambiguïté ?
Décision	• Incompétence ? • Improductivité ?	• Incompétence ? • Improductivité ?	• Incompétence ? • Improductivité ?	• Incompétence ? • Improductivité ?
Action	• Concurrence ? • Conflit ? • Inutilité ?	• Concurrence ? • Conflit ? • Inutilité ?	• Concurrence ? • Conflit ? • Inutilité ?	• Concurrence ? • Conflit ? • Inutilité ?

Tableau 3.2 — Un schéma d'analyse des agents pour l'identification des SNC.

	Pas de réservation		Avec réservation	
	Pas de partenaire	**Avec partenaire**	**Pas de partenaire**	**Avec partenaire**
Perception	**Aucune SNC lors de la phase de perception**			
Décision	• **Incompétence** Incompétence de partenariat • **Improductivité** Improductivité de message	• **Incompétence** Incompétence de partenariat • **Improductivité** Improductivité de message	• **Incompétence** Incompétence de partenariat Incompétence de réservation • **Improductivité** Improductivité de message	• **Incompétence** Incompétence de partenariat Incompétence de réservation • **Improductivité** Improductivité de message
Action	• **Conflit** Conflit de partenariat Conflit de réservation	• **Conflit** Conflit de partenariat Conflit de réservation • **Inutilité** Inutilité de réservation	• **Conflit** Conflit de partenariat Conflit de réservation	• **Conflit** Conflit de partenariat Conflit de réservation • **Inutilité** Inutilité de réservation

Tableau 3.3 — Analyse des SNC pour un BookingAgent dans ETTO.

rivons pas à leur attribuer de situations non coopératives. La conception des ces agents n'est pas pour autant remise en cause, mais toutes leurs actions, leurs interactions et leur évolution sont en fait reléguées au niveau inférieur, celui des BookingAgent.

Au-delà des situations non coopératives, grâce aux scénarios imaginés, cette étape nous permet de déterminer le fonctionnement de l'agent et notamment la manière dont il effectue des réservations et des partenariats. Chaque contrainte est pondérée par la difficulté que doit avoir l'agent à la relâcher. Plus le poids est important, plus le coût du relâchement de la contrainte est élevé. Évidemment, le but est de minimiser le coût total des relâchements dans le système. Au niveau local, cela signifie que chaque BookingAgent (BA) cherche à minimiser son coût personnel. Toutefois, en cas de conflit, l'intérêt personnel passe au second plan et c'est le BA qui coûte le moins cher à la collectivité qui l'emporte.

Cependant, ces coûts sont fixes et pour une organisation donnée, il est possible qu'un agent ne

puisse pas trouver sa place sur la grille, les emplacements qu'il pourrait occuper étant réservés par des agents plus prioritaires. De ce fait, alors que ces derniers agents pourraient se placer autre part, ils empêchent la solution d'émerger. Pour remédier à cela, nous dotons un agent de la capacité à apprendre, au fur et à mesure, la difficulté qu'il a à vérifier ses contraintes. Ainsi, en modulant les poids des contraintes (qui ne sont testées que localement) par ce coefficient (global et évoluant au fil du temps), les priorités relatives des agents vont se modifier. L'agent sans réservation, estimant progressivement qu'il a de plus en plus de difficultés à se placer, va finir par être prioritaire par rapport aux autres BA. Le cas de figure abordé ci-dessus se résorbera par la remise en cause des agents occupant des positions non essentielles pour eux.

En outre, nous décidons que, du moment que la salle est compatible avec ses besoins, un BA sans réservation doit la réserver quel que soit le coût des réservations qu'il doit relâcher. Par la suite, il ne réservera que les salles lui permettant d'améliorer son coût courant. Il en va de même pour les partenariats. Cela permet de trouver une solution non optimale très rapidement, la suite du traitement sert uniquement à améliorer le score global, et donc à améliorer la solution.

La table 3.3 résume les SNC possibles pour un BookingAgent. Les six NCS identifiées sont décrites dans les tables suivantes :

Nom	**Incompétence de partenariat**
État	Indifférent
Description	Le BookingAgent (BA) courant rencontre un autre BA avec lequel un partenariat ne serait pas intéressant.
Condition(s)	Le BA courant est en présence d'un autre BA ***ET*** (cet autre BA n'est pas du bon type ***OU*** le coût du partenariat avec lui serait supérieur ou égal à celui du BA courant).
Action(s)	Le BA courant mémorise l'agent rencontré et se déplace afin de rencontrer de nouveaux BA.

Nom	**Incompétence de réservation**
État	Avec réservation
Description	Le BA (BookingAgent) se trouve dans une cellule dont la réservation ne serait pas intéressante.
Condition(s)	Le BA se trouve dans une cellule ***ET*** le coût de sa réservation est supérieur ou égal à celui de la réservation courante du BA.
Action(s)	Le BA mémorise la cellule et se déplace afin de visiter d'autres cellules.

Nom	**Improductivité de message**
État	Indifférent
Description	Le BA (BookingAgent) reçoit un message dont il ne devrait pas être le destinataire.
Condition(s)	Le BA possède un message à traiter ***ET*** il s'aperçoit que le destinataire de ce message doit être le partenaire de l'émetteur.
Action(s)	Le BA retourne le message à l'envoyeur.

Nom	**Conflit de partenariat**
État	Indifférent
Description	Le BA courant souhaite passer un partenariat avec un agent déjà engagé.
Condition(s)	Le BA courant est en présence d'un autre BA ***ET*** ce dernier a déjà un partenaire ***ET*** le BA courant est qualifié pour ce partenariat.
Action(s)	***SI*** le coût du nouveau partenariat est inférieur à celui du partenariat déjà établi ***ALORS*** (le BA passe le partenariat ***ET*** rompt celui en cours ***ET*** en informe ces frères) ***SINON*** le BA se déplace afin de rencontrer d'autres agents.

Nom	**Conflit de réservation**
État	Indifférent
Description	Le BA souhaite réserver une cellule déjà prise.
Condition(s)	Le BA est dans une salle ***ET*** (cette dernière est déjà réservée ***OU*** un frère a déjà réservé pour le même créneau) ***ET*** le BA est compétent pour la réservation.
Action(s)	***SI*** le coût de la nouvelle réservation est inférieur à celui de la réservation déjà établie ***ALORS*** (le BA réserve la cellule ***ET*** annule son ancienne réservation ***ET*** informe son hypothétique partenaire ***ET*** informe ses frères) ***SINON*** le BA se déplace afin de visiter d'autres cellules.

Nom	**Inutilité de réservation**
État	Avec partenaire
Description	Le BA est en présence de son partenaire. Que ce dernier réussisse à réserver ou non, le BA sera inefficace. Il est inutile qu'il traite la cellule.
Condition(s)	Le BA se trouve dans une salle ***ET*** son partenaire s'y trouve aussi.
Action(s)	Le BA traite les rencontres de la cellule et se déplace pour visiter d'autres cellules.

3.6.5 Prototypage rapide (A_{17})

Le but de cette activité est de tester le comportement des agents en cours de fonctionnement. Ce fonctionnement ne doit être qu'un fonctionnement simulé et limité en terme de développement. Le prototype réalisé doit permettre d'identifier des manques au niveau du comportement des agents, et surtout au niveau de leur attitude coopérative.

Quels aspects peut-on tester ?
- les **protocoles**, afin de trouver si certains interblocages peuvent se produire ou si certains protocoles sont inutiles ou incohérents ;
- les **méthodes**, pour tester leur possible inutilité, leur exhaustivité, ...
- le **comportement** général des agents, pour contrôler si ce comportement correspond à ce qui était attendu, ou bien si le comportement coopératif est suffisant (si la liste des SNC est exhaustive).

Comment réaliser ces tests ?
- tout d'abord, en créant l'environnement de simulation,
- puis, en implantant, les méthodes à tester.

La réalisation d'un prototype nécessite de se munir d'outils adéquats. Dans l'idéal, un outil dans lequel des agents coopératifs génériques peuvent être instanciés afin de simuler les comportements souhaités. Ceci nécessite de se munir d'un modèle d'agent assez détaillé pour pouvoir être exécuté sans modification. Néanmoins, ceci nécessite de faire des choix théoriques peut-être inadéquats. Il faudra attendre une théorie plus complète des AMAS. En l'état, ADELFE propose une simulation du comportement social des agents communicants – décrit par des diagrammes de protocole. Cet outil est décrit dans le paragraphe 4.3 dédié aux outils associés à ADELFE, et notamment OpenTool.

De plus, une fois les simulations exécutées, les résultats doivent être analysés. Ils doivent permettre d'identifier des anomalies comportementales ou des manques. La détection de tels problèmes n'est pas forcément aisée. Là encore, des outils seront nécessaires. Pour l'instant, ADELFE n'en fournit aucun.

Dans l'idéal, ces travaux devraient être :

– une phase d'analyse et d'extraction de théories *a priori* à partir du comportement des agents en exécution permettant de modifier/améliorer/optimiser leur comportement global ;
– un terrain de test pour les agents afin de détecter de nouvelles situations non coopératives (leur exhaustivité ne pouvant être assurée par la théorie).

Tous ces tests peuvent induire des modifications des documents *langages d'interaction* et *architecture détaillée*. En effet, si le comportement est inadéquat, il convient de revenir à l'activité A_{16} précédente afin d'améliorer le comportement des agents.

3.6.6 Compléter les diagrammes de conception (A_{18})

Dans cette activité, le document *architecture détaillée* doit être finalisé de manière à achever les activités de conception. Une fois l'*architecture détaillée* terminée, l'implantation pourra commencer.

Dans cette activité, il faut compléter le travail sur les diagrammes de classes précédemment établis. Ces diagrammes doivent être enrichis en tenant compte de tout ce qui a été effectué durant les activités précédentes.

C'est le moment de doter les classes de derniers traits (attributs ou méthodes). Ce diagramme sera utilisé pour le développement et donc ne pourra plus être révisé (ou alors de manière coûteuse). Les autres diagrammes de conception peuvent aussi être améliorés en conséquence.

Compléter les diagrammes de conception passe aussi par l'attribution de machine à état pour les classes ayant des comportements dynamiques. Le but est de mettre en valeur les différents changements d'état d'une entité quand elle interagit avec les autres.

Pour une classe stéréotypée «cooperative agent» qui possède déjà une (ou plusieurs) machine(s) à états (voir l'étape $A_{16}S_3$), les nouveaux états identifiés doivent apparaître dans une nouvelle machine à états. Cette dernière sera concurrente de la (des) première(s) (voir chapitre 4).

3.7 Les travaux suivants

ADELFE, bien que s'intégrant dans le RUP, ne propose aucune modification des activités suivant la conception. En effet, nous avons fait le choix d'intégrer les problématiques agents uniquement dans les phases préliminaires de développement. Cependant, proposer des aides au développement ou à l'implantation fait partie de nos perspectives de recherche et de développement d'ADELFE.

3.7.1 Développement et Implémentation

Ces enchaînements d'activités restent en grande partie identiques à ceux rencontrés dans le RUP. Les principaux objectifs sont d'associer des classes et objets à des composants (fichier source, codes binaires et exécutables), de tester chaque unité produite et d'intégrer les exécu-

tables dans le système. La différence majeure réside dans l'utilisation du prototypage rapide en amont, lors de la conception, pour faciliter le codage et procéder aux tests unitaires.

Les principaux intervenants de cet enchaînement d'activités sont :
- le développeur qui implante/code et effectue les tests unitaires grâce au scenarii prévus en amont, aidé dans sa tâche par l'outil de prototypage rapide intégré à OpenTool ;
- l'intégrateur système qui assemble les composants pour obtenir le système ;
- l'architecte qui construit le modèle du système ;
- l'auditeur de code qui vérifie la qualité du code.

Les principaux artefacts produits lors de l'implémentation sont :
- Les sous-systèmes d'implémentation ;
- Les composants (codes sources et exécutables) ;
- Le plan d'intégration, définissant la procédure d'assemblage du système.

Ces travaux pourront être mis en place grâce à des règles de production de code, à partir des spécification des modèles de conception, en fournissant, par exemple, un schéma de conception complet des agents coopératifs ou bien en se reposant sur le paradigme MDA (*Model Driven Architecture*[4]) de l'OMG utilisant des interfaces ouvertes et standards.

3.7.2 Tests et validation

L'objectif principal de ces enchaînements d'activités est d'évaluer la qualité du système produit. Cette vérification suit les indications du RUP et est effectuée en vérifiant les interactions, en vérifiant la bonne mise en place de l'intégration, en vérifiant la bonne prise en compte des exigences et en identifiant les anomalies. Chaque niveau de l'architecture possède un jeu de tests adéquats : tests unitaires, tests d'intégration, tests du système et tests d'acceptation par l'utilisateur final. ADELFE complète le RUP par son utilisation de l'outil de prototypage rapide pour la réalisation de jeux de tests unitaires et d'interactions entre les agents.

Les principaux intervenants de cet enchaînement d'activités sont :
- le concepteur de tests ;
- le testeur du système ;
- le testeur de performance ;
- le testeur d'intégration.

Les principaux artefacts produits lors de l'implantation sont :
- Le plan de test ;
- Le modèle de test classique, avec les cas de tests, les procédures, les scripts de tests, ainsi que les prototypes issus de l'exécution du prototypage rapide (essentiellement pour les tests unitaires et inter-agents) ;
- Les modèles de charge de travail pour la performance ;
- Les anomalies identifiées ;
- Les paquetages, classes et composants de tests.

[4] `http://www.omg.org/mda/`

3.8 Bilan et remarques sur le processus d'ADELFE

Ce chapitre a présenté le processus d'ADELFE, des besoins jusqu'à la conception. Ce processus s'intègre dans le processus orienté objet RUP qui est un processus connu des ingénieurs et industriels. En cela nous avons répondu à une des exigences du projet ADELFE, à savoir de s'intégrer dans un cadre de développement connu.

Les particularités que nous apportons sont les suivantes :

L'**analyse de l'environnement** lors des besoins finals doit permettre de pointer les problèmes dus à une particularité de l'environnement du système, comme la dynamique ou l'indéterminisme. Cette caractérisation sert de commencement au traçage des interactions potentiellement non coopératives pouvant mener à des situations non coopératives et donc à des dysfonctionnements du système par rapport à son environnement.

L'**analyse de l'adéquation des AMAS** permet à l'analyste de savoir si l'utilisation des systèmes multi-agents adaptatifs est nécessaire pour le problème posé en fonction de critères de macro-niveau ou de micro-niveau. Si le système peut utiliser les AMAS, cela n'implique en rien que le système à construire soit un système multi-agent, du moins au premier niveau de décomposition. En effet, le système peut être composé d'objets qui ne sont pas des agents coopératifs mais qui en sont composés. Une telle décomposition nécessite alors une nouvelle analyse de ces composantes, c.-à-d. nécessite de leur appliquer ADELFE, afin de les modéliser comme des AMAS.

L'**identification des agents** doit permettre à l'analyste de clairement identifier des classes du domaine qui seront modélisées comme des agents coopératifs. Cette identification est le résultat de l'analyse des interactions (des acteurs aux classes concernées) et des flots de données et de contrôle dits "à risque", et de l'analyse des propriétés des classes, comme l'autonomie de contrôle ou l'activité.

L'**étude des langages d'interaction** grâce à des protocoles AIP, permet de déterminer les jeux de méthodes et d'attributs nécessaires aux communications. Ces protocoles étant génériques, il sera possible de les attribuer à n'importe quelle classe d'agent.

La **conception des agents** et de leur attitude coopérative par l'instanciation du modèle d'agent coopératif présenté dans le paragraphe 3.6.1. Ceci revient à remplir les différents modules de l'agent grâce à des méthodes et des attributs spécifiquement stéréotypés. Le comportement coopératif des agents est présenté sous forme de règles de déclenchement de comportements coopératifs ayant des conditions de détection et des actions associées devant assurer le retour à un état coopératif.

Le **prototypage rapide** des comportements sociaux des agents communicants peut se faire grâce à des outils de pré-codage ou de simulation.

Cependant, le processus d'ADELFE n'est pas encore complet. Les phases postérieures à la conception sont pour l'instant identiques à celles du RUP. La problématique agent apparaît principalement pendant l'analyse et la conception. De plus, au sein du processus certains travaux importants manquent encore de maturité, comme l'identification des agents ou la conception des agents. En effet, le manque de formalisme et de formalisation de la théorie des AMAS est un frein pour l'écriture de règles de conception. Néanmoins, de nom-

breuses directives et aides ont été définies grâce à l'expérience des précédents développeurs d'AMAS.

Tout au long de la description du processus, nous avons pu voir que l'utilisation d'UML dans ADELFE était préconisée. En effet, les agents sont des classes. Les différents modules des agents coopératifs sont obtenus par stéréotypage de méthodes et d'attributs. De plus, les interactions entre agents sont modélisées par des diagrammes de protocoles A-UML.

Dans le chapitre suivant, nous allons présenter les spécificités qu'ADELFE apporte aux notations UML et A-UML pour prendre en compte les agents coopératifs.

4 Les notations et outils utilisés dans ADELFE

LES notations sont une dimension importante dans la définition de méthodes. Compte tenu des remarques et conclusions faites lors de l'analyse des méthodes existantes, nous avons fait le choix de la notation UML. Ce choix est principalement motivé par la large diffusion de ce langage de modélisation, permettant ainsi une présentation plus aisée aux développeurs novices en systèmes multi-agents.Cependant nous sommes conscients des limites d'UML, tant au niveau de l'expressivité que de la validation. En effet, les concepts UML ne sont pas suffisants pour exprimer les propriétés des systèmes multi-agents. De plus, UML propose des solutions de vérification syntaxique mais pas sémantique. Nous allons donc proposer des extensions à UML, en nous appuyant, notamment sur A-UML, qui propose déjà des notations spécifiques pour les interactions entre agents.

Les extensions que nous proposons portent principalement sur deux aspects de la conception des agents :

La *vue statique* des agents, par la définition de stéréotypes d'attributs et d'opérations, permettant un enrichissement sémantique et l'intégration des modules des agents coopératifs ;

La *vue dynamique*, ou comportement, des agents, par l'utilisation et l'extension de A-UML, ainsi que la transformation de protocoles en machines à états, afin de réaliser des vérifications et de simuler le comportement social des agents communicants.

D'autre part, l'utilisation de notations graphiques requiert, pour plus de facilité d'utilisation, de développer des outils intégrant ces notations. Pour cela, les notations propres à ADELFE ont été intégrées à OpenTool[1]. De plus, une des originalités d'ADELFE est de fournir un outil d'aide au suivi du processus, appelé AdelfeToolkit.

Le but de ce chapitre sera donc de montrer comment modifier et adapter les notations existantes, pour prendre en compte les spécificités des systèmes multi-agents adaptatifs, du point de vue statique (§4.1) et du point de vue dynamique (§4.2). Enfin, les outils de support d'ADELFE sont présentés dans le paragraphe 4.3.

[1] http ://www.tni-valiosys.com

4.1 Les stéréotypes d'ADELFE

La première extension apportée à UML par ADELFE est le stéréotypage d'opérations et d'attributs en fonction de leur appartenance aux modules définis pour le modèle d'agent coopératif fourni par ADELFE (voir §3.6.1).

4.1.1 UML est-il adéquat pour ADELFE ?

Comme l'ont souligné Van Dyke Parunak et Odell (voir §2.1), l'agent peut être vu comme une évolution de l'objet et possède de nombreux points communs avec ce dernier. Par conséquent, dans ADELFE, le choix a été fait d'utiliser les notations dédiées à l'objet. La plus connue, et la plus utilisée, est bien entendu UML. De plus, compte tenu du processus proposé par ADELFE, le paradigme objet et les notations affiliées semblent nécessaires.

Cependant, l'objet, en tant que tel, n'est pas suffisant pour exprimer les concepts propres aux agents. Dans ADELFE, les agents coopératifs sont composés de modules spécifiques, suivent un cycle de vie "perception-décision-action" et peuvent participer à des négociations complexes dans lesquelles peuvent survenir des situations non coopératives. Ces aspects peuvent être traités par extension du métamodèle UML. Concernant les interactions, de nombreux travaux ont abouti à des spécifications plus ou moins précises d'A-UML. Cependant, elles ne sont pas suffisantes pour traiter la coopération (voir §4.2).

Du point de vue statique (classes, opérations et attributs), nous proposons donc d'utiliser UML et de l'étendre afin de traiter les agents comme des classes ayant une sémantique particulière.

4.1.2 Comment la notation peut-elle être étendue ?

Deux solutions sont envisageables pour s'étendre à ces spécificités : étendre les métamodèles ou utiliser un profil UML [Desfray, 2000]. Une extension du métamodèle est généralement utilisée lorsque les concepts d'un domaine particulier sont bien définis et établis. Toutefois, dans le domaine multi-agent, le concept d'agent n'est pas uniforme ; des architectures différentes existent : agent coopératifs pour ADELFE ou les BDI pour Gaia par exemple [Wooldridge *et al.*, 2000]. Plusieurs points de vue peuvent aussi exister. Par exemple, ADELFE se base sur l'auto-organisation par coopération alors que d'autres méthodes utilisent des organisations connues et établies à l'avance, comme dans Tropos [Castro *et al.*, 2001]. Il est alors plus difficile d'étendre le métamodèle, et par conséquent de dicter une architecture agent ou multi-agent. C'est pour cela que la seconde solution a été choisie pour plus de flexibilité. De plus, cette solution du stéréotypage a été souvent soulevée par la communauté A-UML [Odell *et al.*, 2000].

Ainsi, dans ADELFE, neuf stéréotypes ont été définis pour exprimer comment un agent est formé et/ou comment son comportement peut être exprimé. Dans la volonté de modifier la sémantique des classes et traits (opérations et attributs), des règles de bon usage, rédigées en langage d'action UML ou en OTScript (voir §4.3.1) par exemple, doivent être définies et attachées à chaque stéréotype afin d'assurer que les modèles sont correctement écrits.

4.1.3 Description des stéréotypes d'ADELFE

Une description de chaque stéréotype ajouté est dressée dans ce paragraphe en donnant sa définition, les règles de cohérence associées et un exemple d'utilisation. Les exemples sont issus d'ETTO (voir §3.2.2) ou d'autres applications. Bien qu'aucune directive de développement et programmation ne soit exprimée dans ADELFE, quelques exemples de codes sources, squelettes de classes, sont proposés pour présenter des solutions possibles de modèles.

Tous ces stéréotypes s'appliquent à des traits (Feature du métamodèle UML), c.-à-d. à des attributs ou des opérations, excepté le stéréotype «cooperative agent» qui s'applique à des classes.

4.1.3.1 Le stéréotype «cooperative agent»

Signification. Le stéréotype «cooperative agent» exprime qu'une classe est un agent qui a une attitude coopérative participant à la construction d'un AMAS. Un agent sera donc modélisé comme une classe stéréotypée. Cette classe devra posséder une opération run simulant le cycle de vie de l'agent. Par conséquent, pour assurer que cette opération existe, un agent doit hériter d'une superclasse abstraite appelée CooperativeAgent, fournie par ADELFE. Cette classe possède les opérations abstraites perceive, decide et act et une opération run.

Exemple 4.1. *Codage en Java de la classe CooperativeAgent :*

```
public abstract class CooperativeAgent {
  public void run() {
    perceive();
    decide();
    act();
  }
  protected abstract void perceive();
  protected abstract void decide();
  protected abstract void act();
}
```

Ceci implique que les agents seront animés par l'opération run, que ce soit par un ordonnanceur préétablis (dans le cas de *threads*, par exemple) ou développé pour l'occasion. Ainsi, à chaque pas d'exécution, l'ordonnanceur appellera l'opération run de chaque agent dans un ordre pré-établi ou aléatoire (dans le cas de *threads* par exemple). Cependant, c'est au développeur de remplir les opérations perceive, decide et act de la classe d'agent, héritant de CooperativeAgent, qu'il code.

Exemple 4.2. *Codage en Java d'une classe héritant de CooperativeAgent :*

```
public class VerySimpleAgent extends CooperativeAgent {
  private int perception;
  private int action;
  private Goal goal;
  ...
  protected void perceive() {
```

```
    perception = getUpdatedInputValue();
    ...
  }
  protected void decide() {
    ...
    action = getNextAction(interpretInput(goal,perception));
    ...
  }
  protected void act() {
    doAction(action);
    ...
  }
  ...
}
```

Règles. Une classe stéréotypée «cooperative agent» doit hériter directement ou indirectement de la classe CooperativeAgent. De plus, afin de modéliser un agent coopératif "viable", une classe «cooperative agent» devra nécessairement posséder au moins des opérations ou attributs stéréotypés de chacun de stéréotypes suivants :

- «perception», car un agent doit forcément percevoir son environnement ;
- «action», car un agent doit agir s'il ne veut pas être inutile et donc non coopératif ;
- «aptitude», car un agent est une entité autonome dans ses décisions et raisonnements ;
- «cooperation», car un agent coopératif doit forcément avoir une attitude coopérative.

Exemple 4.3. *Dans ETTO, les classes Teacher et StudentsGroup ont été identifiées comme étant des agents, et donc stéréotypées «cooperative agent». Cependant, elles héritent de la classe RepresentativeAgent, qui hérite elle-même de CooperativeAgent car stéréotypée «cooperative agent». Par conséquent, les classes Teacher et StudentsGroup n'ont pas besoin d'hériter de CooperativeAgent, mais possèdent cependant les quatre opérations perceive, decide, act et run.*

4.1.3.2 Le stéréotype «characteristic»

Signification. Le stéréotype «characteristic» est utilisé pour pointer les propriétés intrinsèques ou physiques d'un agent coopératif. Un attribut ainsi stéréotypé représente la valeur de cette propriété. Une opération «characteristic» modifie ou met à jour ces propriétés. Une caractéristique peut être lue ou modifiée à n'importe quel moment du cycle de vie de l'agent. Elle peut aussi être lue ou appelée par les autres agents.

Règles. Aucune.

Exemple 4.4. *Dans une application de robotique, où les robots sont modélisés en tant qu'agents, les attributs caractéristiques peuvent être le nombre de roues ou de pattes, leur couleur, leur poids, leur position dans l'espace, la vitesse maximum de déplacement, etc. Les opérations «characteristic» permettront de les modifier ou de les mettre à jour, par exemple.*

4.1.3.3 Le stéréotype «perception»

Signification. Le stéréotype «perception» exprime un moyen que l'agent à de recevoir de l'information venant de son environnement physique ou social. Les attributs représentent les données provenant de l'environnement. Les opérations sont des moyens de mettre à jour ou de modifier les attributs stéréotypés «perception».

Exemple 4.5. *Dans l'exemple 4.2, l'opération getUpdatedInputValue() est stéréotypée «perception».*

Règles. Un attribut stéréotypé «perception» est nécessairement privé ou protégé.

Exemple 4.6. *Dans ETTO, un BookingAgent peut obtenir des informations sur la salle qu'il occupe grâce à la gestion de l'attribut «perception» roomInfo. Ce dernier stocke les informations à propos des autres agents situés dans la même salle grâce à l'utilisation de l'attribut «perception» vectoriel nearAgents.*

4.1.3.4 Le stéréotype «action»

Signification. Le stéréotype «action» est utilisé pour signaler un moyen d'agir sur l'environnement durant la phase d'action. Les opérations sont les actions possibles pour un agent, alors que les attributs sont des paramètres de ces actions. Un agent est le seul à pouvoir activer ses propres actions afin d'assurer d'autonomie de contrôle.

Règles. Une opération stéréotypée «action» est nécessairement privée et ne peut être appelée que lors de la phase d'action. Ceci signifie que le code des opérations stéréotypées «action» ne peut apparaître que dans des opérations appelées directement, ou indirectement par l'opération act. Cette vérification nécessite d'analyser le code ou pseudo-code fourni lors de la conception (ce qui reste possible avec des outils adéquats, cf. §4.3.1) et de continuer à s'assurer du respect de ces règles lors du développement (ce qui est beaucoup moins évident).

Exemple 4.7. *Dans ETTO, un BookingAgent peut agir pour réserver une salle (opération bookRoom) qui lui semble intéressante compte tenu de ses contraintes, pour libérer une salle si elle n'est plus intéressante (opération freeRoom), ou pour envoyer des messages aux autres agents (sendMessage), etc.*

4.1.3.5 Le stéréotype «skill»

Signification. Le stéréotype «skill» est utilisé pour étiqueter les compétences, c.-à-d. des connaissances spécifiques à un domaine, permettant à l'agent de réaliser sa fonction partielle. Les opérations représentent les règles de raisonnement que peuvent avoir les agents. Les attributs sont des données (ou faits) sur le monde ou les paramètres des opérations stéréotypées «skill». De tels attributs ou opérations ne sont accessibles qu'à l'agent les possédant pour exprimer son autonomie de décision. Les compétences peuvent être représentées par un système multi-agent adaptatif si les compétences ont besoin de changer au cours du

le temps, comme dans ABROSE [Gleizes *et al.*, 2000]. Une telle décomposition doit être détectée lors de l'analyse du système. Le système multi-agent résultant sera alors un attribut stéréotypé «skill» de la classe d'agent décomposée.

Règles. Un attribut ou opération stéréotypé «skill» est nécessairement privé. De tels attributs peuvent uniquement être utilisés dans des opérations stéréotypées «skill». De plus, une telle opération peut être appelée uniquement lors de la phase de décision de l'agent (seule l'opération decide peut la voir apparaître dans son code de manière directe ou indirecte).

Exemple 4.8. *Dans ETTO, un BookingAgent sait comment interpréter les message reçus (opération interpretMessage) ou évaluer si une proposition reçue à propos d'une réservation est intéressante (opération isProposalAcceptable).*

Exemple 4.9. *Dans l'exemple 4.2, l'opération interpretInput() qui interprète les messages en entrée, en fonction du but (attribut goal) de l'agent, est stéréotypée «skill».*

4.1.3.6 Le stéréotype «aptitude»

Signification. Le stéréotype «aptitude» exprime la capacité d'un agent à raisonner sur ses perceptions, ses connaissances et ses compétences. Les opérations expriment le raisonnement qu'un agent est capable de faire, par exemple de l'inférence sur ses compétences (qui peuvent alors être des règles et des faits). Les attributs représentent les données de fonctionnement ou les paramètres du raisonnement. Une opération ou attribut stéréotypé «aptitude» peut seulement être accessible à l'agent lui-même, pour exprimer son autonomie de décision.

Règles. Une opération ou attribut stéréotypé «aptitude» est nécessairement privée ou protégée. Un attribut «aptitude» peut seulement être utilisé par une opération «aptitude». Une opération «aptitude» peut uniquement être appelée lors de la phase décision du cycle de vie de l'agent (opération decide). Une opération «aptitude» peut uniquement faire appel à des attributs ou opérations non stéréotypés ou stéréotypés «characteristic», «perception», «skill», «representation» ou «interaction». Cette dernière règle met en place le fonctionnement interne de l'agent, comme exprimé dans le paragraphe 3.6.1.

Exemple 4.10. *Un BookingAgent possède certaines contraintes concernant les créneaux horaires qu'il peut réserver dans l'emploi du temps. Il doit être capable de gérer ses contraintes et de les modifier ou les mettre à jour. Il le fait par la biais de la opération manageConstraints qui est stéréotypée «aptitude».*

Exemple 4.11. *Dans l'exemple 4.2, l'opération getNextAction(), qui détermine la prochaine action à exécuter, est stéréotypée «aptitude».*

4.1.3.7 Le stéréotype «representation»

Signification. Le stéréotype «representation» est un moyen d'indiquer les représentations du monde que possède l'agent. Les attributs «representation» sont des unités de connais-

sances. Les opérations «representation» sont des moyens de les manipuler : accès, modification, etc. Les représentations peuvent évoluer, et, de ce fait, être modélisées par un AMAS (il en est de même pour les compétences).

Règles. Une opération (ou attribut) «representation» est nécessairement privée ou protégée (la connaissance d'un agent est locale et personnelle, à moins qu'il ne la communique via des messages par exemple). Un attribut «representation» peut seulement être utilisé par des opérations stéréotypées «representation» ou «aptitude». Enfin, une opération «representation» peut seulement être appelée lors de la phase de décision (opération decide).

Exemple 4.12. *Dans ETTO, s'il a établi un partenariat, un BookingAgent connaît ses partenaires par l'attribut ownPartner. Un attribut appelé lastEncounteredAgents lui permet de mémoriser les derniers agents rencontrés sur la grille.*

4.1.3.8 Le stéréotype «interaction»

Signification. Le stéréotype «interaction» étiquette les outils qui permettent à l'agent de communiquer directement ou indirectement avec son environnement. Les opérations «interaction» expriment la capacité d'un agent à interagir avec les autres. Les attributs «interaction» représentent les données de fonctionnement ou les paramètres des interactions. Les interactions peuvent être classés en deux groupes : les perceptions et les actions qui devront être étiquetées respectivement «perception» et «action».

Règles. Une opération stéréotypée «interaction» peut seulement appeler des opérations stéréotypées «skill», «representation» ou «interaction».

Exemple 4.13. *Dans ETTO, les BookingAgents communiquent directement par échange de messages. Différentes opérations sont stéréotypées «interaction» car elles gèrent des messages : inform (pour informer un autre agent), acceptPartnership (pour signifier à un autre agent l'acceptation d'un partenariat).*

4.1.3.9 Le stéréotype «cooperation»

Signification. Le stéréotype «cooperation» exprime l'attitude sociale coopérative de l'agent en implantant les règles de résolution des situations non coopératives. L'agent doit avoir un ensemble de règles (ou prédicats) permettant de détecter les SNC. Ces règles sont écrites en utilisant les perceptions, les représentations et les compétences. L'agent doit aussi posséder des opérations de résolution associées aux règles de détection. Une opération «cooperation» pourra être, par exemple :

- une opération renvoyant un booléen indiquant la détection d'une SNC et possédant des paramètres provenant de perceptions, de représentations et/ou de compétences ;
- une opération de résolution associant une ou plusieurs actions possibles à chaque SNC. Le choix, dans le cas d'actions multiples, devra aussi être géré par une opération «cooperation» ou «aptitude».

Ces opérations «cooperation» doivent subsumer les opérations «aptitude», c.-à-d. que les actions choisies par les opérations «cooperation» prennent le pas sur les actions choisies en cas de fonctionnement nominal, sans SNC. Ceci peut être obtenu de plusieurs manières : instructions conditionnées ou usage d'exceptions pour simuler la priorité des résolutions de situations non coopératives.

Exemple 4.14. *Codage en Java de l'opération decide d'un agent ayant un comportement coopératif obtenu par l'utilisation d'exceptions :*

```
protected void decide() {
  try {
    ...
    action = getNextAction(interpretInput(goal, perception));
    ...
  } catch (NCS1 ncs1) {
    action = ACTION_TO_SOLVE_NCS1;
  } catch (NCS2 ncs2) {
    action = ACTION_TO_SOLVE_NCS2;
  } catch (NCS2 ncs3) {
    action = ACTION_TO_SOLVE_NCS3;
  }
  ...
}
```

Règles. Une opération ou un attribut «cooperation» est nécessairement privé ou protégé.

Exemple 4.15. *Dans ETTO, une SNC possible pour un BookingAgent est l'incompétence de partenariat. La condition de détection est "Le BA courant est en présence d'un autre BA **ET** (cet autre BA n'est pas du bon type **OU** le coût du partenariat avec lui serait supérieur ou égal à celui du BA courant)" (voir exemple 3.32). Une opération partnershipIncompetence renvoie la valeur booléenne vraie si toutes les conditions sont remplies. Ces conditions peuvent être des perceptions ("Le BA courant est en présence d'un autre BA") ou des représentations ("le coût du partenariat avec lui serait supérieur ou égal à celui du BA courant").*

Les actions à entreprendre en cas de détection sont "Le BA courant mémorise l'agent rencontré et se déplace afin de rencontrer de nouveaux BA". L'opération stéréotypée «action» correspondante est freeRoom.

4.2 Extensions d'A-UML dans ADELFE

Dans ADELFE, le comportement dynamique des agents est modélisé grâce à des machines à états finis ou des protocoles d'interaction dans le cas d'agents communicants. Nous avons dû étendre les notations A-UML afin de prendre en compte la coopération des agents lors de communications. De plus, comme ADELFE propose une activité de prototypage rapide (voir §3.6.5), nous avons choisi d'utiliser une modélisation par machine à états pour l'exécution. Ainsi, simuler le comportement d'un agent revient à exécuter sa machine à états afin de déterminer si le comportement est adéquat.

Nous disposons alors de deux formalismes de modélisation : les protocoles et les machines à états finis. Nous proposons un algorithme de transformation de protocoles en machine à états afin d'unifier les notations et de faciliter la simulation dans OpenTool (voir §4.3.1).

4.2.1 Les protocoles A-UML dans ADELFE

Le diagramme de protocole développé pour ADELFE est une évolution du diagramme de séquences *générique* d'UML1.4. Il permet de faire intervenir des classes et des rôles de classe d'agent, et de les faire communiquer entre eux. Comme c'est un diagramme générique, il peut être attribué à n'importe quel objet.

Il faut noter que la notion de rôle est uniquement relative à un protocole, et donc est très locale. Ce concept est légèrement différent du concept de rôle tel que l'on a pu en parler dans le paragraphe 2.4, qui représente un rôle dans une organisation générale et non point à point. Dans notre cas – celui des AMAS –, un agent ne joue pas un rôle dans le système, mais jour un rôle pour une interaction ponctuelle.

Outre les lignes de vie avec les rectangles d'activation et les envois de message, il est possible de saisir des embranchements (ou jonctions) AND (barre verticale), OR (losange), et XOR (losange avec une croix), comme il est spécifié dans les propositions A-UML, et de fournir certaines connaissances supplémentaires sur les envois de messages, comme le traitement coopératif à la réception d'un message, par exemple. Il est également possible d'y spécifier des clauses IF...ELSE...ENDIF, permettant plus d'expressivité lors de la conception des protocoles. L'introduction de ces clauses est la conséquence directe de l'analyse des spécifications d'ABROSE, dans lesquelles les protocoles de communication en faisait souvent usage de simple notes sur les diagrammes pour exprimer des alternatives de haut niveau ayant une sémantique particulière [Gleizes *et al.*, 2000].

Un exemple de diagramme de protocole pour ADELFE est présenté en figure 4.1. Ce protocole est celui présenté par Odell *et al.* dans les premières spécifications d'A-UML [Odell *et al.*, 2000]. Cependant, quelques différences existent ; notamment, les flèches en pointillés signalent des interactions potentiellement non coopératives auxquelles sont associés des opérations de traitement «cooperation».

4.2.1.1 Contraintes de spécification des protocoles

Les diagrammes de protocoles étant destinés à une transformation en machine à états pour tester le comportement par simulation, ces diagrammes doivent obéir à certaines contraintes :

- le temps graphique n'a de sens qu'au sein d'une activation, ceci en raison de l'introduction des jonctions AND, OR et XOR qui perturbent la disposition graphique ;
- le diagramme de protocoles doit commencer par un message initial ;
- un déroulement du protocole sous forme d'arbre doit être possible à partir du message initial ; toutefois, certaines exceptions sont possibles.

Les rectangles d'activation du protocole fournissent une trame de continuité au sein d'un enchaînement possible d'envois de messages : le message qui suit un message est recher-

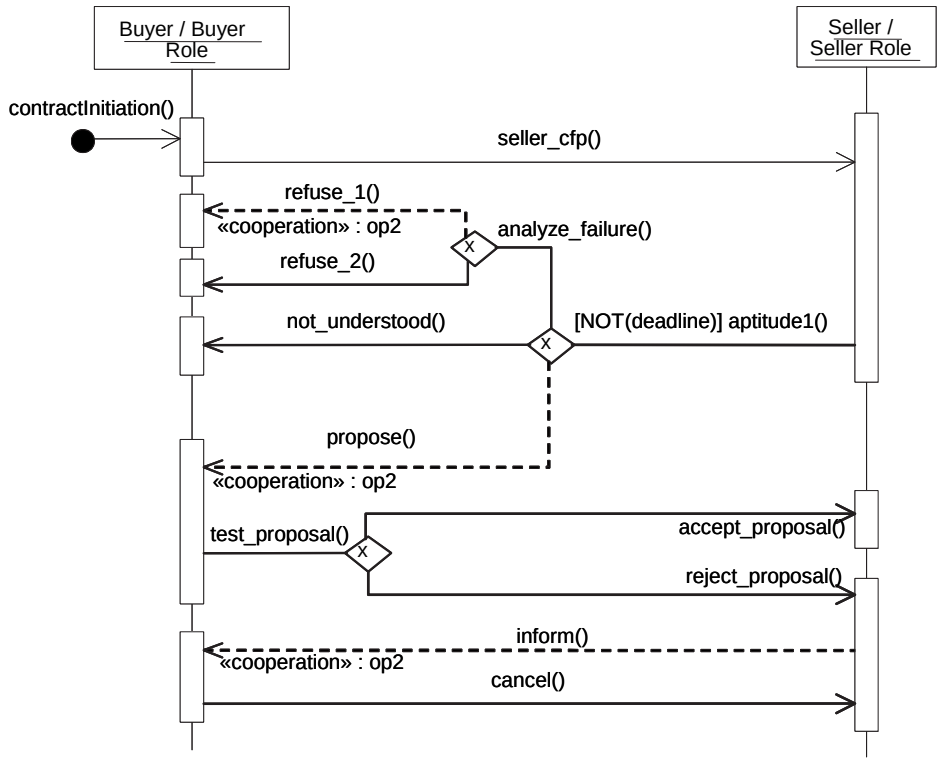

Figure 4.1 — Un exemple de protocole dans ADELFE.

ché dans l'activation réceptrice de ce message (message graphiquement immédiatement en-dessous du départ de cette activation). Les embranchements sont dus à l'introduction des jonctions AND, OR ou XOR, et de clauses IF. Les cas suivants, qui présentent une rupture de continuité, sont cependant compris :

- envoi de plusieurs messages successifs à partir d'une classe ou rôle d'agent ;
- réception de plusieurs messages successifs sur une classe ou rôle d'agent ;
- rupture de continuité, via les rectangles d'activation, suivie d'une clause IF ;
- rupture de continuité, via les rectangles d'activation, suivie de la clause ELSE d'un IF ;
- rupture de continuité, via les rectangles d'activation, suivie de la clause ENDIF d'un IF.

4.2.1.2 Utilisation des diagrammes de protocole

Lignes de vie. Lors de la création d'une ligne de vie, il faut lui associer une classe, et un rôle d'agent s'il s'agit d'un agent qui en possède plusieurs. Il est important de créer au moins un rôle d'agent par classe agent susceptible d'en avoir plusieurs, pour pouvoir éventuellement en introduire d'autres par la suite, et d'utiliser le rôle d'agent pour créer une ligne de vie, et non l'agent lui-même.

Messages. Les envois de messages ne peuvent être créés que sur des rectangles d'activation et des jonctions AND, OR ou XOR. Il est possible de créer sur une ligne de vie autant de rectangles d'activation que l'on veut. Sur création d'un envoi de message, ne doivent être

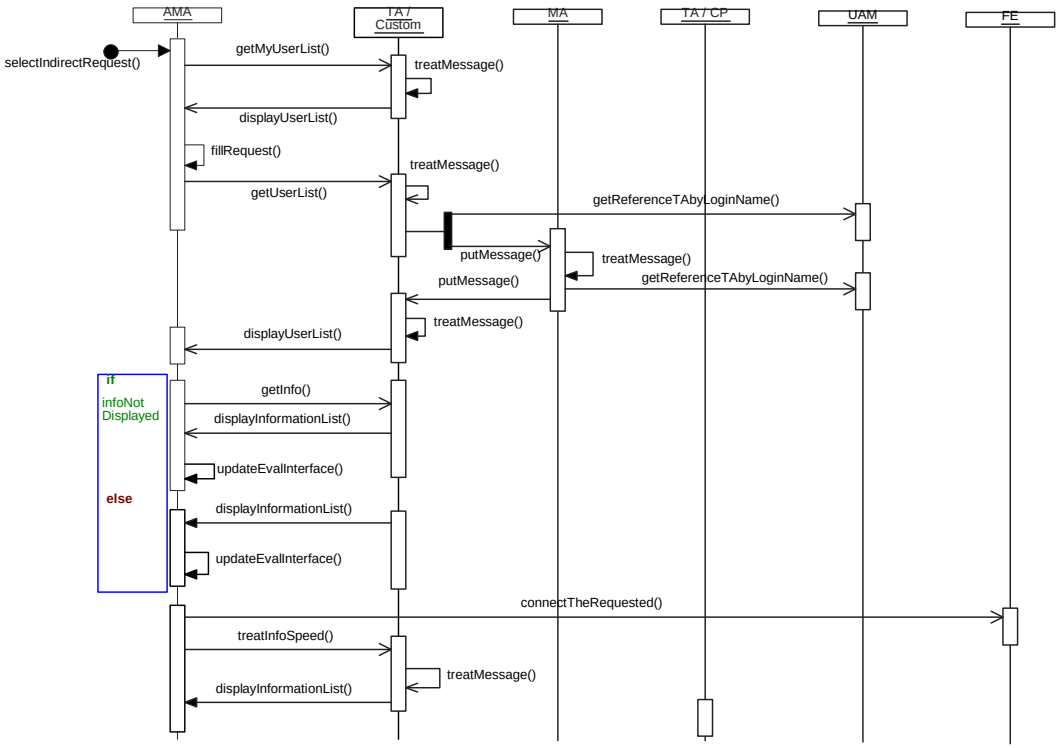

Figure 4.2 — Un exemple de protocole de requête indirecte avec clause IF (rectanggle de couleur à gauche de la ligne de vie de AMA.

proposées que les opérations publiques stéréotypées «interaction». Il est possible de préciser une opération stéréotypée «cooperation» à la réception d'un envoi de message : ceci permet de renvoyer à d'autres comportements traités dans d'autres protocoles, par exemple.

Jonction. Une jonction AND, OR ou XOR doit être rattachée à une classe ou à un rôle d'agent. Cependant, il faut faire attention aux règles qui sont différentes selon le type de jonction. Le message interne qui va d'un agent à un losange " OR " ou " XOR " doit porter une opération agent stéréotypée «aptitude» : l'indéterminisme des jonctions A-UML représente l'autonomie de décision des agents.

Clauses IF...ENDIF ou IF...ELSE...ENDIF. Ces clauses sont liées, à leur création, à une classe ou un rôle d'agent, et doivent porter une condition qui est une opération booléenne de cette classe. La figure 4.2 présente un exemple de clause IF bien formée. Le positionnement graphique d'une clause IF est de première importance :

- Une clause IF n'a d'effet que s'il y a rupture préalable d'enchaînement au travers des activations qui la précèdent (fin apparente via la trame de continuité des activations) ;
- Le point de départ de la clause correspond graphiquement au haut de son rectangle ;
- Il doit y avoir à nouveau rupture de la continuité des activations avant le ELSE éventuel, puis redémarrage après le ELSE éventuel d'une nouvelle chaîne d'activations, et de même, rupture avant la fin du rectangle de la clause IF qui correspond au ENDIF ;
- Il est conseillé de placer, pour plus de lisibilité, la clause IF à proximité de la classe ou du rôle d'agent auquel elle est rattachée.

4.2.1.3 Diagrammes de séquence d'instances enrichis

Le diagramme de séquence d'*instances* d'UML1.4 a également été enrichi de manière à prendre en compte le concept de rôle d'agent, ainsi que les jonctions AND, OR, et XOR. Ce type de diagramme peut être utilisé à titre documentaire, mais il ne pourra être ni utilisé comme protocole associable à une classe, ni transformé en machine à états.

Le concept de rôle d'agent a été introduit de la manière suivante :
- un rôle d'agent est relatif à une et une seule classe agent ;
- une classe agent peut avoir plusieurs rôles d'agent ;
- il est toujours possible de représenter des instances sans rôle d'agent ;
- la ligne de vie peut être simple ou porter des rectangles d'activation ;
- dans la ligne de vie dédiée à une instance caractérisée par un rôle d'agent, le titre est construit avec : <nom instance>/<nom rôle agent> : <nom classe>.

4.2.2 Transformation en machine à états finis

Comme il a été précisé plus haut, afin d'unifier les notations et de faciliter la simulation, nous proposons de transformer les protocoles d'interaction, tels que spécifiés ci-dessus, en machines à états finis. Ce paragraphe expose la méthode à suivre et est illustré par la transformation du protocole de la figure 4.1.

4.2.2.1 Construction de machines à états finis à régions concurrentes

Tout d'abord, une machine à état initiale est attribuée à chaque classe «cooperative agent». Cette machine est composée d'un état initial et d'un état composite Alive. Dans ce dernier, un région concurrente est ajoutée pour chaque rôle tenu par l'agent, comme le montrent les figures 4.3 et 4.4. Ici, un seul rôle provient d'un protocole (région du haut). Bien sûr, d'autres régions concurrentes peuvent être ajoutées pour d'autres comportements dynamiques, par exemple la région du bas dans les figures présentées. Ensuite, chaque région concurrente est développée comme une machine à états. Ceci permet à l'agent de pouvoir répondre et participer à plusieurs protocoles à la fois.

Chacune de ces sous-machines est composée de deux états principaux : un état Idle, par défaut, et un état, ayant, par convention, le même nom que l'événement initiateur du protocole pour le rôle avec la première lettre en majuscule. Dans le protocole, le rôle Buyer Role est activé à la réception de l'événement contractInitiation. Par conséquent, la région concurrente correspondant à ce rôle contient un état ContractInitiation. De même, le rôle Seller Role est activé à la réception de l'événement seller_cfp. Par conséquent, la région concurrente correspondant à ce rôle contient un état Seller_cfp. Ces deux états principaux sont reliés par des transitions bidirectionnelles : ainsi, un agent peut exécuter plusieurs fois un protocole. L'agent sort de l'état Idle lorsqu'il reçoit l'événement de début de protocole (transition Idle vers "état de rôle"). Pour sortir du protocole (transition "état de rôle" vers Idle), il y a deux possibilités : soit le protocole est arrivé à son terme pour ce rôle, soit une exception s'est produite suite à une SNC, par exemple.

En considérant la sémantique donnée aux lignes de vie des protocoles, un sous-état est

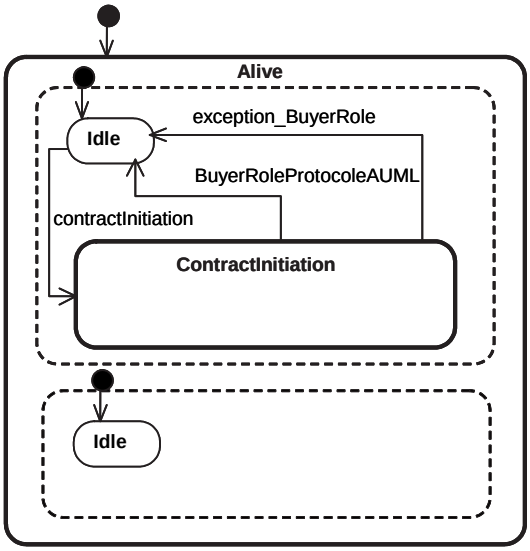

Figure 4.3 — Machine à états finis pour la classe Buyer du protocole de la figure 4.1.

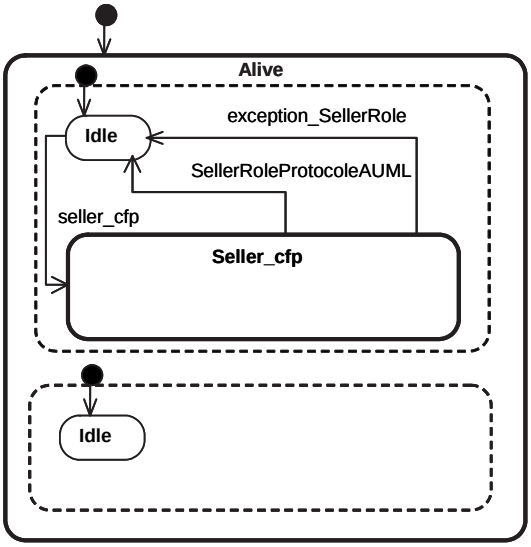

Figure 4.4 — Machine à états finis pour la classe Seller du protocole de la figure 4.1.

ajouté à l'état composite du rôle pour chaque réception ou émission de message, comme le montre la figure 4.5. Par exemple, un sous-état Seller_cfp_sent est associé à l'envoi du message seller_cfp. De même, un état Refuse_1 est associé à la réception du message refuse_1.

Les transitions entre états sont créées à la réception ou à l'émission d'un message. Par exemple, quand l'agent Buyer se trouve dans l'état Seller_cfp_sent et reçoit le message refuse_1, une transition est créée entre les états Seller_cfp_sent et Refuse_1. Si un état ne reçoit ou n'envoie plus aucun message, il est relié à un état final avec l'action de retour à l'état Idle.

4.2.2.2 Jonctions A-UML et clauses IF

Concernant les jonctions A-UML, une opération stéréotypée «aptitude» est associée au processus de décision du branchement, et, par conséquent, une action d'entrée (entry) est ajoutée dans l'état correspondant à l'envoi du message. Cette action exécute l'aptitude qui renvoie la signature de l'opération à activer et le résultat est mémorisé dans une variable conditionResult. Dans le cas de jonctions XOR ou OR à *n* branches, il faut créer *n* transitions. Dans la figure 4.5, les conditions correspondant à la jonction XOR sont [conditionResult = "seller.acceptProposal"] et [conditionResult = "seller.refuseProposal"]. Ces conditions sont attachées aux transitions vers les états Accept_proposal_sent et Reject_proposal_sent.

Pour les branchements AND, un état appelé AND est créé, et la transition vers cet état est créée avec une action composite exécutant les opérations à effectuer en séquence. Les contraintes, comme des échéances (*deadlines*), sont exprimées par des opérations renvoyant des booléens et traitées par des transitions conditionnelles : si la condition est vraie, l'automate continue (ainsi que le protocole), sinon il revient à l'état Idle, comme le montre la figure 4.6.

La distinction des rôles d'agent d'un agent est faite dans la structure concurrente des

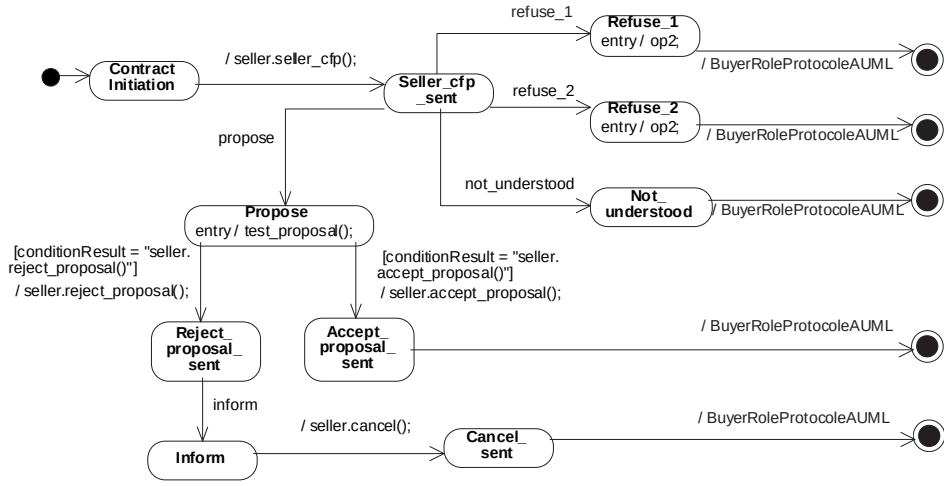

Figure 4.5 — Sous-machine de l'état ContractInitiation de la classe Buyer pour le rôle Buyer Role.

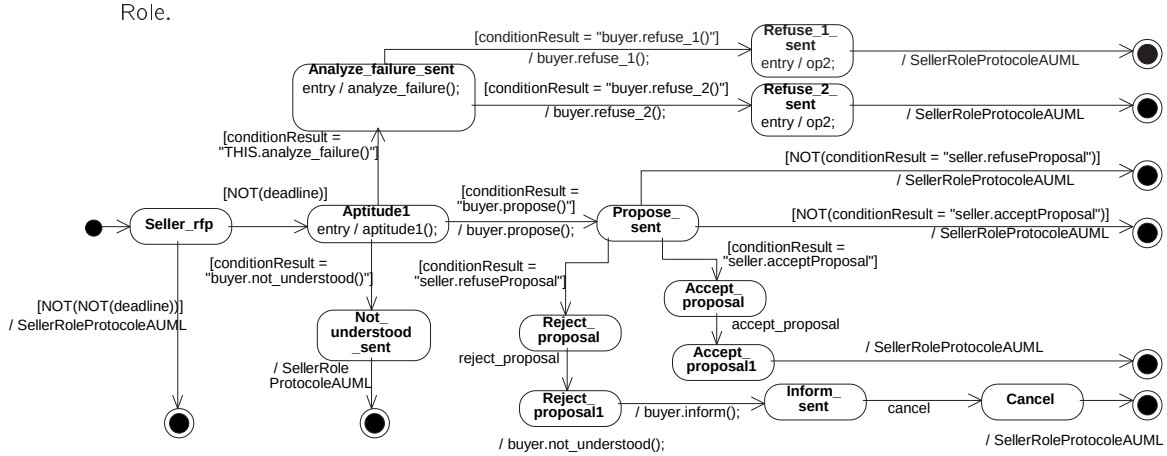

Figure 4.6 — Sous-machine de l'état Seller_cfp de la classe Seller pour le rôle Seller Role.

machines à états ; ainsi, cette distinction n'est pas à faire au moment de l'envoi : l'événement est envoyé à un agent, dont une ou plusieurs de ses régions concurrentes sont capables de traiter l'événement, à ce moment-là.

La transition équivalente à une jonction AND porte, en action à exécuter, l'envoi de tous les événements des branches du AND. Par contre, le même message envoyé à différents rôles d'une instance ne fait l'objet que d'un seul envoi. Le nom de l'état d'arrivée est constitué de AND, auquel s'ajoute éventuellement un numéro pour obtenir un identifiant unique. Il convient de rechercher ensuite les chemins de continuité (ceux qui permettent d'arriver à une réception d'événement pour le rôle d'agent courant) pour chacun des événements envoyés afin de poursuivre l'arbre de la machine à états.

Avec la jonction AND, tous les événements sont envoyés ; avec la jonction XOR, un seul est envoyé. Nous étions alors dans des cas simples. Dans le cas des jonctions OR se pose le problème de la combinatoire de n événements. La proposition actuelle laisse l'opération «aptitude» effectuer dans son code les envois de messages selon son analyse, et la machine à états ne présume pas des chemins de continuité qui peuvent en résulter ; par contre, l'exception est chargée de lever un éventuel blocage dans un état (cas où aucune des transitions

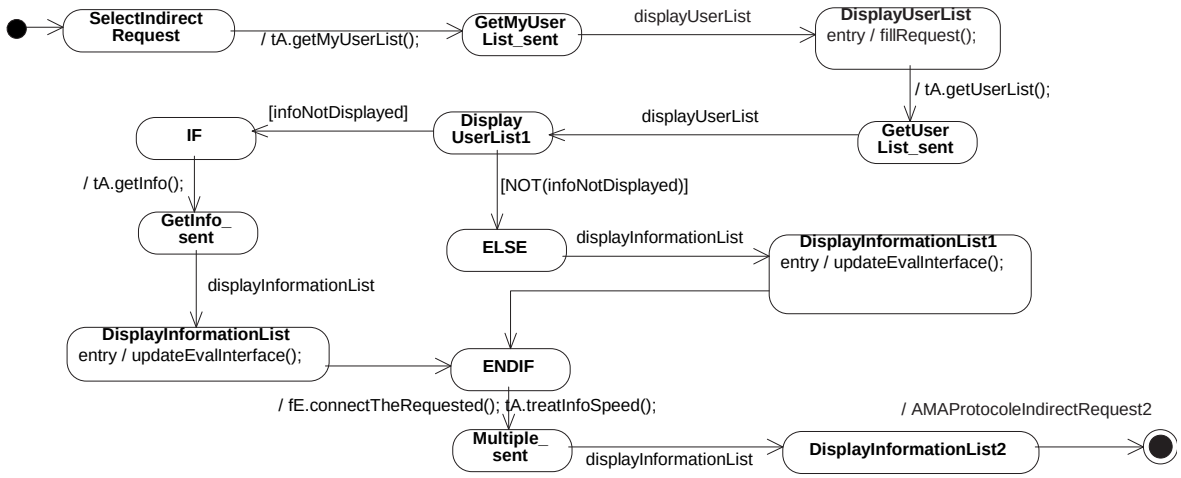

Figure 4.7 — Sous-machine de l'état SelectIndirectRequest de la classe AMA du protocole de la figure 4.2.

proposées en sortie n'est franchissable). Un message réflexif, lui, est considéré comme un traitement interne et vient compléter le code de la clause *entry* de l'état courant.

Lors de la génération de la machine à états, il y a prise en compte de la branche courante du protocole si des jonctions XOR sont rencontrées. Un problème peut survenir si l'on revient sur des rectangles d'activation distincts d'une même classe (ou rôle d'agent) via un même événement, sans qu'il y ait moyen de distinguer dans la machine à états d'où l'on vient ; cependant cette ambiguïté peut être détectée en traçant les messages et en les identifiant.

Les protocoles affichant des clauses IF sont transformables en machines à états finis, comme le montre la figure 4.7. A partir de l'état arrivant sur le IF, deux états, IF et ELSE sont créés avec des transitions dont la garde correspond à la condition de la clause. Les deux chemins résultants se rejoignent ensuite sur un état ENDIF.

4.2.2.3 Remarques

Voici un bref récapitulatif des règles de bonne formation d'un protocole A-UML dans ADELFE :

- ▷ un protocole doit débuter par un message initial ;
- ▷ les messages provenant d'un XOR, OR ou AND et arrivant sur une même classe doivent être sur des activations distinctes ;
- ▷ nécessité d'une connaissance de la manière atteindre le récepteur d'un message : existence d'une association unique entre les classes communicantes ;
- ▷ un message peut s'accrocher à des activations ou des jonctions (pas directement sur la ligne de vie) ;
- ▷ le temps graphique n'a de sens qu'au sein d'une activation ;
- ▷ une opération de stéréotype «aptitude» précédant une jonction XOR ou OR est placée dans la clause *entry* de l'état courant ;
- ▷ un message réflexif doit voir sa flèche arriver sur sa propre activation, et donc avant

tout autre message envoyé à suivre ;

- le déroulement d'un protocole doit avoir, *a priori*, une structure d'arbre ; cependant les cas suivants sont traités :
  - envoi de *n* messages consécutifs depuis une même activation (traité comme un AND) ;
  - réception de *n* messages consécutifs sur une même activation (traité comme la nécessité de recevoir ces messages les uns après les autres) : l'ordre de réception est pris en compte, ce qui signifie que la machine à états ne fonctionne plus si l'ordre n'est pas respecté ;
  - rupture de continuité puis clause IF, ELSE ou ENDIF ;
- les choix conditionnels peuvent s'exprimer par un XOR ou par une condition ou par une clause IF...ENDIF ou IF...ELSE...ENDIF :
  - le XOR convient lorsque le choix est suivi de différents messages issus de ce même émetteur ;
  - une condition sur un message permet au protocole de continuer si la condition est vraie, mais l'arrête si elle est fausse ;
  - une clause IF...ENDIF ou IF...ELSE...ENDIF est rattachée à une activation : elle est testée à la position du sommet de sa boîte graphique, et permet de réaliser une ou deux dérivations ; elle n'est prise en compte que sur rupture de la chaîne des activations :
    - il faut donc préalablement au IF qu'il y ait rupture de continuité des messages via les activations ;
    - le message suivant est recherché sur l'activation liée au IF selon sa position en ordonnées (côté cible) ;
    - il y a lecture des messages pour cette nouvelle chaîne d'activations qui doit se terminer avant le ELSE éventuel ou le ENDIF (bas du rectangle du IF) ;
    - s'il y a un ELSE, une chaîne de messages, via des activations démarrant après le ELSE et finissant avant le ENDIF, doit être prise en compte ;
    - pas d'imbrication possible de protocoles avec ces méthodes.

4.3 Les outils associés à ADELFE

Nous avons vu, dans le paragraphe 2.5, que les outils de support des notations étaient l'une des composantes principales d'une méthode. Par exemple, le RUP est officiellement pris en charge par Rational Rose. Il en est de même pour ADELFE qui possède aussi ses outils propres : OpenTool pour prendre en charge les notations et AdelfeToolkit pour prendre en charge le processus.

4.3.1 Prise en charge des notations avec OpenTool

OpenTool, de TNI-Valiosys, est un outil de conception UML du style de Rational Rose. Il a le net avantage sur ce dernier d'être facilement modifiable en fonction des besoins des utilisateurs. Une version d'OpenTool a pu ainsi être modifiée pour répondre aux spécifications d'ADELFE : *OpenTool UML1.4 / Adelfe 1.2*. OpenTool possède son propre langage d'action,

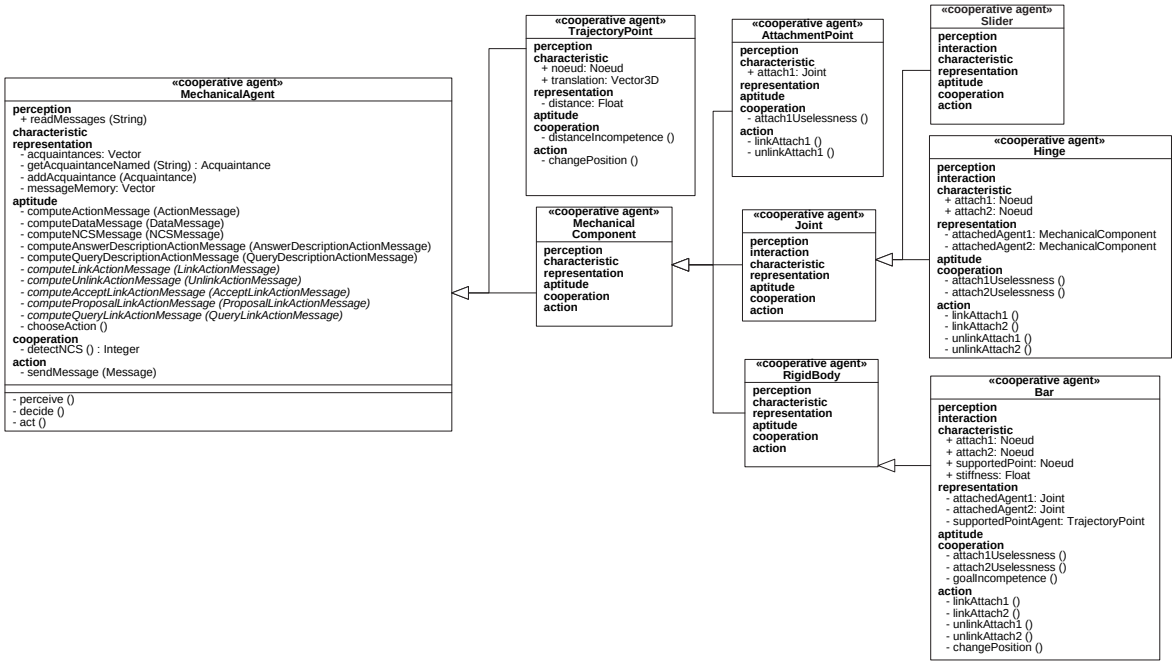

Figure 4.8 — Diagramme de classes d'agents dans OpenTool UML1.4 / Adelfe 1.2.

OTScript. Ce dernier, qui est un langage d'exploration du métamodèle, permet d'associer du code à des opérations afin de tester leur comportement dans des *scenarii* proposés par l'utilisateur.

4.3.1.1 Stéréotypage ADELFE et diagrammes de classes étendus

La première extension faite à OpenTool concerne l'intégration des stéréotypes spécifiques à ADELFE. Bien sûr, de nombreux outils de conception UML permettent aux utilisateurs d'ajouter des stéréotypes en fonction de leurs besoins. Cependant, OpenTool nous a permis de facilement intégrer les stéréotypes (notations «...»), mais aussi leurs règles de bonne utilisation (voir paragraphe §4.1.3) et des modifications dans les diagrammes de classes afin de faciliter la lecture des classes d'agent «cooperative agent», comme le montre la figure 4.8 tirée des spécification de MSS, un système de conception de mécanismes par auto-organisation coopérative [Capera *et al.*, 2004a].

Un compartiment a été ajouté aux compartiments de la représentation graphique de classe. Dans ce compartiment apparaissent des rubriques identifiées en police grasse. Chaque rubrique correspond à un module de l'agent représenté par la classe stéréotypée «cooperative agent».

La vérification de la bonne utilisation des stéréotypes ADELFE est faite en cours de conception, à chaque ajout/suppression/modification d'attribut, d'opération ou de classe dans OpenTool. Ceci permet d'afficher des messages d'erreurs ciblés comme le montre la figure 4.9, dans laquelle la classe Class1 est marquée en rouge, car elle est génératrice de deux erreurs :

1. elle ne possède aucun attribut stéréotypé «perception», «action» ou «cooperation», alors

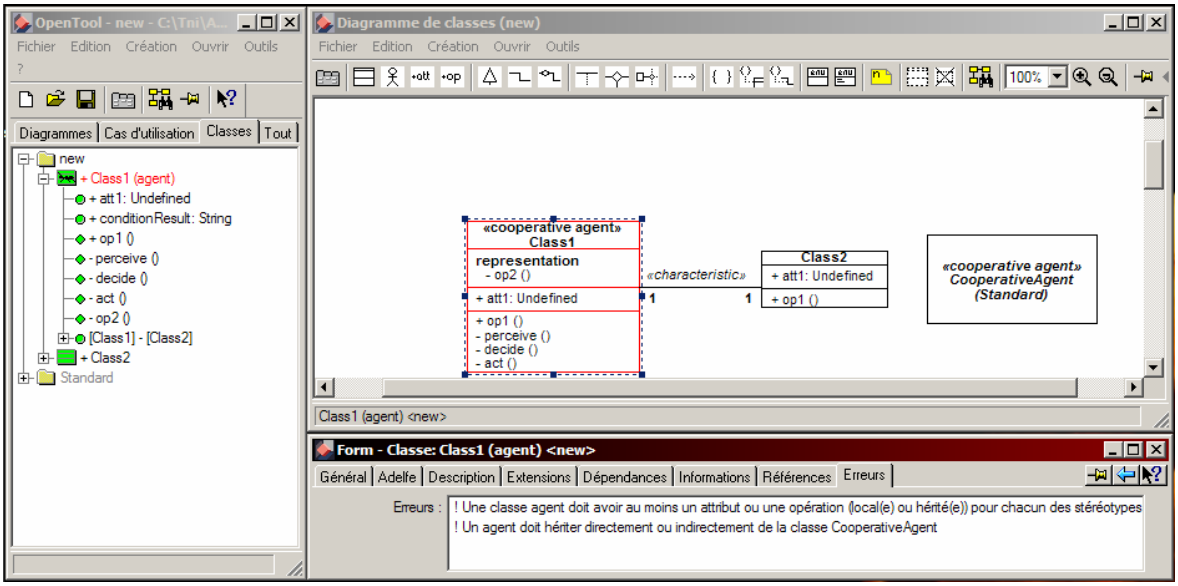

Figure 4.9 — Gestion des erreurs d'utilisation des stéréotypes ADELFE dans OpenTool.

qu'elle est stéréotypée «cooperative agent» ;

2. elle n'hérite ni directement, ni indirectement de la classe CooperativeAgent, alors qu'elle est stéréotypée «cooperative agent».

4.3.1.2 Diagrammes de protocole A-UML et transformation automatique en machines à états finis

Comme ADELFE utilise une notation de type A-UML pour spécifier les protocoles de communication entre agents, OpenTool a été modifié pour prendre en charge cette notation. Ces diagrammes sont des extensions de diagrammes de séquence génériques UML1.4. Un éditeur spécialisé permet de saisir toutes les composantes utilisées dans ADELFE : jonctions, clauses IF ou coopération à la réception de messages. Tous les diagrammes de protocole présentés dans ce mémoire ont d'ailleurs été édités via OpenTool. Ici, encore, une vérification à la volée de l'utilisation des spécificités d'ADELFE est faite.

Il est possible d'attribuer à des classes d'agent des protocoles génériques, comme le suggère l'activité A_{16} d'ADELFE (voir §3.6.4). Nous avons aussi intégré l'algorithme de transformation de protocoles en machines à états finis, présenté dans le paragraphe 4.2.2, afin de simuler le comportement des agents dans OpenTool.

4.3.1.3 Simulation pour prototypage rapide

Une fonctionnalité avancée d'OpenTool permet de simuler les machines à états finis hiérarchiques afin de valider des comportements. Les simulations sont initialisées à partir de diagrammes de collaboration dans lesquels apparaissent les instances de classes à simuler avec les associations nécessaires et les valeurs attribuées en fonction des besoins. Le langage d'action d'OpenTool permet d'implanter du pseudo-code afin de réellement faire fonctionner les opérations.

Une fois la simulation initialisée, il suffit de la lancer et de suivre le déroulement de la simulation, c.-à-d. le changement d'état en état pour chacune des instances en présence. Dans les diagrammes d'états, les états actifs sont signalés, ainsi que les transitions activées, par des codes de couleurs. Il est ensuite possible, une fois la simulation achevée, de consulter les enchaînements de messages et d'appels d'opérations dans des diagrammes de séquences générés en cours de simulation. Ceci permet de déterminer la validité des protocoles simulés.

4.3.1.4 Limites actuelles

OpenTool est un outil de conception orienté objet classique, auquel nous avons ajouté les stéréotypes ADELFE et les protocoles A-UML. Cependant, la fonctionnalité de simulation ne permet de réellement simuler et analyser que des protocoles assez simples compte tenu de la nécessité de définir des instances. Etant donné que les problématiques de l'auto-organisation et des systèmes émergents font appel à de nombreuses entités, les interfaces offertes par OpenTool, et les possibilités d'analyse en découlant sont trop limitées.

4.3.2 Prise en charge du processus avec AdelfeToolkit

Outre les notations dans OpenTool pour ADELFE, le suivi du processus de développement est lui aussi pris en charge grâce à l'outil AdelfeToolkit. L'idée générale est d'aider le concepteur à suivre le processus d'ADELFE, par des descriptions, des exemples, et de lui présenter une synthèse des travaux et artefacts déjà effectués et restant à faire.

4.3.2.1 Utilisation du métamodèle SPEM

Comme point de départ, nous avons utilisé la spécification SPEM que nous avons établie pour ADELFE (voir §3.1). La description est assez fine pour intégrer dans un outil des objets, instances de classes du SPEM, qui représentent les activités d'ADELFE. Chaque activité possède des conditions d'achèvement, une description, un exemple et des documents à produire associés.

La figure 4.10 présente la fenêtre principale d'AdelfeToolkit. La sous-fenêtre en haut à gauche liste les activités (Activity) et leurs étapes (Step). Les activités accessibles (en fonction de l'avancement) ou achevées sont signalées par des codes de couleurs. Il est possible de consulter l'état de chaque activité. La sous-fenêtre en bas à gauche liste les produits (WorkProduct), les modèles (Model) et les documents (Documents) associés à l'activité en cours (ou sélectionnée). L'état des différents produits est consultable dans des *tooltips*. Les différents états possibles pour chacun des produits apparaissent dans les diagrammes d'activités des figures 3.2, 3.3, 3.11 et 3.15.

Chaque activité ou étape est décrite dans la fenêtre en haut à droite. De plus, la fenêtre en bas à droite permet de visualiser un exemple d'application, comme nous l'avons fait dans le chapitre 3. De plus, une autre fenêtre de synthèse permet de connaître l'état de production des modèles et documents.

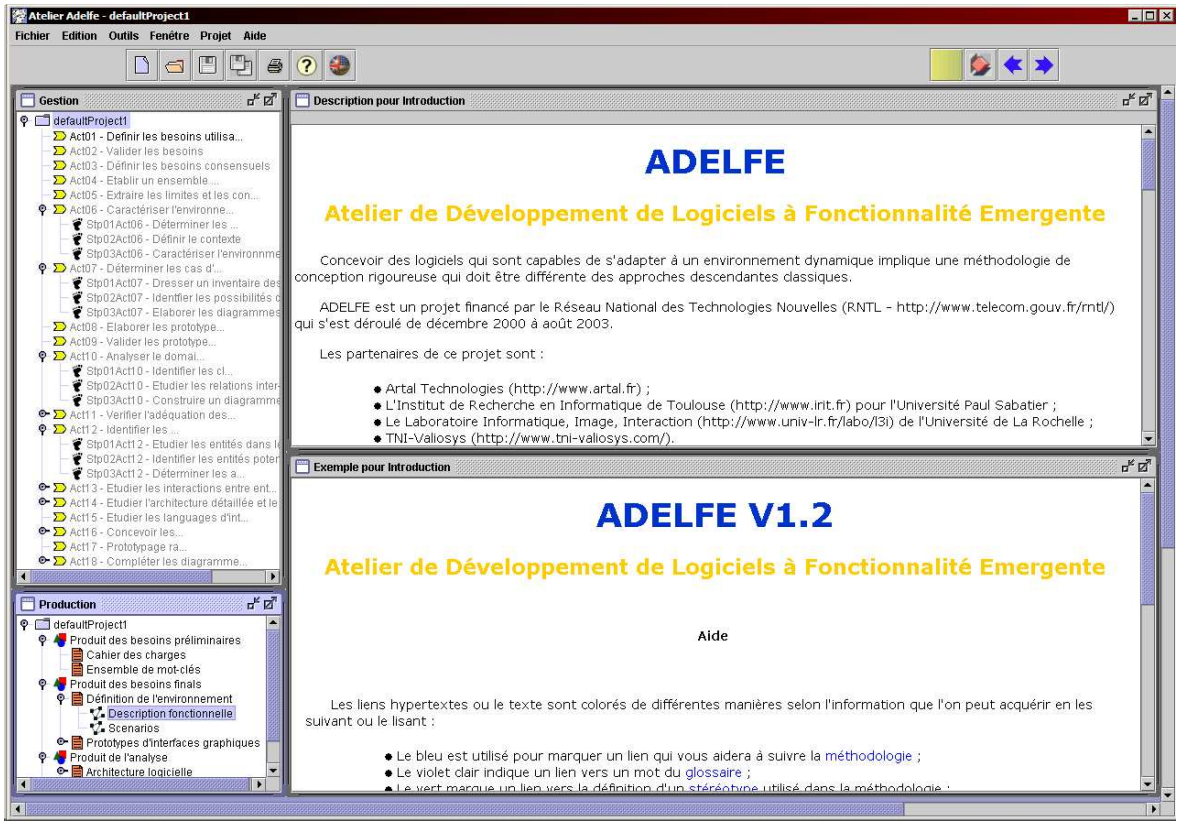

Figure 4.10 — L'outil d'aide au suivi du processus d'ADELFE : AdelfeToolkit.

4.3.2.2 Connexions avec d'autres outils

La validation d'une activité ou d'une étape, afin de passer à la suivante, est obtenue en vérifiant l'ouverture de certains outils, ou la production de certains modèles. Pour cela, AdelfeToolkit peut se connecter à OpenTool afin de vérifier que des classes stéréotypées «cooperative agent» ont bien été produites, par exemple. Certains documents sont validés uniquement si on lance l'application idoine (par exemple, un traitement de textes pour écrire le cahier des charges).

4.3.2.3 Outil d'adéquation des AMAS

Comme précisé dans l'activité A_{11} du processus d'ADELFE, l'analyste doit vérifier si le problème qu'il tente de résoudre a besoin ou non des AMAS. Pour cela, un outil est fourni afin de répondre à la question au niveau macro (huit critères) et au niveau micro (trois critère). L'importance de chaque critère dans la décision a été établie en fonction de l'expérience des précédents développeurs d'AMAS. Les résultats peuvent être plus ou moins précis, en fonction de l'acuité des réponses de l'utilisateur. Les réponses aux questions sont graduées entre 0 et 20 (et non pas *"oui ou non"*).

Chaque résultat (macro et micro) est composite : valeur du résultat (entre 0 et 20) et précision du résultat (empattement de part et d'autre de la valeur du résultat). La valeur est obtenue par somme pondérée des différentes réponses aux questions. La précision dépend

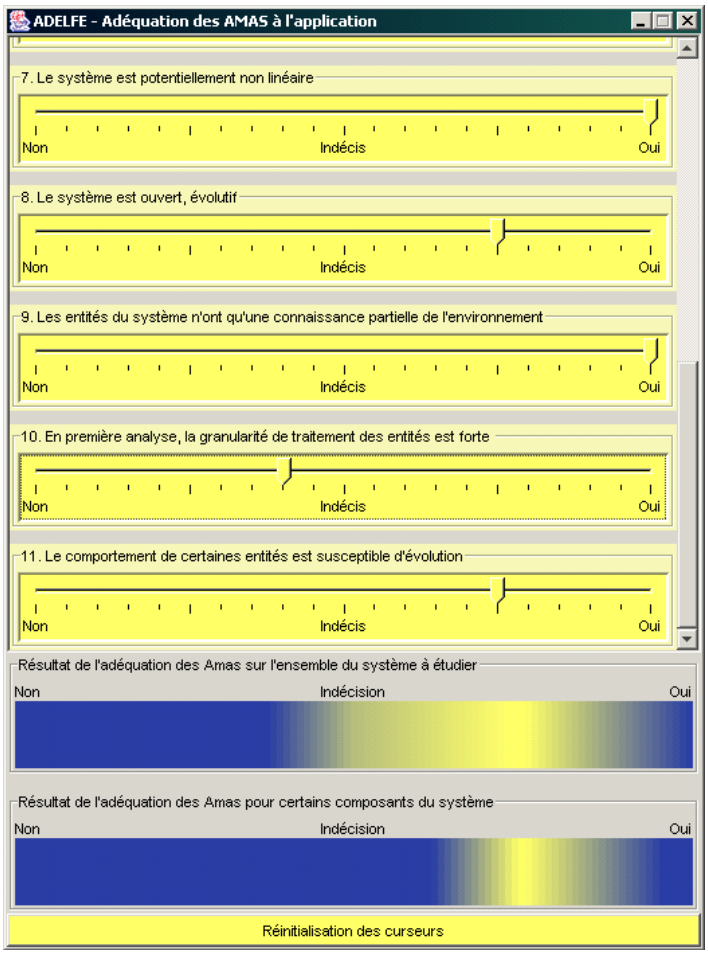

Figure 4.11 — L'outil d'adéquation aux AMAS d'AdelfeToolkit.

de l'écart entre les réponses et la valeur du résultat.

4.3.2.4 Limites actuelles

L'idée d'un outil d'aide au suivi du processus semble intéressante. Cependant, il reste pour l'instant à l'état de prototype car il soulève de grandes difficultés :

- la validation des activités passe souvent par la production de modèles ou de documents. Actuellement, le prototype d'AdelfeTookit vérifie uniquement la production, mais non la validité ou la pertinence des productions. En effet, ce n'est pas parce qu'un concepteur va ajouter un classe «cooperative agent», que son approche est correcte. Il n'y a pas de vérification sémantique ;
- la rigidité du processus ADELFE, comme implanté dans AdelfeToolkit, peut empêcher le bon développement des applications. Un processus doit être vivant.

Malgré ces remarques, AdelfeToolkit, qui se veut être un livre interactif, est l'un des seuls outils de ce genre, mis à part le *Rational Process Workbench*[2] qui permet de modifier le RUP.

[2]http ://www.ibm.com/developerworks/rational/

4.4 Bilan et remarques sur les notations et les outils d'ADELFE

Dans le paragraphe 2.5, nous avions exprimé notre volonté d'utiliser la notation UML comme support de modélisation dans ADELFE. En effet, UML est connu des ingénieurs, et donc plus facilement assimilable. De plus, s'intégrer dans la mouvance A-UML semblait être un avantage, compte tenu du nombre des personnes y travaillant.

Nous avons répondu à ces besoins d'une part par l'ajout de stéréotypes UML pour modéliser les modules des agents coopératifs, et d'autre part par la modification les protocoles A-UML afin de les simuler et de prendre en compte la coopération à la réception de messages. Nous avons même proposer un algorithme de transformation de protocoles en machines à états afin d'unifier les modèles de description de comportements d'agents.

Toutefois, ceci n'est qu'une étape préliminaire vers un réel travail de métamodélisation. En effet, l'utilisation des protocoles d'interaction n'est pas suffisant pour pouvoir illustrer la richesse des interactions et leur complexité, lorsque l'on parle de système auto-organisateurs. De plus, modéliser des systèmes ouverts, dynamiques et composés d'un grand nombre d'entités n'est pas chose facile avec A-UML. Pour répondre à ces problème, une extension radicale du métamodèle UML et A-UML semble donc nécessaire.

De plus, UML est connu pour sa facilité d'assimilation, mais aussi pour son manque de formalisme, ce qui le limite en terme de vérification. Cependant, il est toujours possible, comme nous l'avons fait, d'attacher des règles de bonne utilisation à vérifier en cours de modélisation. Dans des travaux précédents et préliminaires, une tentative de modélisation d'AMAS en B a montré cependant la difficulté de formaliser de tels systèmes, lorsque la fonction du système est inconnue des parties et qu'aucune connaissance globale n'est possible. En effet, la méthode B est mal adaptée à la preuve de convergence, qui est bien sûr le but d'une formalisation de systèmes auto-organisateurs. Le B événementiel permettrait, quant à lui, de montrer des propriétés de vivacité, c.-à-d. de convergence en temps fini, mais non de montrer des propriétés de convergence à la limite [Vareilles, 2002].

Troisième partie

Expérimentations et Analyses

5 Expérimentation d'ETTO

ETTO est l'application didactique d'ADELFE, grâce à laquelle nous avons illustré les activités de son processus. Un prototype pour ETTO a été développé afin de valider l'approche des systèmes multi-agents adaptatifs pour ce genre de problème. Ayant déjà exposé toutes les étapes nécessaires à la conception de ce système, ce chapitre est voué uniquement à exposer les résultats de diverses expérimentations avec le prototype obtenu et à préciser le comportement des agents.

Les expérimentations exposées sont basées sur le cahier des charges du problème de l'emploi du temps[1] que nous avons proposé au groupe de travail ASA[2] (Approches par Sociétés d'Agents) du Groupe GDR-I3 Systèmes Multi-Agents et de l'AFIA. Le cahier des charges se décompose en quatre variantes ajoutant chacune plus de complexité au problème que la précédente : de la résolution d'un problème simple sans relâchement des contraintes jusqu'à l'ouverture du système par l'ajout et la destruction d'agents. Pour chacune d'entre elles, nous fournissons une solution possible, qui n'est pas forcément unique.

[1] http://www-poleia.lip6.fr/~guessoum/asa/BenchEmploi.pdf
[2] http://www-poleia.lip6.fr/~guessoum/asa.html

5.1 Les variantes du problème

Le cahier des charges propose quatre variantes de cas à traiter, de complexité croissante. On suppose que pour chacun des acteurs, les contraintes sont données sous la forme d'une liste. Chacune d'elles est affectée d'un poids fixé par l'utilisateur indiquant l'importance relative qu'il accorde à la contrainte.

5.1.1 Variante 1 : résolution possible sans relâcher les contraintes

Dans un premier temps, on simplifie en supposant que l'on manipule des créneaux horaires de 2h : 8h-10h, 10h-12h, 14h-16h, 16h-18h et que l'on doit trouver un emploi du temps sur deux jours : j_1, j_2.

Trois enseignants e_1, e_2 et e_3 enseignent chacun une matière spécifique et leurs impossibilités d'enseignement sont les suivantes :

- e_1 ne peut enseigner le jour j_1 de 16h à 18h et le jour j_2 de 14h à 16h ;
- e_2 ne peut enseigner le jour j_2 de 10h à 12h et le jour j_1 de 16h à 18h ;
- e_3 ne peut enseigner le jour j_1 de 14h à 16h et le jour j_2 de 8h à 10h ;

On considère trois groupes d'étudiants g_1, g_2 et g_3. Chacun d'eux doit suivre, sur ces deux jours, deux enseignements de 2h assurés par chacun des enseignants e_1, e_2 et e_3 (soit, pour chaque groupe, 12h d'enseignement au total).

Pour l'instant, on suppose que le système n'a pas à gérer la disponibilité des salles : on dispose d'une salle par groupe. On suppose aussi que les acteurs ne peuvent pas relâcher de contraintes. Dans ce cas, une solution possible pourrait être la suivante :

	Jour j_1		
8h-10h	$e_1 + g_1$	$e_3 + g_2$	$e_2 + g_3$
10h-12h	$e_3 + g_1$	$e_2 + g_2$	$e_1 + g_3$
14h-16h	$e_2 + g_1$	$e_1 + g_2$	
16h-18h			$e_3 + g_3$

Jour j_2		
$e_1 + g_1$	$e_2 + g_2$	
	$e_1 + g_2$	$e_3 + g_3$
$e_2 + g_1$	$e_3 + g_2$	$e_1 + g_3$
$e_3 + g_1$		$e_2 + g_3$

Tableau 5.1 — Une solution à la variante 1 d'ETTO.

5.1.2 Variante 2 : résolution possible en relâchant certaines contraintes

On ajoute des contraintes sur les salles. On suppose que trois salles s_1, s_2 et s_3 sont disponibles. Seules les salles s_1 et s_2 sont munies d'un rétroprojecteur.

- la salle s_1 n'est pas disponible le jour j_1 de 10h à 12h ;
- la salle s_2 n'est pas disponible le jour j_2 de 16h à 18h et de 8h à 10h ;
- la salle s_3 n'est pas disponible le jour j_2 de 16h à 18h et le jour j_1 de 14h à 16h ;

Tous les enseignants veulent utiliser un rétroprojecteur au moins une fois pour chaque groupe lors des deux jours. On suppose qu'un ou plusieurs acteurs peuvent relâcher des contraintes pour proposer une solution.

Le coût des contraintes sur les enseignants est fixé à 10 et celui des salles à 5.

A condition de relâcher la contrainte "salle s_1 non disponible le jour j_1 de 10h à 12h" pour placer e_1 et g_3 à ce créneau et de relâcher la contrainte "salle s_2 non disponible j_2 de 16h à 18h" pour placer e_3 et g_1 à ce créneau (puisque les coûts sont moins élevés sur les salles), une solution pourrait être :

	Jour j_1		
	Salle s_1	Salle s_2	Salle s_3
8h-10h	$e_2 + g_1$	$e_3 + g_2$	$e_2 + g_3$
10h-12h	**$e_1 + g_3$**	$e_3 + g_1$	$e_2 + g_2$
14h-16h	$e_2 + g_1$	$e_1 + g_2$	
16h-18h	$e_3 + g_3$		

Jour j_2		
Salle s_1	Salle s_2	Salle s_3
$e_2 + g_2$		$e_1 + g_1$
$e_1 + g_2$	$e_3 + g_3$	
$e_3 + g_2$	$e_1 + g_3$	$e_2 + g_1$
$e_2 + g_3$	**$e_3 + g_1$**	

Tableau 5.2 — Une solution à la variante 2 d'ETTO (en gras apparaissent les partenariats avec relâchement de contraintes).

5.1.3 Variante 3 : les contraintes peuvent varier en temps réel

On ajoute la possibilité de modifier/ajouter des contraintes en temps réel (via une interface graphique). Ainsi, un enseignant pourra signaler qu'il est ou non disponible pour un certain créneau horaire, une salle pourra se libérer ou au contraire devenir occupée.

Les contraintes pouvant évoluer dynamiquement, il faut être capable de concevoir un système qui puisse s'adapter aux changements sans ré-initialisation complète du système. Par exemple, au cours de la recherche d'une solution à l'emploi du temps ci-dessus, l'enseignant e1 signale qu'il ne peut plus assurer un cours le jour j1 de 10h à 12h mais qu'à la place, il est disponible le même jour de 16h à 18h.

Comment le système peut-il résoudre le problème sans interrompre la recherche de la solution et sans repartir de zéro ?

Une solution serait de demander à l'enseignant e_3 de faire cours au groupe g_3 le jour j_1 de 14h à 16h pour faire passer e_1 du créneau 10h-12h ce jour là, au créneau ainsi libéré.

5.1.4 Variante 4 : ouverture du système

Une variante 4 est proposée pour tester l'adaptabilité du système au fait qu'il devienne ouvert. De nouveaux acteurs (enseignants, groupes d'étudiants ou salles) peuvent apparaître ou disparaître en temps réel, le système doit être capable de s'adapter à ces nouveaux "imprévus".

5.2 Expérimentations et résultats

Un prototype a été développé pour ETTO afin de valider l'approche multi-agent coopératif pour la résolution d'emploi du temps dynamique. Ce prototype nous a permis de tester

les différentes variantes et d'obtenir quelques résultats.

5.2.1 Le comportement des agents

Dans le paragraphe 3.6.4, les modules des BookingAgents sont décrits en exemple, ainsi que leur comportement coopératif (les règles de résorption des SNC). Pour résumer et clarifier, nous présentons deux comportements sous forme d'algorithmes simples correspondant au comportement général (5.1) et au comportement à l'arrivée dans une case (5.2).

```
début
    tant que je suis vivant faire
        traiter mes messages;
        si j'ai atteint mon but alors
            aller sur la cellule réservée
        sinon
            si je sais qu'une cellule est libre alors
                aller sur la cellule libre
            sinon
                aller sur une cellule au hasard
            fin
            traiter les rencontres;
            si je n'ai pas toutes mes rendez-vous ou partenaires OU que mes réservations ne
            sont pas optimales alors
                traiter la salle
            fin
        fin
    fin
fin
```

Algorithme 5.1 — Comportement général simplifié d'un BookingAgent.

Ce n'est pas parce qu'un agent a réservé une cellule qu'il va y rester. Il continue à explorer la grille tant qu'il n'est pas satisfait, mais laisse des informations dans la cellule réservée pour aider les autres (un *post-it* virtuel). De plus, si un BookingAgent réserve une cellule correspondant à une heure, il va le communiquer à son RepresentativeAgent correspondant. Ce dernier, alors, ajoute une contrainte d'indisponibilité pour les autres BookingAgent afin d'éviter des situations d'ubiquité.

Dans le prototype développé, les BookingAgents s'échangent les noms des partenaires éventuels de manière coopérative afin d'optimiser la recherche. Cependant, l'exploration de la grille et, quant à elle, aléatoire : lorsqu'un agent veut changer de cellule, il en tire une au hasard. Une version future d'ETTO intégrera la gestion coopérative des salles.

5.2.2 Expérimentation de la variante 1

Ce cas est extrêmement simple : ETTO trouve une solution – pas étonnant, le problème est sous-contraint. La figure 5.1 montre que le système converge (coût total des contraintes égal à 0) en 67 pas de temps. Les partenariats sont vite mis en place (7 pas), alors que la

```
début
    si j'arrive sur une cellule alors
        consulter ma mémoire;
        si j'ai déjà vu cette cellule alors
            mettre à jour ma mémoire
        sinon
            mémoriser la nouvelle cellule
        fin
        examiner la cellule;
        si la cellule est intéressante alors
            si la cellule est délà prise alors
                m'ajouter dans la liste des intéressés
            sinon
                réserver la cellule
            fin
        fin
    fin
fin
```

Algorithme 5.2 — Comportement d'un BookingAgent à l'arrivée sur une cellule.

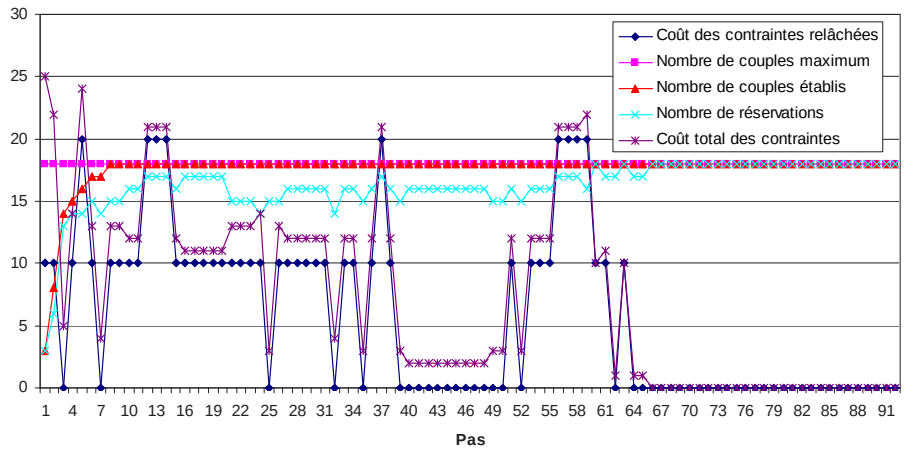

Figure 5.1 — Variation des contraintes en cours de résolution pour la variante 1.

recherche des bonnes cellules prend beaucoup plus de temps (67 pas).

Cependant, une analyse intéressante, si l'on veut vérifier l'efficacité de l'approche d'ETTO, est de faire des tests avec plus d'agents. Ici nous augmentons aussi le nombre de cellules afin de ne pas faire varier la taille de l'espace de recherche. Les seules contraintes utilisées sont celles des disponibilités des professeurs : un créneau bloqué par jour. Ces indisponibilités ne se chevauchent pas afin d'avoir suffisamment de cellules libres pour accueillir tous les cours (64 au maximum pour les tests effectués). Il existe donc plusieurs solutions optimales à ces conditions initiales.

La figure 5.2 montre l'évolution du temps de résolution en fonction du nombre d'agents. Passé son maximum (autour de 8 agents), le nombre de cycles diminue légèrement lorsque

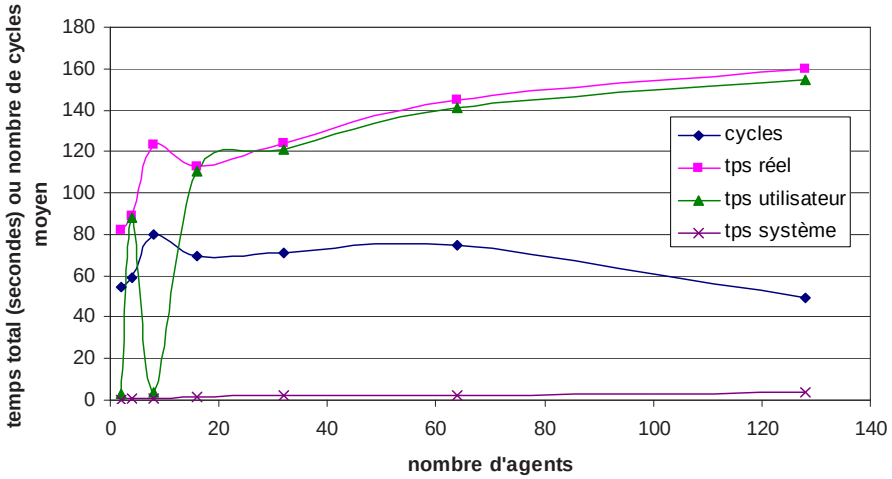

Figure 5.2 — Variation du temps de résolution en fonction du nombre d'agents.

le nombre d'agents augmente. L'espace de recherche restant de même dimension, les agents trouvent plus facilement un partenaire avec qui faire une réservation car l'environnement est plus densément peuplé. Le temps qui varie le moins entre deux occurrences du même test est le temps réel (temps mesuré par la machine). C'est donc l'indicateur le plus fiable pour mesurer l'évolution du temps de résolution. On remarque que son évolution est logarithmique, au delà de trente-deux agents dans l'environnement.

5.2.3 Expérimentation de la variante 2

Dans cette variante, les agents doivent relâcher des contraintes pour trouver une solution. La figure 5.3 présente l'efficacité de la résolution par ETTO, pour cette variante. Les réservations sont mises en place plus tardivement que les partenariats, comme dans la variante 1. Le système trouve la solution coûtant 10 points énoncée plus haut, en 265 pas. Cependant, certaines solutions mettent plus de 1000 pas pour être trouvées. Ceci est en partie dû à l'exploration aléatoire de la grille par les agents de réservation.

Une autre remarque importante est le fait que le système ne s'arrête jamais sur une solution. Même s'ils ont trouvé la solution, les agents n'en sont pas "conscients" et continuent à explorer la grille. Nous n'avons fixé aucun critère d'arrêt de la recherche. Nous considérons qu'il existe un oracle (le responsable des emplois du temps) qui choisira une solution fournie.

5.2.4 Expérimentation des variantes 3 et 4

Dans la variante 3, les contraintes deviennent dynamiques. Les indisponibilités des salles et des professeurs ou groupes d'élèves, ainsi que les besoins en vidéo projecteurs peuvent apparaître ou disparaître en cours de résolution. Dans la variante 4, ce sont les agents et les salles qui peuvent apparaître ou disparaître dynamiquement.

Compte tenu de la modélisation choisie, ajouter des contraintes n'apporte guère de diffé-

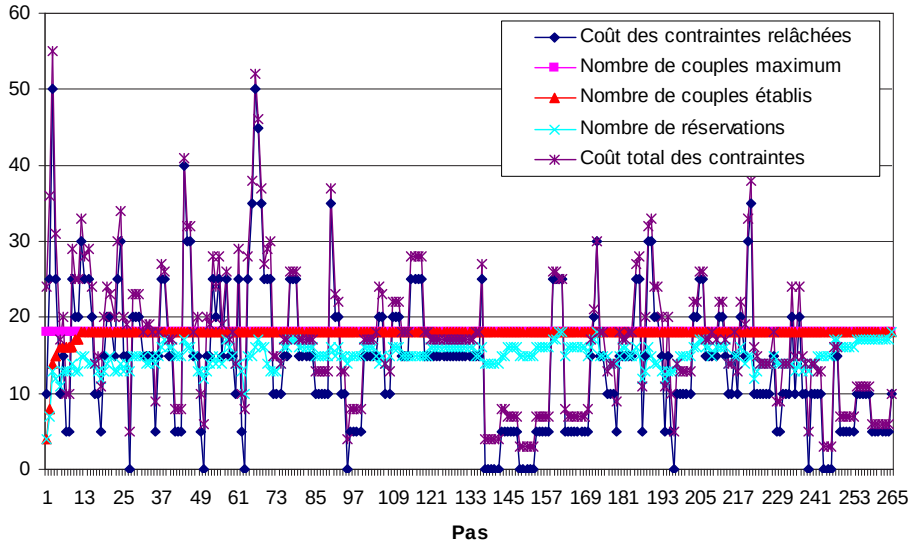

Figure 5.3 — Variation des contraintes en cours de résolution pour la variante 2.

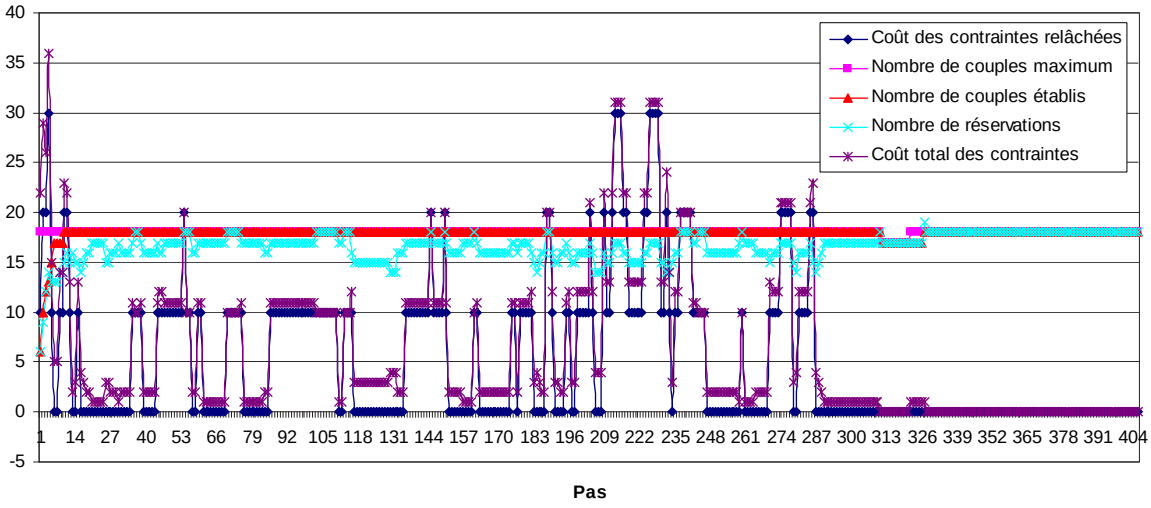

Figure 5.4 — Variation des contraintes en cours de résolution pour la variante 4, avec suppression d'un BookingAgent après stabilisation du système.

rence par rapport à l'ajout d'agents porteurs de contraintes. Nous avons lancé la résolution de la variante 1 avec une suppression d'un agent après que le système ait trouvé une solution. La figure 5.4 montre l'évolution des partenariats et des contraintes au cours de la résolution. Le système met 7 pas d'exécution pour retrouver la solution (pas n°329) après qu'un nouvel agent ait été ajouté (pas n°323). La figure 5.5 montre le même type de résultats pour une résolution durant laquelle huit agents de réservation ont été enlevés – ce qui correspond à la disparition d'un enseignant. ETTO trouve une nouvelle solution en très peu de pas, encore une fois (du pas n°384 au pas n°391).

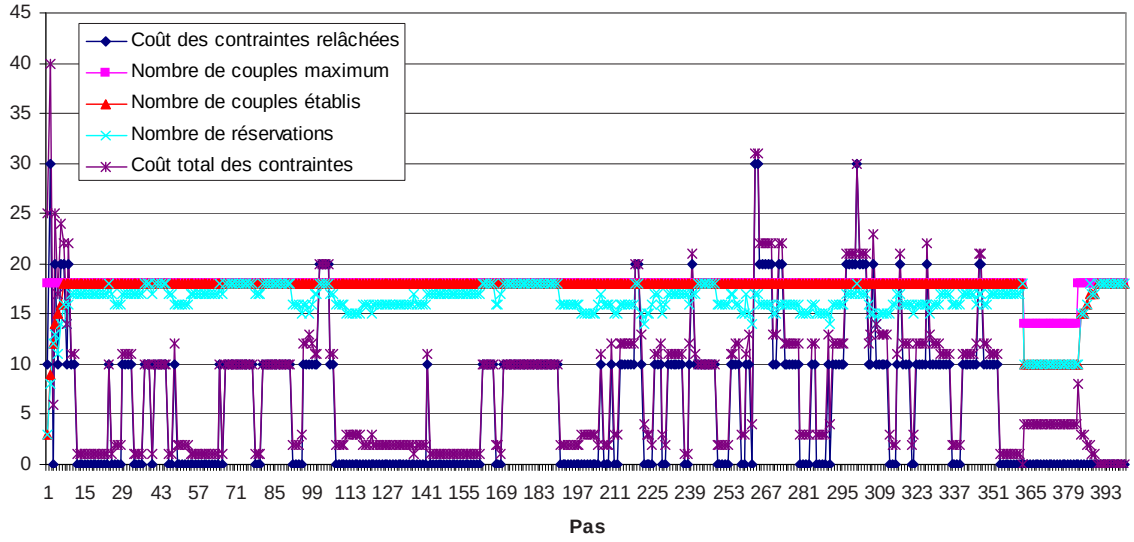

Figure 5.5 — Variation des contraintes en cours de résolution pour la variante 4, avec suppression de huit BookingAgents après stabilisation du système.

5.3 Discussion

ETTO est un excellent terrain d'expérimentations pour les AMAS, et d'application pour ADELFE. Bien qu'ADELFE nous permette d'identifier les agents, trouver les SNC relatives aux BookingAgents n'est pas forcément évident, et surtout pas automatique. Les expérimentations, bien qu'étant effectuées sur un prototype restreint, montrent l'intérêt de cette approche, tant au niveau conception, qu'au niveau utilisation.

Toutefois, ETTO possède actuellement plusieurs points discutables et remarquables, dont notamment les suivants :

- le système ne possède aucun critère d'arrêt. En effet, même si la solution est trouvée, les agents, n'ayant que des connaissances locales, continuent à chercher. C'est à la fois un point fort et un point faible. C'est un point fort car cela permet de générer des emplois du temps en continu, et d'ajouter des contraintes ou des agents à tout moment. C'est un point faible, car on est jamais sûr de l'optimalité de la solution ;
- les agents de réservation ne sont pas efficaces dans leur exploration de la grille. Comme nous l'avons vu, lorsque les agents de réservation se déplacent sur la grille, il choisissent leur destination aléatoirement. Cependant, il est tout à fait envisageable de l'implanter comme pour les échange de partenaires potentiels.
- en l'état actuel du logiciel, seules les modifications concernant les agents ont été implémentées car elles étaient prioritaires. Les salles ne sont pas des agents et leur possibilité d'ajout ou de suppression va apporter des modifications dans la gestion du monde dans lequel les agents évoluent ;
- il est intéressant de noter que lors du traitement de la variante 2, l'ajout d'agents surnuméraires peut favoriser l'amélioration de la solution. En effet, le système peut se fixer sur une solution qui n'est pas très bonne (toutes les contraintes relâchables l'ont

été et les agents trouvent une solution ainsi). L'agent surnuméraire, en bousculant les autres agents, favorise l'amélioration continue de la solution courante ;
– que ce soit pour l'ajout ou la suppression d'agent, les actions à mener sont relativement simples. Les agents étant coopératifs, ils annulent leurs éventuels réservations et partenariat avant de quitter le système ;
– ETTO n'as pas été comparé à d'autres approches, plus classiques. Cela figure dans nos perspectives.

6 Un problème de transport de ressources en robotique collective

ADELFE est destinée à concevoir des systèmes multi-agents auto-organisateurs, dans lesquels la fonction doit émerger des interactions entre agents. Dans ETTO, les interactions entre agents sont "classiques", dans le sens où, elles passent par des messages entre agents de type requête ou information. Cependant, de telles interactions ne sont pas toujours faciles ou possibles à mettre en place. Les interactions peuvent être moins directes et passer par l'environnement, ou uniquement par les perceptions que les agents ont sur leurs partenaires.

Dans l'optique de montrer comment un SMA auto-organisateur peut être modélisé sans utiliser de communications directes entre les agents, le problème de transport multi-robots de ressources est un exemple pertinent. En effet, dans ce chapitre, nous allons appliquer ADELFE à un problème de conception de collectif de robots. Cependant, les robots étant incapables de communiquer, nous établirons un mécanisme d'auto-organisation et un apprentissage minimal, uniquement basés sur la perception des robots . Initialement, ce problème avait été utilisé afin de montrer l'apparition de sens de circulation au sein de collectifs contraints comme un exemple de phénomène émergent [Picard, 2001; Picard et Gleizes, 2002].

N'étant qu'une application sans utilisateur, nous n'allons pas illustrer pleinement les phases préliminaires d'ADELFE qui concernent essentiellement les interactions entre le système et les utilisateurs, comme nous l'avons fait pour ETTO, mais seulement en présenter les principaux résultats d'analyse (§6.1). Ce chapitre se focalise donc, essentiellement, sur la conception des robots en tant qu'agents coopératifs, et sur l'apport de la coopération par rapport à des comportements nominaux. Par conséquent, l'étude des comportements coopératifs (§6.3) sera séparée de la conception des robots (§6.2). Enfin, une série d'expérimentations a été faite afin de valider notre approche et sera présentée dans le paragraphe 6.4.

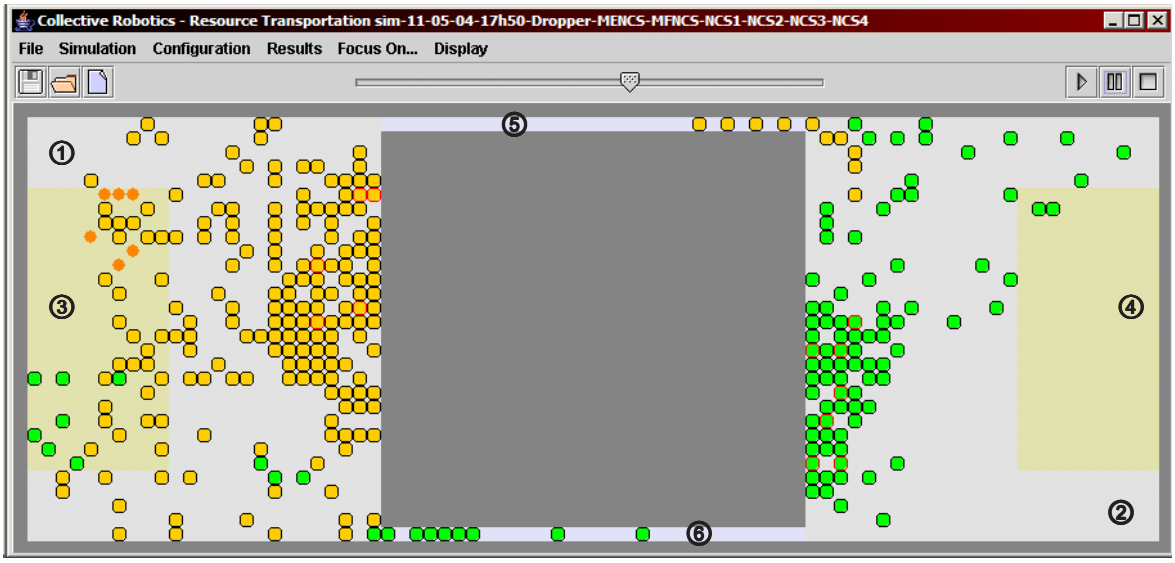

Figure 6.1 — L'environnement de transport multi-robot de ressources.

6.1 Transport collectif de ressources – travaux pré-conception

Le problème de transport de ressources par des robots est une tâche classique en robotique collective, souvent inspirée de mécanismes mis en place par les insectes sociaux pour s'adapter aux conditions de leur milieu [Vaughan *et al.*, 2000a; Kube et Zhang, 1993]. Les robots doivent transporter des ressources d'une zone de retrait jusqu'à une zone de dépôt. Ces zones sont situées dans deux salles différentes séparées par des accès trop étroits pour que deux robots puissent y passer côte à côte (voir figure 6.1). Ici apparaît un problème d'interférence spatiale, dans le sens où les robots doivent partager une ressource commune : les couloirs. Une fois engagé dans un couloir, que doit faire un robot rencontrant un autre robot circulant en sens inverse ? Ce problème d'interférence a été abordé par Vaughan *et al.*, pour des robots circulant dans des couloirs et devant traverser des passages étroits [Vaughan *et al.*, 2000b]. Leur solution a été de résoudre les conflits par une compétition agressive (avec hiérarchie explicite, c.-à-d. robots prioritaires identifiés *a priori*), similaire à l'éco-résolution [Ferber, 1995]. Simonin et Ferber proposent une résolution de tels problèmes grâce à un système d'attraction-répulsion basé sur le déclenchement de comportements altruistes, vision inversée de l'éco-résolution, et sur la communication d'intentions [Simonin et Ferber, 2003]. Pour notre part, nous proposons une vision intermédiaire, dans laquelle les robots ne sont ni altruistes, ni individualistes et ne possèdent aucun moyen de communiquer directement leurs intentions. De plus nous n'utiliserons pas de système à planificateur de trajectoire. En effet, l'utilisation de la planification est rarement utilisée dans ce genre de tâche car elle reste peu performante, compte-tenu de la forte dynamique de l'environnement d'un robot.

Nous allons brièvement exposer les travaux préliminaires des besoins et d'analyse, afin de nous concentrer sur la conception des comportements coopératifs des robots.

6.1.1 Les besoins et le cahier des charges

L'environnement se présente sous la forme de deux salles séparées par plusieurs couloirs étroits (voir figure 6.1). La configuration de cet environnement aura bien sûr une grande incidence sur l'apparition et la nature des phénomènes émergents. Les salles disposent d'une zone de dépôt ④ (dans la salle de dépôt ②) ou d'une zone de retrait ③ (dans la salle de retrait ①). Ces zones se situent au fond des salles, contre le mur qui fait face aux couloirs, afin de forcer les robots à bien rentrer dans la salle pour prendre ou déposer des ressources. Les couloirs ⑤ et ⑥ sont étroits, et leur longueur peut être paramétrée. En effet, on peut penser que leur longueur aura une grande importance sur l'apparition d'un sens de circulation. Deux cas se présenteront :

- soit la longueur du couloir est inférieure à la distance de perception des robots et un robot pourra voir si un robot est déjà engagé dans le couloir qu'il a choisi ;
- soit la longueur est supérieure à la distance de perception et un robot devra s'engager avant de constater que le couloir est libre ou occupé.

On remarque aussi que le nombre de couloirs (2 dans l'exemple) peut être paramétré. Ce nombre a lui aussi une influence sur l'apparition de sens de circulation. En effet, le cas où un seul couloir sépare les deux salles n'est pas à prendre en compte : nous retombons alors sur un problème d'interblocage tout à fait similaire à celui développé par Vaughan *et al.*. Mais il semble intéressant d'étudier l'influence d'un nombre supérieur ou égal à deux.

Considérant que l'analyste, ou le développeur, ne possède aucun robot physique, nous proposons de faire un système de simulation du problème proposé. Le développeur aura donc à construire son propre outil de simulation ou bien à réutiliser un outil existant (par exemple, oRis). Le choix que nous avions pris dans l'étude précédente fut de simuler le système grâce à oRis, qui est gourmand en ressources car c'est un outil graphique de haut niveau [Harrouet, 2000; Picard, 2001; Picard et Gleizes, 2002]. Pour cette étude, nous allons développer notre propre système "léger" de simulation en Java.

Aucune contrainte de réutilisabilité n'a été fixée sur le système à construire. Le système sera donc construit sans extension possible évidente. Par contre, le souhait de pouvoir lancer de nombreuses simulations implique la prise en charge de sauvegardes. De plus, concernant la simulation, il sera prévu que l'environnement soit dynamique (ajout de ressource, fermeture ou ouverture de couloirs, ...) afin d'observer différents comportements d'adaptation à un environnement de simulation dynamique.

Enfin, une dernière contrainte, et non des moindres, est de pouvoir modifier le comportement des robots et de pouvoir lancer des simulations pour des robots possédant des comportements coopératifs incomplets ou inexistants afin de montrer l'apport de la coopération.

Ce problème étant un problème de simulation, deux niveaux sont à analyser : celui de l'application de simulation (système à utilisateur(s)), et celui du système à simuler (le collectif de robots). Nous allons nous concentrer sur le niveau des robots, car les problématiques soulevées par la définition d'un environnement de logiciels avec utilisateur(s) ont été illustrées par ETTO (voir chapitres 3 et 5).

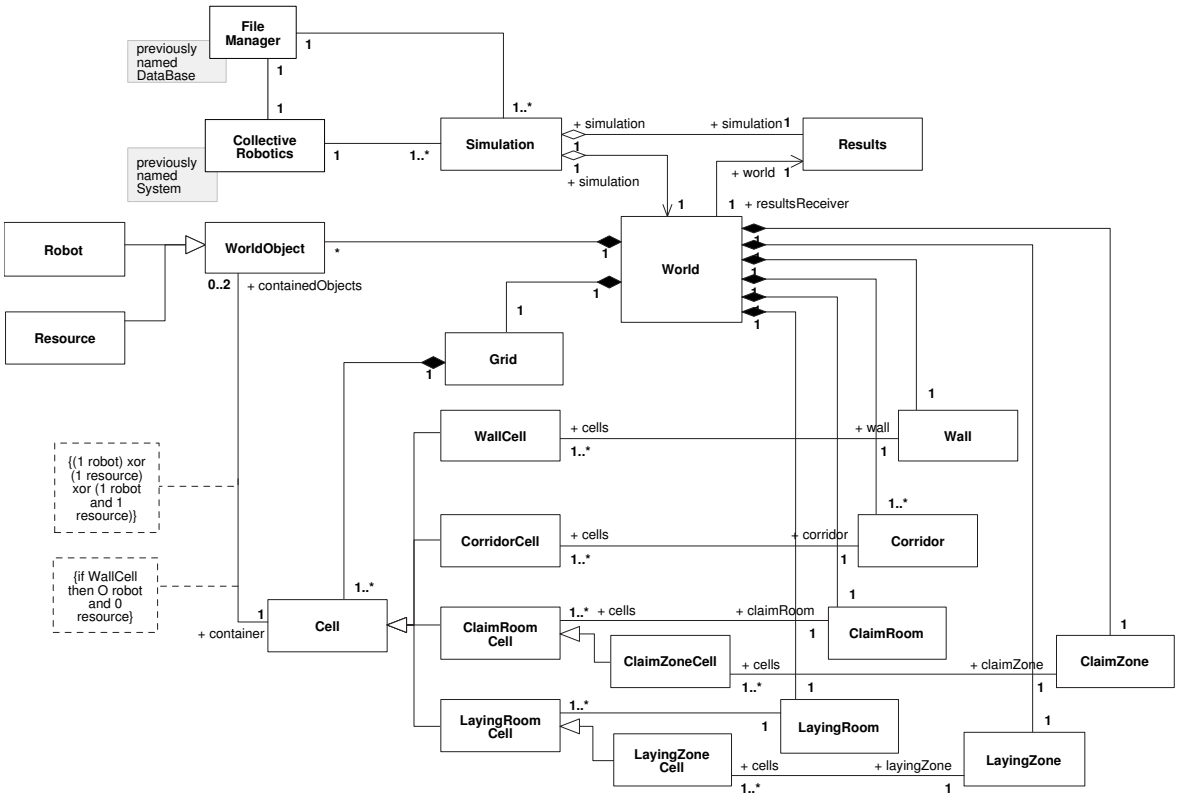

Figure 6.2 — Le diagramme de classes préliminaire d'analyse pour le problème de transport collectif.

6.1.2 Analyse

A partir du cahier des charges, nous avons construit un diagramme de classes préliminaire, représenté en figure 6.2. Une fois les entités identifiées, le système peut être analysé afin de déterminer si l'utilisation des AMAS est nécessaire ou non, et si une décomposition de certaines parties en AMAS est envisageable. Le système multi-robot est composé d'un nombre fixe de robots, mais ce nombre peut être élevé. L'activité corrélée des robots semble nécessaire afin d'obtenir une meilleure efficacité collective. Compte tenu des précédentes recherches présentées dans le cahier des charges, il ne semble pas nécessaire d'opter pour une architecture d'agent à gros grain, ce qui indique qu'il ne sera pas nécessaire de décomposer les robots en systèmes multi-agents pour leur contrôle (contrairement à ce que nous avions fait dans une étude précédente [Picard, 2001; Picard et Gleizes, 2002]). Cependant l'environnement d'un robot est très dynamique car constitué des autres robots qui bougent. Il est donc utile d'utiliser les AMAS pour développer le système multi-robot.

Maintenant, il faut analyser toutes les classes du domaine apparaissant dans le diagramme : robots, ressources, murs, couloirs, zones, et salles. Dans ce cas, les seules classes vérifiant les caractéristiques des agents sont les robots. En effet, les ressources n'ont pas un comportement autonome et ne sont mues que par les robots. Leurs caractéristiques sont fortement statiques. Les murs, les salles et les zones ne sont que des objets du monde sans comportement dynamique. Par contre, les robots, dans le monde réel, sont des entités autonomes, non radioguidées et ayant un comportement dynamique non dirigé par une quel-

conque autre entité (contrôleur). De plus, ce sont les seules entités à pouvoir agir pour résorber des échecs à la coopération comme des blocages dans les couloirs. En conclusion, le seul candidat pertinent pour être stéréotypé «cooperative agent» est la classe Robot.

6.2 Conception des robots transporteurs

Nous allons maintenant instancier le modèle d'agent coopératif afin de concevoir des robots capables de réaliser cette tâche de transport de manière coopérative. Ce travail s'inscrit dans la phase de conception d'ADELFE (WD_4, §3.6), et plus particulièrement dans l'activité de conception des agents (A_{16}, §3.6.4).

6.2.1 Définition des différents modules

Concernant les perceptions des robots, il semble possible de leur permettre de connaître la position des zones de retrait et de dépôt. En effet, on souhaite se focaliser sur l'adaptation pour un problème de circulation et non de fourragement, c.-à-d. que la tâche des robots n'est pas de trouver l'emplacement des ressources, mais de les transporter de salle en salle. Voici une liste des perceptions possibles pour les robots : position de la zone de retrait, position de la zone de dépôt, un cône de perception dans lequel les objets sont identifiables (robot, boîte ou mur), une information sur l'état de chacun des objets (portant une boîte, par terre, libre, ...), des capteurs de proximité (devant, derrière, à droite et à gauche), une boussole, position absolue dans l'espace[1]. L'environnement est discrétisé par une grille dont chaque case représente une partie élémentaire sur laquelle peut se situer un robot, une boîte ou un mur. Le module de perception définit aussi une valeur limite de perception (5 cases, par exemple).

Les actions possibles d'un robot sont les suivantes : *repos*, *prendre*, *poser*, *avant*, *arrière*, *droite* et *gauche*. Il est à noter que les robots ne peuvent poser les boîtes n'importe où dans l'environnement ; uniquement dans la zone de dépôt. Ils ne peuvent pas communiquer directement ou déposer de marques sur l'environnement, nous n'utiliserons donc pas le formalisme des protocoles A-UML pour spécifier leur module d'interaction.

Les compétences doivent permettre à un robot de remplir la tâche de transport de ressources. Pour cela, un robot peut calculer quel objectif il doit atteindre compte tenu de ses perceptions : s'il porte une boîte, il doit atteindre la zone de dépôt sinon il doit chercher une boîte dans la zone de retrait. En fonction du but courant, le module de compétences doit fournir une action à effectuer pour atteindre l'objectif voulu. Les buts d'un agent sont d'*atteindre la zone de retrait* ou d'*atteindre la zone de dépôt*. De plus, les robots possèdent des caractéristiques physiques, comme leur vitesse, le nombre de boîtes transportables, ou la préférence à vouloir aller vers l'avant plutôt que faire demi-tour, comme peuvent le faire des fourmis fourrageuses (voir §1.4 et §2.2). De telles préférences sont appelées valeurs réflexes et correspondent à des traits stéréotypés «characteristic».

[1]Ceci est certainement le point faible de notre approche, car c'est une connaissance de haut niveau. Cependant, il est possible d'imaginer que l'environnement possède des codes couleurs de position, des bornes d'identification ou un système GPS.

Les représentations que possèdent un robot sur son environnement sont assez limitées. À partir de ces perceptions, il ne peut identifier un robot parmi les autres, mais peut savoir s'il porte ou non une boîte. Il peut mémoriser ses anciens positions absolues, direction, but et action.

Le module d'aptitudes doit permettre au robot de choisir une action en fonction de son état. Il devient nécessaire de faire un choix de conception à ce stade. Le module de compétences, en fonction du but courant du robot, fournit des valeurs de préférences pour chacune des actions. Le module d'aptitudes doit choisir parmi ces préférences, quelle action exécuter afin d'atteindre le but courant. Le module d'aptitudes utilise, ici, une fonction de tirage de Monte Carlo sur le vecteur de préférences d'action fourni par le module de compétences.

De la même manière, le module de coopération doit fournir des vecteurs de préférences afin de résoudre les situations non coopératives décrites dans la section 6.3.

6.2.2 Décision et choix des actions

À chaque instant t, un robot devra donc choisir entre les différentes actions proposées par les différents modules de décision (aptitudes et coopération). À l'instant t, chaque action act_j du robot r_i va être évaluée. Pour chaque action, cette valeur est calculée en fonction des perceptions, des représentations et des réflexes, dans le cas d'un fonctionnement nominal :

$$V_{r_i}^{nomi}(act_j, t) = wp_{r_i}(act_j, t) + wm_{r_i}(act_j, t) + wr_{r_i}(act_j)$$

où :

- $V_{r_i}^{nomi}(act_j, t)$ représente la valeur de l'action act_j au temps t pour le robot r_i ;
- $wp_{r_i}(act_j, t)$ représente la valeur (ou poids) calculée en fonction des perceptions ;
- $wm_{r_i}(act_j, t)$ représente la valeur calculée en fonction de la mémoire ;
- $wr_{r_i}(act_j, t)$ représente la valeur calculée en fonction des réflexes.

Comme pour les aptitudes, un vecteur d'action coopérative est généré par le module de coopération : $V_{r_i}^{coop}(act_j, t)$. Une fois les valeurs calculées pour chacune des actions du robot, par les deux modules, le vecteur sur lequel portera le tirage de Monte Carlo est une combinaison des deux vecteurs dans laquel le vecteur de coopération subsume le vecteur nominal :

$$V_{r_i}(t) = V_{r_i}^{nomi}(t) \prec V_{r_i}^{coop}(t)$$

6.3 Étude des comportements coopératifs

Dans les paragraphes précédents, nous avons détaillé les différents modules d'un robot et leurs composantes, à l'exception du module de coopération. Cette section a pour but de décrire les règles de coopération à mettre en oeuvre pour que le système multi-robot soit en adéquation fonctionnelle avec son environnement.

6.3.1 Gestion coopérative des blocages

À partir de deux robots effectuant la tâche de transport dans le même environnement, le comportement nominal peut ne plus être adéquat. En effet, un robot possède les compétences pour remplir sa tâche, mais pas pour travailler avec les autres. Dans cet environnement, fortement contraint à cause des couloirs étroits, des zones d'interférences spatiales apparaissent. Si deux robots, l'un portant une boîte et se dirigeant vers la zone de dépôt, et un autre robot, se dirigeant vers la zone de retrait pour retirer une boîte, se rencontrent face-à-face dans un couloir, ils restent bloqués – car il ne peuvent poser les boîtes que dans la zone de dépôt. Il devient alors nécessaire de pourvoir les robots de comportements coopératifs. Nous distinguons alors deux situations non coopératives pouvant être résolues de manière réactive :

Un robot est bloqué. Un robot r_1 ne peut plus avancer à cause d'un mur, ou d'un robot r_2 allant dans le sens opposé[2]. Dans ce cas, s'il en a la possibilité, r_1 se déplacera sur les côtés. Ceci correspond à une augmentation des valeurs du vecteur d'actions coopératives correspondant aux actions de déplacements latéraux : $V_{r_1}^{coop}(t, droite)$ et $V_{r_1}^{coop}(t, gauche)$. Dans le cas où r_1 n'a pas de choix de déplacements latéraux, deux autres possibilités sont ouvertes. Si r_2 est un robot ayant un but antagoniste, c'est le robot qui est le plus loin d'atteindre son but qui se retourne (augmentation de $V_{r_i}^{coop}(t, arriere)$) pour laisser place à celui qui est le plus proche de son but (qui augmente $V_{r_i}^{coop}(t, avant)$ quitte à attendre un tour). Dans le cas où r_2 a le même but, sauf si r_1 est suivi d'un robot ayant un but antagoniste ou que r_1 s'éloigne de son but (visiblement il se déplace vers une zone à risque), r_1 se retourne ; sinon r_1 avance et r_2 recule.

Un robot se retourne. Un robot r_1 se retourne[3] à cause d'un blocage. S'il en a la possibilité, r_1 se déplacera sur les côtés. Sinon, r_1 continue à avancer jusqu'à ce qu'il n'y ait plus de marge de manœuvre ou bien qu'il rencontre un autre robot r_2 qui se retourne et qui est plus près de son but que lui. La table 6.1 résume le comportement du robot face à cette situation.

Ces règles correspondent à des *conflits* de ressources (les couloirs) ou bien à de l'*inutilité* lorsque le robot se retourne et s'éloigne de son but. Dans le cas des robots, nous ne spécifierons pas de situations d'incompréhension, par exemple, car les robots sont incapables de communiquer directement. Ces règles, relativement simples à spécifier, assurent que les robots ne s'inter-bloquent pas dans les couloirs. Par contre, ces situations ne font que résoudre les problèmes sur l'instant, créant des mouvements de retournement et donc de perte de temps pour rapporter ou aller chercher des boîtes.

6.3.2 Anticipation des situations non coopératives

Compte tenu de la remarque précédente, il semble possible de spécifier des règles de coopération permettant d'anticiper les situations de blocage afin que le collectif soit plus efficace. Dans ce cas, nous parlerons de règles de coopération d'*optimisation*. Les règles pré-

[2] Dans le cas où r_2 se déplace sur les côtés ou dans le même sens que r_1, il n'est pas considéré comme bloquant car il ne gênera plus au pas suivant.

[3] Un robot sera considéré comme se retournant tant qu'il n'aura pas le choix de déplacements latéraux.

Condition	*Action*
$ret \wedge libD$	$\nearrow V_{r_i}^{coop}(t, droite)$
$ret \wedge libG$	$\nearrow V_{r_i}^{coop}(t, gauche)$
$ret \wedge \neg(libG \vee libD) \wedge ant \wedge vBut \wedge pBut$	$\nearrow V_{r_i}^{coop}(t, arriere)$
$ret \wedge \neg(libG \vee libD) \wedge ant \wedge vBut \wedge \neg pBut$	$\nearrow V_{r_i}^{coop}(t, avant)$
$ret \wedge \neg(libG \vee libD) \wedge ant \wedge \neg vBut$	$\nearrow V_{r_i}^{coop}(t, arriere)$
$ret \wedge \neg(libG \vee libD) \wedge \neg ant$	$\nearrow V_{r_i}^{coop}(t, avant)$

Avec :
- *ret* : le robot est en retournement ;
- *libD* : la case à droite est libre ;
- *libG* : la case à gauche est libre ;
- *ant* : un robot antinomique est devant ;
- *vBut* : le robot se dirige vers son but ;
- *pBut* : le robot est plus proche de son but que d'un but antinomique.

Tableau 6.1 — Exemple de formulation de la situation non coopérative d'inutilité

cédentes permettent à un robot de résoudre une situation de blocage. Un robot se retrouve dans une telle situation parce qu'il est entré dans une zone empruntée par des robots remplissant un but antinomique. Afin d'anticiper cette situation, il faut donc que les robots soient capables d'éviter de rentrer dans une zone à risque, c.-à-d. une zone d'où sortent des robots ayant un but antinomique. Ainsi, une règle d'anticipation peut être définie :

Un robot voit un robot antinomique. Si un robot r_1 voit un robot r_2 ayant un but antinomique (un état de transport différent), si r_1 peut se déplacer sur les côtés, il le fera, sinon il continue à avancer.

Cette anticipation réactive présente cependant un problème majeur : une fois qu'un robot a évité une zone à risque, rien n'assure qu'il n'y revienne à nouveau, guidé par son but. Afin de pallier cet oubli, munissons notre robot d'une mémoire (dans le module représentation) des zones à risque. Chaque fois qu'un robot r_i rencontre une situation non coopérative d'anticipation face à un robot r_j, il va ajouter à sa mémoire un tuple (ou balise virtuelle) $\langle posX(r_j,t), posY(r_j,t), goal(r_i,t), w\rangle$ où $posX(r_i,t)$ et $posY(r_i,t)$ représentent les coordonnées de r_j au moment de la situation non coopérative ; $goal(r_i,t)$ représente le but que r_i essayait de remplir au moment de la situation ; w représente une valeur de répulsion. Plus la valeur w est élevée, plus le robot aura tendance à éviter la zone lorsqu'il tente de remplir un but autre que $goal(r_i,t)$. Ainsi, le robot analyse toutes les balises dont la distance est inférieure à sa valeur de perception (pour conserver le principe de localité). Une balise de poids w située dans la direction dir à une distance de d cases induira une augmentation de $V_{r_i}^{coop}(t, dir_{opp})$ de w où dir_{opp} indique la direction opposée à dir.

Comme la mémoire est limitée, les tuples ajoutés doivent pouvoir disparaître au fur et à mesure de la simulation. Par exemple, le poids w doit pouvoir diminuer d'une certaine valeur δ_w (appelée *facteur d'oubli*) à chaque pas de simulation. Une fois que w devient nul, le tuple est enlevé de la liste. Cette méthode correspond à une utilisation de phéromones *virtuelles* et *personnelles*. Enfin, comme chez les fourmis fourrageuses, les robots renforcent leurs balises : un robot passant par la position d'une de ces balises et ayant un but autre que le but courant, ré-initialise la valeur de la balise. En effet, si le robot se trouve sur cette position, c'est bien que cette zone doit être évitée lorsqu'il remplit un autre but.

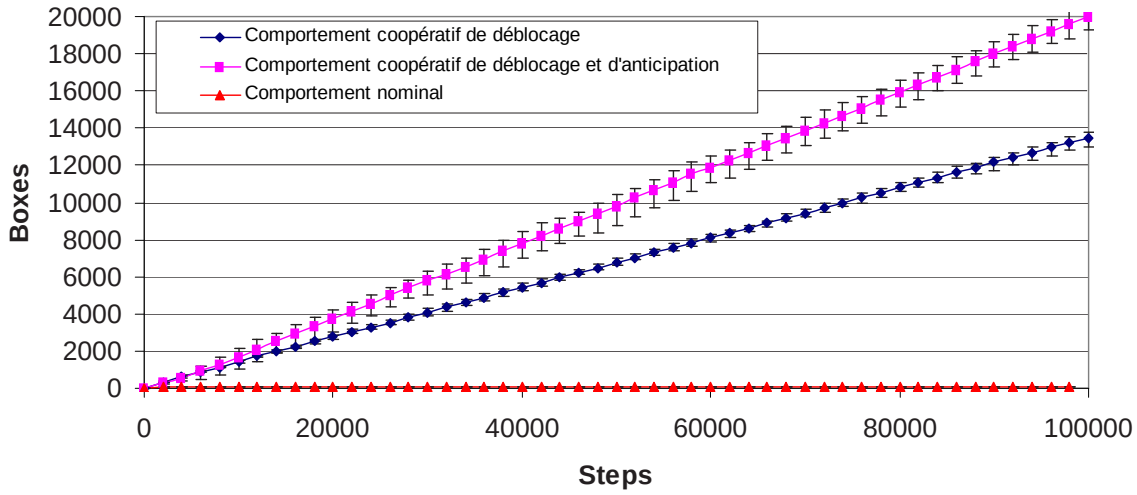

Figure 6.3 — Nombre de boîtes rapportées pour 15 simulations (300 robots, 2 couloirs, 5 cases de perception), du comportement nominal (individualiste) et des deux comportements coopératifs : le comportement de déblocage (voir 6.3.1) et le comportement d'anticipation avec déblocage (voir 6.3.2)

6.4 Expérimentations et résultats

Afin de valider notre approche incrémentale de l'attribution de comportements coopératifs aux robots transporteurs, le modèle a été implémenté et simulé grâce à une plate-forme Java permettant de régler de nombreux paramètres.

6.4.1 Dispositif expérimental

L'environnement de simulation correspond à deux salles (25 x 30 cases) séparées par deux couloirs longs et étroits (30 x 1 case) comme dans la figure 6.1. 300 robots sont placés aléatoirement dans la salle de retrait. Ces robots possèdent une capacité de perception de 5 cases. Ils peuvent se déplacer d'une case par tour. S'ils possèdent la capacité à anticiper les conflits, leur mémoire peut contenir 1500 tuples avec $w = 400$ et $\delta_w = 1$.

6.4.2 Comparaison réaction/anticipation

La figure 6.3 présente une comparaison des résultats pour les comportements coopératifs présentés ci-dessus. Le comportement de déblocage permet d'obtenir une efficacité linéaire permettant aux robots de remplir leur tâche de transport sans être bloqués, tandis que les robots avec comportement nominal (et individualiste) sont systématiquement bloqués. En ajoutant l'anticipation de blocage, le collectif gagne en efficacité (au moins 30% de boîtes rapportées en plus). Ceci revient à avoir défini une optimisation du comportement de déblocage. Conformément à la théorie des AMAS, nous observons expérimentalement que la résorption des situations non coopératives locales conduit à l'adéquation fonctionnelle collective. Enfin, plus les situations non coopératives sont prises en compte localement, meilleures sont les performances.

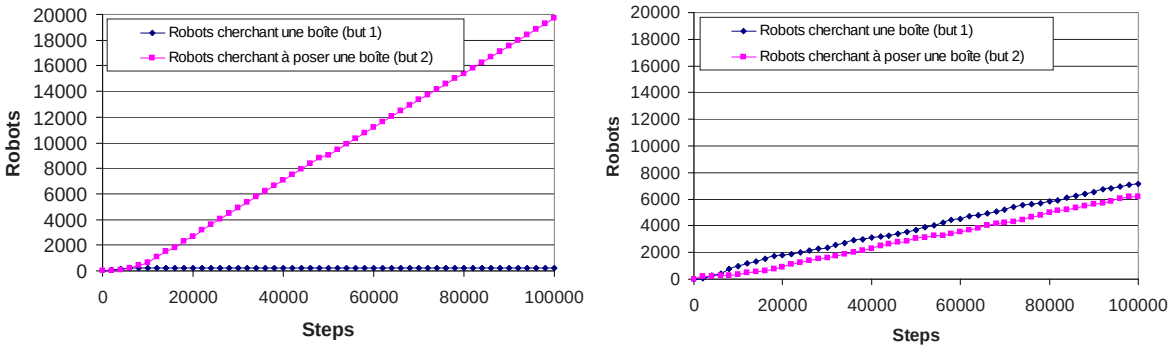

Figure 6.4 — Courbe de fréquentation d'un couloir pour les deux comportements coopératifs : le comportement de déblocage uniquement (à droite) et le comportement d'anticipation avec déblocage (à gauche).

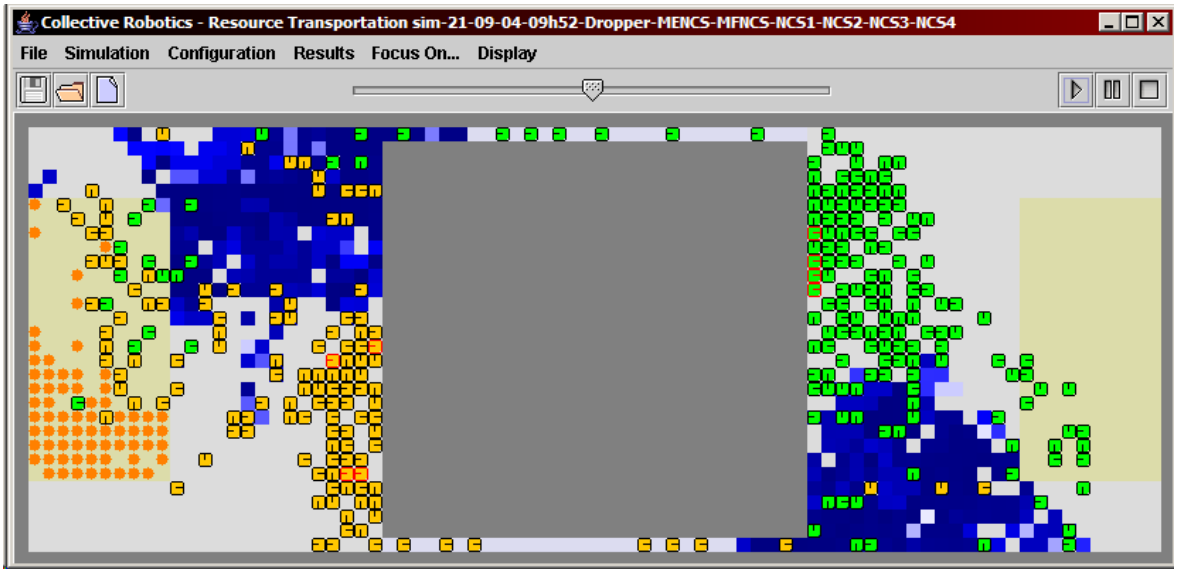

Figure 6.5 — Placement des balises (en bleu) pour les deux buts.

6.4.3 Émergence de sens de circulation

La figure 6.4 présente la fréquentation d'un couloir pour les deux comportements coopératifs. Dans le cas du comportement coopératif avec anticipation nous pouvons observer l'émergence de sens de circulation : le couloir est uniquement fréquenté par des robots cherchant une boîte. Au contraire, dans le cas d'un comportement de déblocage simple, le couloir est fréquenté par des robots ayant des buts différents.

Les robots coopératifs par anticipation dédient les couloirs à des buts particuliers, comme le précise la figure 6.5 dans laquelle apparaissent les balises virtuelles (en bleu plus ou moins foncé, en fonction de la concentration) de tous les robots[4]. Nous pouvons ici attribuer le caractère émergent à ce phénomène car les robots n'ont aucune notion de couloir – contrairement à des précédents travaux [Picard et Gleizes, 2002]. Ainsi, uniquement grâce

[4]C'est une agrégation de toutes les balises à titre informatif : les robots ne les perçoivent pas toutes. C'est une sorte de mémoire commune observée.

aux informations de bas niveau dont ils disposent, ils ont établi un comportement de circulation ce qui conduit à une optimisation de leur rendement. De plus, le sens de circulation peut varier à chaque simulation : une petite variation initiale de quelques robots va rapidement en décider. Ici, d'après la distinction faite par [Müller, 2003], nous parlerons d'émergence dite *faible* car les robots n'ont pas la connaissance (globale) du nombre de boîtes qu'ils rapportent et ceci ne les encourage pas pour rapporter plus de boîtes : il n'y a pas de *feedback* sur la quantité transportée.

6.4.4 Question d'optimalité

Lorsque l'on met en place des algorithmes d'apprentissage et d'adaptation, la question de l'optimalité est très souvent soulevée. Ici, la tâche est simple, donc l'attribution de sens de circulation aurait pu être codée dans les agents, au lieu d'être apprise[5]. Mais, l'optimalité du système n'en reste pas moins difficile à définir, même après avoir extrait le temps mis pour l'apprentissage, qui ne dure que quelques milliers (soit une dizaine d'aller-retours) de pas dans les configurations de simulations adoptées.

En considérant, l'environnement de simulation de la figure 6.1, le temps pour un robot de faire un cycle *"prend boîte - pose boîte - prend boîte"* est en moyenne optimale de :

$$t = (l_{salle_1} + \frac{h_{salle_1}}{2} + l_{couloir_1} + l_{salle_2} + \frac{h_{salle_2}}{2}) \times 2 = 220\ pas$$

Ceci signifie que, théoriquement, 300 robots devraient transporter 135000 ($\approx 300 \times \frac{100000}{220}$) boîtes en 100000 pas de temps, soit près de 6,75 fois plus que le collectif de robots coopératifs étudié (voir figure 6.3). Cependant, ceci ne reste qu'un optimum théorique : un seul chemin y correspond, ce qui signifie que faisant 220 cases de long, au maximum 220 robots peuvent le suivre. Mais, même avec une telle configuration, les robots ne seraient pas efficaces car ils auraient toujours un robot devant eux. Donc, il faut leur laisser une case libre devant, afin d'être plus efficace dans leur mouvement ; ce qui réduit le collectif à 110 individus pour atteindre une optimalité réelle pour cet environnement. Le nombre théorique optimal de boîtes rapportées est alors de 50000 ($\approx 110 \times \frac{100000}{220}$). Des simulations faites avec 110 robots coopératifs montre une moyenne de 6200 boîtes rapportées, soit 8,3 fois moins que l'optimum.

Est-ce un résultat décevant pour autant ?

Non, car même s'il est loin de l'optimum, le comportement coopératif est de plus en plus efficace avec l'augmentation du nombre de robots. En effet, bien que l'environnement devienne de plus en plus contraint compte tenu de la densité de population, le nombre moyen de pas nécessaire pour un robot pour faire un cycle *"prend boîte - pose boîte - prend boîte"* passe de 1826 à 1485, lorsque le nombre de robots passe 110 à 300 robots.

[5]Mais n'oublions pas, que le but de ces expériences était originellement de montrer un exemple de comportement collectif émergeant des interactions coopératives locales.

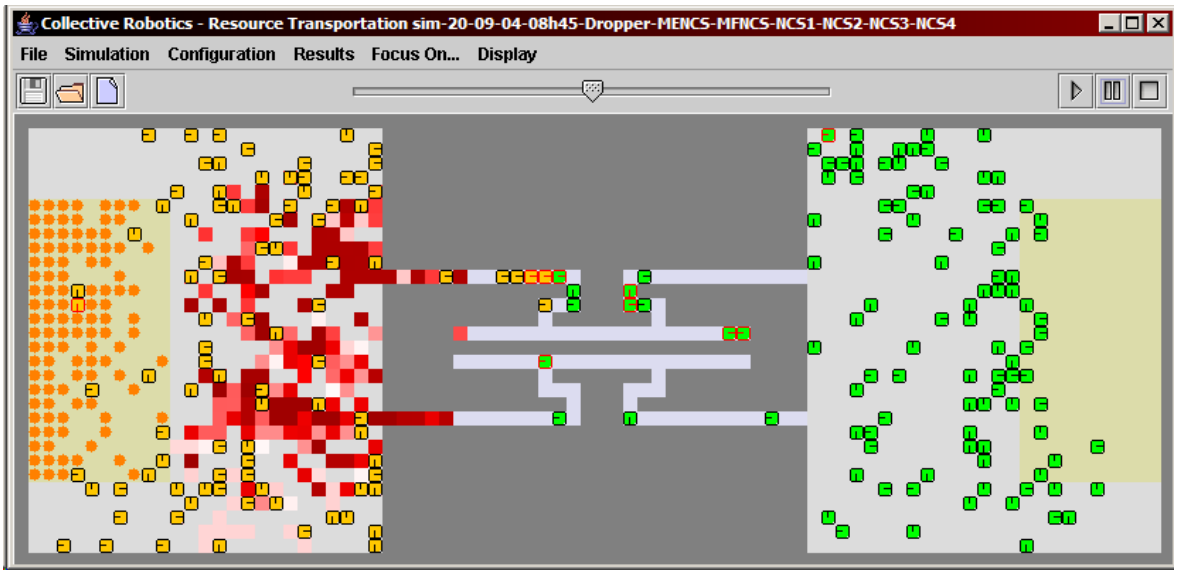

Figure 6.6 — Environnement "difficile". Placement des balises (en rouge) dans un environnement "difficile" pour le but *atteindre la zone de dépôt*.

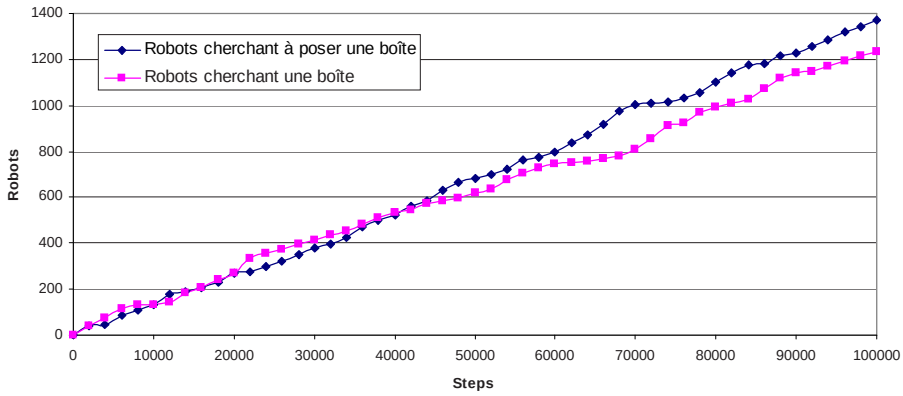

Figure 6.7 — Fréquentation d'un couloir dans un environnement "difficile".

6.4.5 Environnements difficiles

L'environnement de simulation des expérimentations et du cahier des charges est très simple : couloirs distants, pas d'impasse, couloirs droits.

6.4.5.1 Environnement avec virages et impasses

Pour vérifier, à titre indicatif, si le comportement des agents est robuste et adapté à des environnements plus difficiles, des simulations ont été faites dans un monde présenté dans la figure 6.6.

L'analyse de ces simulations montre que :

– les robots ont du mal à différencier les couloirs, et donc que le sens de circulation ne se met pas en place. La figure 6.6 montre l'ensemble des balises (en rouge) de tous les robots pour le but *atteindre la zone de dépôt*. Les balises sont placées devant les deux cou-

loirs, ce qui indique que les robots ne vont préférer aucun couloir, comme le confirme le graphe de fréquentation correspondant de la figure 6.7 – comme dans le cas d'un comportement sans anticipation. Par conséquent, l'efficacité du collectif en souffre ;
– il n'y a pas de blocage dans les couloirs ou les impasses : l'efficacité, même faible, reste constante.

Le comportement collectif coopératif semble donc robuste (pas de blocage), bien que perdant en efficacité (comme il n'y a pas d'apprentissage de sens de circulation). Pour résoudre ce problème, deux possibilités s'offrent à nous :
– changer des paramètres de simulation : la distance de perception, et donc la surface des zones balisées (car un robot place des balises dans son champ de vision) ou la force de répulsion des balises par exemple. Mais ceci signifie que le comportement coopératif ne serait plus adéquat en de telles circonstances car trop dépendant de paramètres généraux ;
– ajouter l'apprentissage coopératif sur ces paramètres, c.-à-d. trouver un moyen de faire apprendre aux robots à s'adapter à de telles situations, en ajoutant, par exemple, de nouvelles définitions de SNC.

6.4.5.2 Environnement avec fermeture aléatoire de couloirs

Un autre point à analyser est le comportement du collectif face à un environnement (hors robots) dynamique dans lequel les couloirs se ferment au hasard. Ici, le collectif perd encore en efficacité, mais ne reste pas bloqué. Bien sûr, des obturations de couloirs trop fréquentes ne permettent pas au collectif d'apprendre un sens de circulation. En effet, il faut quelques milliers de pas sont nécessaires pour qu'un collectif de 300 robots apprenne. Cependant, sur un temps assez long, le contrecoup de la dynamique de l'environnement est négligeable.

La figure 6.8 montre le placement des balises virtuelles dans un environnement à quatre couloirs, dont deux choisis au hasard, se ferment tous les 10000 pas de temps. Seules les extrémités sont bloquées, et certains robots peuvent se retrouver prisonniers dans les couloirs, et ne plus aider le collectif. Le collectif s'adapte bien, et assez vite, comme le montrent la figure 6.9, qui présente la fréquentation des couloirs, et la figure 6.10 qui présente le nombre de boîtes rapportées. Lors de toutes les périodes de 10000 pas, les deux couloirs ouverts sont exploités et un sens de circulation leur est affecté. Le nombre de boîtes rapportées est du même ordre que lorsque l'environnement est statique à deux couloirs. Le temps d'apprentissage est amorti par l'emprisonnement de certains robots : l'environnement est moins saturé. Ceci est un exemple d'adaptation du collectif à un environnement dynamique et difficile.

6.5 Discussion

L'instanciation du modèle proposé par ADELFE présente plusieurs points forts. Premièrement, contrairement aux algorithmes inspirés des fourmis fourrageuses, les robots ne marquent pas leur environnement avec des phéromones, mais gardent des balises virtuelles et personnelles. Deuxièmement, contrairement aux approches de [Vaughan *et al.*, 2000b] ou [Simonin et Ferber, 2003], les robots n'ont pas besoin de moyens de communication directe

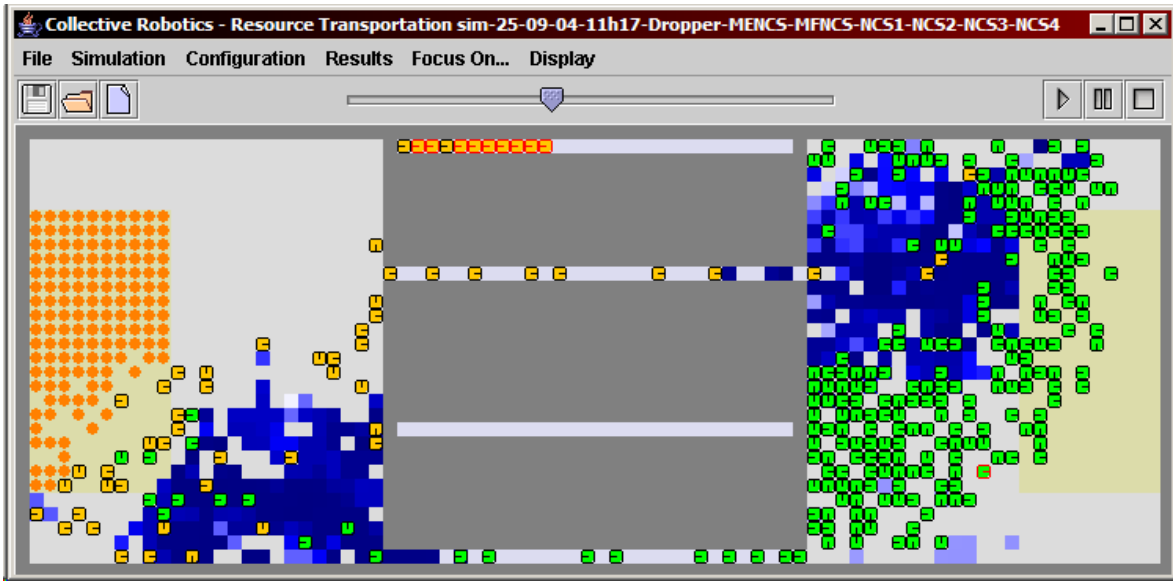

Figure 6.8 — Placement des balises virtuelles dans un environnement dynamique à quatre couloirs dont deux sont obturés, pour 300 robots.

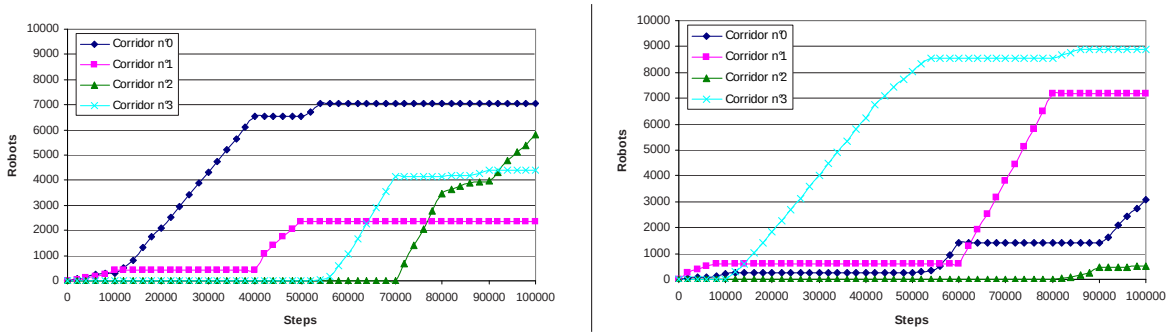

Figure 6.9 — Fréquentation des couloirs pour les deux buts : *atteindre la zone de retrait* (à gauche) ou *atteindre la zone de dépôt* (à droite), pour 300 robots.

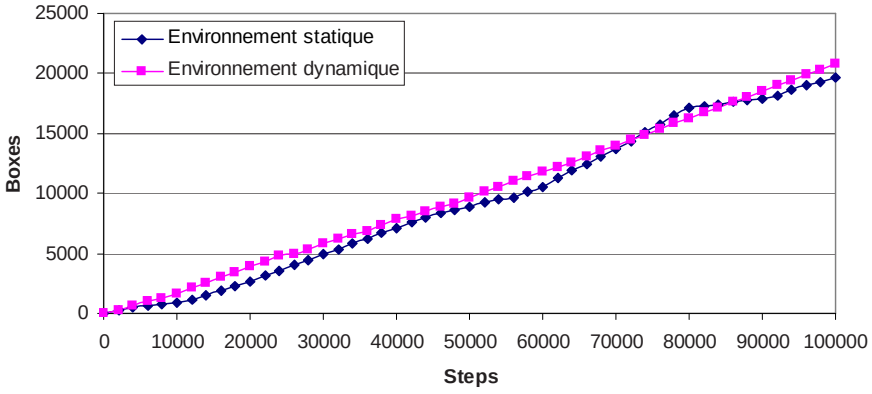

Figure 6.10 — Nombre de boîtes rapportées pour un environnement dynamique à quatre couloirs, pour 300 robots, comparé au nombre de boîtes rapportées dans un environnement statique à deux couloirs.

pour "effrayer", prévenir les robots proches d'un quelconque problème, ou bien échanger des requêtes ou des intentions. Troisièmement, le codage des comportements coopératifs est insensible au nombre de robots, à la topographie et aux dimensions de l'environnement, aux paramètres des balises près. Cependant, dans un environnement peu contraint (50 robots, par exemple), la nécessité de mettre en place une circulation est moins claire, car les robots se rencontrent rarement. Enfin, la méthode incrémentale proposée par ADELFE pour définir les situations non coopératives, nécessaires puis d'optimisation, ouvre la voie vers une méthode de conception vivante dans laquelle les robots se voient attribués des comportements au fur et à mesure de leur conception et de leur développement, en fonction des besoins du concepteur (cette activité est appelée *prototypage rapide* dans ADELFE). Ceci nécessitera bien sûr de se doter d'une plate-forme de simulation/conception adéquate au modèle d'agent coopératif.

Néanmoins, des choix ont été faits concernant l'affectation de valeurs pouvant changer radicalement le comportement global. En cela, ADELFE n'est d'aucune aide : c'est au concepteur de définir ces valeurs. Par exemple, le poids initial des balises et le facteur d'oubli ont été réglés sur le temps nécessaire à un robot pour faire un aller-retour. Ceci peut être totalement différent pour un environnement plus complexe avec, par exemple, plus ou moins de couloirs pouvant se fermer. Il est peut-être possible de faire apprendre ce genre de valeur aux robots en cours de simulation.

De plus, nous nous sommes focalisés, à partir de l'exemple du transport de ressources, sur des situations non coopératives particulières : conflit et inutilité. Si les robots étaient dotés de capacités de communication de haut niveau (pour échanger des données, comme des balises pour partager leurs expériences, par exemple), il faudrait gérer les problèmes d'incompréhension ou d'ambiguïté : ADELFE propose d'analyser des protocoles d'interaction entre agents et d'identifier de telles situations.

Enfin, l'exemple des robots est un exemple typique de système "à plat", dans lequel aucune hiérarchie n'est définie *a priori*. L'auto-organisation mène le collectif vers un fonctionnement adéquat sans prédéfinir d'organisation statique ; et ceci est en partie dû à l'homogénéité du collectif. Si par contre, le collectif d'agents est hétérogène (vitesses diverses, fonctions différentes, complémentaires ou non), les notions de hiérarchies et/ou priorités sont justifiées par ces différences. Afin de garantir les propriétés d'extensionnalité et d'irréductibilité des systèmes émergents (voir §1.1), il est proscrit de prédéfinir une organisation – auquel cas, la fonction du système est définie de manière intentionnelle, et donc non adaptative. Par conséquent, l'organisation est un phénomène émergent des relations entre agents et non un schéma prédéfini.

7 Analyse et Perspectives

ADELFE a été développée en quatre ans et a été expérimentée dans plusieurs phases d'analyse et conception de projets variés. Dans les chapitres 5 et 6, deux applications ont illustré son utilisation dans deux domaines différents : l'un étant orienté vers la résolution de contraintes avec utilisateurs et l'autre étant un problème classique de robotique collective. ADELFE a aussi été appliquée à des problèmes de conception mécanique dans SYNAMEC [Capera *et al.*, 2004a,b], de bio-informatique dans MICROMEGA [Macchion, 2004], ou d'inférence adaptative avec AdaptiveJRules [Petit, 2004].

Ces applications d'ADELFE ont toutes exhibé des avantages mais aussi des limites et faiblesses. Dans ce chapitre, nous allons tout d'abord exprimer ces analyses. Nous allons utiliser les même critères que ceux utilisés pour comparer les méthodes du chapitre 2 suivant quatre axes : les concepts, les notations, le processus et la pragmatique. Ceci nous permettra de placer ADELFE par rapport à l'état de l'art des méthodes orientées agent.

Outre les conclusions émises à la suite de l'analyse d'ADELFE, plusieurs perspectives sont envisageables. La première consiste en une proposition de glissement des phases dites "vivantes" du processus en direction du développement ; ce que nous appellerons le *Living Design* . La deuxième est une idée "récursive" dans le sens où nous proposons d'appliquer ADELFE à elle-même afin d'obtenir une méthode de développement adaptative : MétADELFE.

Analyse d'ADELFE

Comme pour les autres méthodes que nous avons présentées dans le chapitre 2, nous allons analyser ADELFE suivant les quatre axes de Shehory et Sturm ou Dam et Winikoff : concepts agents, notations/langages, processus et pragmatique [Shehory et Sturm, 2001; Dam et Winikoff, 2003]. Nous rapprocherons ensuite cette analyse de celle réalisée pour les autres méthodes.

Concepts et propriétés

ADELFE est dédiée à des systèmes multi-agents très spécifiques : auto-organisateurs par coopération. Ceci implique que certains concepts ne soient pas abordés explicitement, ou bien que leur manipulation ne soit pas immédiate.

Par exemple, les concepts de groupes ou d'organisations ne sont pas utilisés comme des concepts de modélisation, mais plutôt comme des observations, donc utilisés *a posteriori*. La notion de groupe dans un système auto-organisateur peut être une notion tellement dynamique qu'il est difficile, voire impossible, de les spécifier *a priori*. Par contre, si groupe il y a, il est possible qu'ils soient eux-mêmes définis comme des agents – comme des holons – afin de pallier la dynamique des interactions et des compositions. Par exemple, dans ETTO, nous ne parlons pas explicitement de groupes pour des partenariats entre BookingAgent. Les agents font forcément partie d'une organisation, par définition des AMAS ; mais ce terme n'est pas explicitement employé pour la modélisation. Dans le même ordre d'idée, les termes de norme (qui est souvent attaché à un groupe) et de tâche sont inexploités. Compte tenu de son éloignement de l'architecture BDI classique, les notions de désir ou d'intention sont aussi absentes de la modélisation proposée par ADELFE.

Les concepts manipulés par ADELFE sont, quant à eux, directement issus de ses fondements, les AMAS. En effet, l'*adaptation* du système est obtenue par *interactions* coopératives entre les agents *autonomes* ayant des *buts* locaux. Ces derniers sont intégrés dans le module de compétence des agents. De même, les *croyances* peuvent être vues comme des représentations – qui sont forcément locales et relatives. L'autonomie d'un agent est assurée par l'encapsulation des compétences et des représentations dans les modules de l'agent, ainsi que par l'accès privé aux décisions de l'agent qui garantit la pro-activité. La réactivité des agents est obtenue, comme pour PASSI ou MESSAGE, par l'utilisation de machines à états finis pour transcrire les comportements dynamiques et sociaux. Les interactions sont en effet prises en compte dans le module d'interaction – via les actions et les perceptions. Concernant les interactions de communication, l'emploi des protocoles A-UML est proposé. L'adaptation d'un système auto-organisateur peut aussi passer par l'ajout et la suppression d'agents, et donc par l'*ouverture* du système. L'ouverture n'est pas explicitement manipulée dans ADELFE mais reste néanmoins un concept clé. Elle est intégrée dans certaines applications comme AdaptiveJRules, par exemple, qui propose de faire un moteur d'inférence dans lequel les règles, dont le nombre varie, sont utilisées de manière coopérative [Petit, 2004]. Enfin, l'*environnement* est très clairement cité comme concept majeur. C'est grâce à la pression qu'il exerce sur le système, que les agents vont s'organiser et fournir la bonne fonction. Dans ADELFE, il apparaît explicitement lors des phases de besoins. A partir des

cas d'utilisation jusqu'aux classes, via les divers diagrammes d'interactions, la nature de l'environnement est prise en compte par les agents. Cependant, il reste encore des efforts de formalisation et de définition restent encore à faire.

Notations et langages

Comme MESSAGE, ADELFE préconise l'emploi d'UML et A-UML. Les agents sont vus comme des agents stéréotypés «cooperative agent» possédant des attributs et opérations stéréotypés en fonction des modules. L'utilisation de ces stéréotypes est réglée et en limite la mauvaise utilisation (en fonction de l'outil de conception utilisé, bien sûr).

UML et A-UML ont les désavantages suivants, dont hérite ADELFE : aucune gestion de la consistance ou de la vérification d'erreurs n'est possible, et ces notations manquent de précision – car ce sont des notations plus que des langages – lors de la modélisation et d'expressivité dans la représentation de raisonnements logiques. La prise en compte de la complexité ne passe que par la structuration en paquetages, modèles et sous-systèmes, mais n'est pas aussi efficace que dans Aalaadin, par exemple [Ferber et Gutknecht, 1998].

Les notations UML et A-UML sont très diffusées, annotées, et *accessibles*. Ces notations sont orientées objet et donc très *modulaires*. De plus, l'utilisation du prototypage rapide, notamment grâce à la transformation des protocoles en machines à états finis, fait de l'*exécutabilité* des comportements un point fort d'ADELFE. L'*analyse* des modèles peut se faire par l'utilisation d'OpenTool et de ces fonctionnalités de vérification à la volée, par exemple. Enfin, la proposition de transformation des protocoles rend plus accessible la notation A-UML, dans le sens où l'implémentation par machines à états concurrentes est préconisée.

Processus

Comme MESSAGE, ADELFE s'appuie sur le RUP. Cependant, en l'état, ADELFE n'intègre la notion d'agent que dans les travaux d'*analyse* et de *conception* – même si l'environnement est pris en compte dès les besoins. Par contre, la gestion des *tests*, via des scénarios issus des protocoles de communication, et donc simulables, est envisageable lors du prototypage rapide. L'analyse de ces tests est faite à partir des diagrammes de séquences générés en cours de simulation.

Comme nous avons pu le voir dans les chapitres précédents, le processus d'ADELFE a été défini de manière claire, en termes d'activités, d'étapes, de participants et de documents. Les activités et étapes sont ordonnancées et facilement identifiables. Dans chacune de ces applications, le suivi de cette démarche a toujours été respectée, même sans les créateurs aux commandes [Macchion, 2004; Petit, 2004]. En effet, le processus, inscrit dans le RUP, est assez connu des concepteurs (ici ingénieur ou ingénieur-maître) et ne demande pas trop de temps d'adaptation.

L'ordonnancement et le contenu des *délivrables* sont clairement définis, mais la gestion de la qualité est identique à celle proposée par le RUP et ne tient pas compte (mais est-ce possible ?) de la problématique agent. La *gestion de projet* et de suivi du processus est

facilité par l'utilisation d'AdelfeToolkit qui guide le concepteur tout au long du processus de développement. Enfin, comme le RUP, ADELFE est adéquate pour n'importe quel *contexte* de développement : création de logiciels, reverse engineering, prototypage ou réutilisation.

Pragmatique

Nous avons voulu ADELFE facile d'accès. Des *ressources* diverses ont été mises à la disposition des utilisateurs via un site web[1] décrivant le processus et l'exemple d'ETTO, et permettant l'accès aux logiciels d'ADELFE. *Aucun langage spécifique* n'est préconisé par ADELFE, bien que la plupart des applications d'ADELFE ait été codées en Java. ADELFE est une *méthode générique* dans le sens ou aucun domaine d'application n'est réellement privilégié. Cependant, il est possible que les AMAS ne soit pas nécessaires au développement d'une application donnée, vu les fondements systémiques de la méthode. Dans de telles circonstances, l'utilisateur est invité à se tourner vers une autre méthode de développement, plus adaptée.

Les systèmes multi-agents auto-organisateurs sont adéquats à *grande échelle* comme le prouve le logiciel de prévision de crues installé dans le bassin de la Garonne, STAFF [Sontheimer et Glize, 2000].

Cependant, les retours restent mitigés lorsque des utilisateurs, concepteurs d'ADELFE exclus, tentent d'appliquer ADELFE. Certes, le processus est bien compris, mais le manque de clarté des phénomènes auto-organisateurs à mettre en place reste un problème majeur.

Bilan et perspectives

La table 7.1 reprend la table 2.2 du paragraphe 2.5 en y ajoutant l'évaluation d'ADELFE (à droite). Suivant l'axe des notions méthodologiques (verticalement), le constat est clair : ADELFE n'est pas *LA* méthode multi-agent. En effet, l'architecture d'agent coopératif proposée implique une spécialisation vers des systèmes *adaptatifs* (par auto-organisation coopérative), *ouverts* (grâce à l'auto-organisation), dont les agents ont la propriété de *réactivité* (cycle de vie et machines à états concurrentes) et dont les *interactions* sont clairement modélisées en A-UML. Cependant, les autres concepts, souvent issus d'architectures spécifiques, comme BDI, sont mis à l'écart.

Concernant les notations, ADELFE est équivalente à MESSAGE avec une amélioration de l'*exécutabilité* grâce aux transformations de protocoles. Mais, ADELFE manque cruellement d'un langage de modélisation formel pour assurer précision et consistance.

Comme de nombreuses autres méthodes, ADELFE s'est focalisée sur les premières phases de développement logiciel : besoins, analyse et conception. Un développement futur devra se tourner vers les phases d'implémentation, de tests, de déploiement ou de maintenance, en s'intégrant, encore une fois dans le RUP, par exemple. De plus, nous avons fait un réel effort de modélisation du processus en SPEM, ce qui se traduit par une clarté de la définition des travaux, activités et délivrables. Cet effort se poursuit dans l'outil AdelfeToolkit

[1] `http://www.irit.fr/ADELFE`

		AAII	Aalaadin	Cassiopée	DESIRE	Gaia	MaSE	MASSIVE	MESSAGE	PASSI	Prometheus	Tropos	Voyelles	ADELFE
Concepts & Propriétés	*Adaptation*	-	+	+	-	-	-	±	±	±	-	-	+	++
	Autonomie	+	+	+	+	+	+	+	+	+	+	+	±	+
	Buts	+	±	-	+ +	±	+ +	+ +	+ +	±	+ +	+ +	±	+
	Croyances	+ +	±	-	+ +	±	+	±	-	+	+ +	+ +	+	+
	Désirs	+	±	-	+	±	+ +	+	+	+ +	+ +	+	+	-
	Environnement	- -	±	-	-	±	-	+	-	±	+	±	+	+
	Groupe	-	+ +	+	-	±	-	±	±	+	+	±	+	±
	Intention	+	±	-	+	±	±	±	±	±	+ +	+ +	+	-
	Interaction/Message	+	+ +	-	+	+	+ +	+ +	+ +	+ +	+ +	+	+ +	++
	Norme	+	+ +	+	+	+	+ +	+	±	-	±	+	+	±
	Organisation	+	+ +	+ +	- -	+ +	±	+	+ +	+	+	+	+ +	+
	Ouverture	- -	-	+	- -	- -	±	±	±	±	-	- -	+	++
	Pro activité	+	+	±	+	+	+	±	+	+	+	+	±	+
	Réactivité	+	+	+ +	+	+	-	±	+	+	±	+	±	++
	Rôle	-	+ +	+ +	+	+ +	+ +	+ +	+ +	+ +	-	+ +	+	+
	Tâche	-	±	±	+ +	+	+ +	+ +	+ +	+ +	+	±	+	±
Notations & langage de modélisation	*Accessibilité*	+	+	+	±	±	±	±	+	+	+	+	?	+
	Analysabilité	+	+ +	±	+	- -	+	+	+	+	+ +	+ +	?	+
	Complexité	+	+ +	±	+	- -	±	+	+	+	+	-	±	+
	Consistance	+	±	+	+	+	±	±	±	+	+ +	+ +	?	+
	Exécution	-	+	+	+	- -	+ +	+	+	+ +	+	+ +	+	++
	Expressivité	+	+	+	±	±	±	±	±	±	+	±	-	+
	Modularité	+ +	+	±	+	+ +	±	+	+ +	+ +	+	+	?	++
	Portabilité	+	+	+ +	- -	±	±	±	±	+	±	-	+	±
	Précision	+	+	±	+ +	+ +	+ +	-	-	±	+	+ +	- -	-
	Raffinage	+	+	+	+	+	+ +	+ +	+	+	+	+ +	?	+
	Traçabilité	+	+	+	+	+	-	±	+	+	+	±	?	+
Processus de développement	*Contexte*	Tous	Tous (Proto.)	Tous (Proto.)	Tous (Proto.)	Tous	Tous	Tous	Tous	Tous	Tous	Tous	Tous	Tous
	Besoins	-	-	- -	-	-	- -	+	+	+	-	+ +	-	+
	Analyse	+ +	+ +	+ +	±	+ +	+ +	+ +	+ +	+ +	+ +	+ +	+ +	++
	Conception	+	+	+	+ +	+ +	+ +	+ +	+ +	+ +	+ +	+	+ +	++
	Implémentation	- -	+ +	- -	±	- -	+	+	+	+	+	+	+	+
	Test	- -	-	- -	+ +	-	±	+	±	±	-	±	+	+
	Déploiement	- -	+	- -	- -	- -	- -	+ +	±	+ +	- -	- -	+	±
	Maintenance	- -	-	- -	- -	- -	- -	±	±	±	- -	- -	- -	±
	Délivrables	+	-	±	-	+ +	+ +	+	+ +	+ +	+	-	- -	++
	Gestion de la qualité	- -	- -	- -	- -	- -	- -	-	+	- -	- -	- -	- -	+
	Gestion de projet	- -	- -	- -	- -	- -	- -	-	+	- -	- -	- -	- -	++
Pragmatique	*Ressources*	-	+	±	-	-	+	+	+	+ +	-	-	- -	++
	Expertise requise	Oui	Non	Non	Oui	Oui	Non	Non	Non	Non	Oui	Oui	Non	Plutôt Oui
	Langage spécifique	Non	Oui	Non	Oui	Non	Non	Non	Non	Non	Non (Java)	Non	Non	Non
	Domaine d'application	Non	Simulation	Simulation	Connais-sances	Gros Grains	Non	Non	Non	FIPA	Utilisateur	Utilisateur	Non	Non
	Scalabilité	Oui	Oui	Non	Non	Oui	Non	Oui	Oui	Oui	Oui	Non	Oui	Oui

Notation : (++) pour les propriétés pleinement et explicitement prises en charge ; (+) pour les propriétés prises en charge de manière indirecte ; (±) pour des propriétés potentiellement prises en charge ; (−) pour des propriétés non prises en charge ; (−−) pour des propriétés explicitement non prises en charge.

Tableau 7.1 — Evaluation d'ADELFE et comparaison aux autres méthodes.

permettant de suivre un projet activité par activité.

Ce constat nous ouvre principalement trois perspectives : une formalisation des AMAS, une complétion du processus d'ADELFE et une redéfinition adaptative de la méthode (MétADELFE).

Vers plus de formalisme

Un des principaux points faibles est le manque de fondements théoriques formels sur lesquels ADELFE peut se reposer pour aider à une conception sûre en terme de spécification des comportements des agents et donc du système. La théorie des AMAS est une théorie des systèmes générale fondée sur l'observation de phénomènes naturels et sociaux. La formalisation d'une telle théorie peut donc passer par l'étude *a posteriori* de systèmes coopératifs, comme dans les sciences naturelles, ou bien par l'étude de la conception des systèmes coopératifs artificiels. Ces études sont bien sûr les directions privilégiées et sont illustrées par les différents travaux sur les AMAS que nous avons survolés dans le paragraphe 1.4. Chacune de ces études explore une vue possible sur les AMAS, pour tenter d'en extraire des règles ou des propriétés à partir des problèmes rencontrés et des solutions proposées. Par exemple, la thèse de Georgé portait sur l'idée de programmation émergente, mais s'est aussi attachée, à un niveau plus théorique, à classifier les types de systèmes auto-organisateurs émergents [Georgé, 2004].

Outre les tous premiers travaux sur la théorie des AMAS [Camps *et al.*, 1997], quelques tentatives de formalisation des AMAS ont essayé de répondre à ces manques [Vareilles, 2002]. Vareilles a proposé une formalisation en B des agents coopératifs, mais sans réel succès.

Actuellement, des travaux de l'équipe SMAC envisagent l'utilisation des automates formels au croisement entre les algèbres de processus et la programmation par contraintes. L'objectif de ces travaux, côté multi-agent, est d'établir un modèle général de système multi-agent et d'environnement puis de le particulariser afin d'étudier des propriétés de ceux qui sont à fonctionnalité émergente fondée sur l'auto-organisation coopérative. Les deux questions de base qui se posent sont :

- Est-il possible de définir les propriétés que les agents doivent posséder pour que le système multi-agent satisfasse une propriété globale particulière ?
- Si les agents satisfont une certaine propriété, que peut-on en déduire au niveau global ?

Le modèle proposé est une forme de produit d'automates qui étend à la fois les automates communicants, les automates cellulaires et les systèmes de réétiquetage de graphes.

Vers un processus plus complet et vivant : le *Living Design*

Un autre constat majeur est l'incomplétude du processus. Bien sûr une des perspectives à court terme – et nécessaire – pour ADELFE est de *poursuivre le développement du processus vers les phases tardives de développement*. Concernant l'implémentation, il semblera nécessaire de se pencher vers la transformation de modèles, de type *Model Driven Architecture* (ou MDA), afin de générer des squelettes de développement fidèles aux modèles de conception [Soley et the OMG Staff Strategy Group, 2002]. Pour les tests, l'activité de prototypage rapide

semble être une bonne base de travail. Côté déploiement, compte tenu de la possibilité de passage à l'échelle des AMAS, les considérations de déploiement semblent similaires à celles proposées par le RUP, mise à part la possibilité de modifier le comportement des agents en vie dans le système déployé pour assurer la maintenance. Dans un article, nous avons proposé le concept de *Living Design* [Georgé *et al.*, 2003c] afin de répondre aux problèmes de conception de systèmes ouverts.

Glissement vers les phases "vivantes" de développement

Dans ce paragraphe, une nouvelle approche à la conception de systèmes est proposée – reposant sur les expériences des biologistes [Maturana et Varela, 1994] – compte tenu de l'insuffisance actuelle des méthodes de conception de systèmes artificiels pour appréhender les systèmes ouverts, fortement dynamiques et à grande échelle. Ce paragraphe propose des idées pour de futurs travaux.

Le fossé entre méthodes actuelles et méthodes requises. La tendance actuelle de modélisation et de spécification de logiciels est d'utiliser le paradigme objet et les méthodes et notations associées. En considérant le concept d'agent comme un moyen de concevoir des systèmes artificiels tels que des systèmes biologiques, on peut admettre un large fossé entre les méthodes de conception classiques et la conception de systèmes ouverts, de la même manière que le fossé existant entre agents et objets [Odell, 2002]. Les différences fondamentales sont :

- il n'y a plus de contrôle sur le système complet en cours d'exécution ;
- toutes les situations auxquelles le système peut faire face ne peuvent être spécifiées ;
- la fonction du système peut évoluer en cours de fonctionnement par rapport aux besoins initialement exprimés ;
- la taille du système n'est pas constante : des composants peuvent apparaître ou disparaître.

Les méthodes pour concevoir de tels systèmes devront donc être différentes des existantes. Avec l'accroissement en complexité, la large distribution et la dynamique environnementale des systèmes, les méthodes d'ingénierie logicielle classiques ne sont pas suffisantes. Les spécifications ne sont pas complètes. Le système doit évoluer car il est ouvert, et le contrôle est distribué. Toutefois, de telles considérations nécessitent de fournir des aides méthodologiques, des méthodes et des outils associés.

Décalage de processus objet-agent par chevauchement Une possibilité pour appliquer ces idées peut être d'utiliser et de décaler les processus de développement en termes de concepts d'abstraction : objet et agent. Nous appelons cela un décalage de processus objet-agent par chevauchement.

Un processus à deux couches. Actuellement, les méthodes et processus connus pour modéliser des systèmes multi-agents (ADELFE en fait maintenant partie) soit manipulent de nouveaux formalismes soit réutilisent des formalismes objets, comme UML, et les processus associés, comme le RUP (voir chapitre 2). Un premier pas vers une conception plus vi-

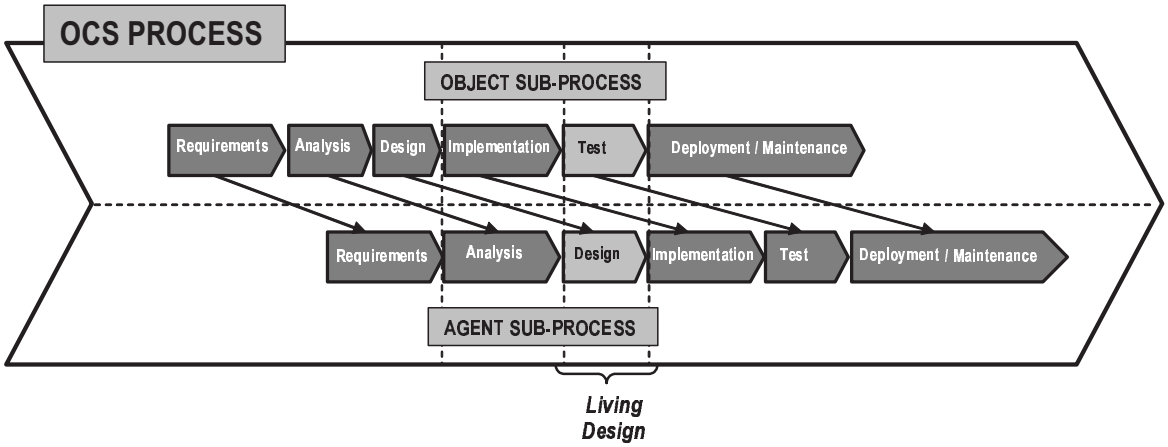

Figure 7.1 — Décalage de processus objet-agent par chevauchement pour les systèmes artificiels ouverts (*Open Computational Systems* ou OCS).

vante serait de manipuler les deux paradigmes, objet et agent, dans un même processus à deux couches : un processus objet-agent par chevauchement. Toutefois, même si les concepts d'objet et d'agent peuvent être exprimés dans le même langage, il est clair qu'il nécessitent des vues différentes. Une solution pour lier les deux couches d'abstraction pourrait être d'utiliser en parallèle deux processus tels que le processus orienté agent soit décalé vers les phases tardives du processus orienté objet (tests, déploiement et maintenance), appelées les "phases vivantes" – c.-à-d. celles durant lesquelles le système "vit". Pour résumer, un processus de développement de systèmes ouverts, comme des AMAS, serait composé de deux sous-processus : un pour les agents, un pour les objets.

Décaler les processus vers les phases "vivantes". La figure 7.1 montre les deux sous-processus décalés. Le sous-processus agent est décalé vers les phases vivantes du processus objet. Nous allons uniquement nous focaliser sur la *conception vivante*, ou *Living Design*, indiqué en gris clair sur la figure 7.1.Le lien entre les deux phases signifie que concevoir des agents correspond à la phase de test dans le sous-processus objet. Brièvement, compte tenu des autres phases, les besoins pour le processus agent sont considérés par le concepteur en même temps que les trois premières phases du processus objet (besoins, analyse et conception). Ce décalage entre les deux sous-processus apparaît dans ADELFE lors de la vérification de l'adéquation des AMAS, par exemple (voir 3.5.2). A la fin du processus, les phases d'implémentation, de tests, de déploiement et de maintenance du processus agent sont considérées en même temps que les phases de déploiement et de maintenance du processus objet. De plus, la phase de conception n'est pas entièrement sous la responsabilité du concepteur, comme en conception classique, mais ici le système doit déjà avoir des capacités à réagir aux situations non pré-définies dans les spécifications. Par conséquent, les systèmes pourraient devenir plus robustes, plus autonomes et plus complexes. Ils pourraient s'auto-modifier, s'auto-réparer ou travailler dans des conditions dégradées.

Les pré-requis pour le Living Design. Le *Living Design* est défini par le lien entre les phases de conception et de tests des deux sous-processus et signifie "construire des agents

en cours de fonctionnement". Ainsi, le concepteur est comme un biologiste étudiant le comportement de créatures vivantes et modifiant ces modèles en accord avec ses observations. Cela implique l'existence d'outils pour modéliser et simuler les agents incomplets dont il manquerait des compétences, des aptitudes, des actions, ou d'autres composantes. De plus, nous pourrions envisager une sorte de *Living Design* durant les phases de déploiement, i.e. le système évolue et est en construction dans son environnement cible. Bien sûr, il semble difficile dès à présent, pour des utilisateurs finals, d'accepter que leur logiciel soit incomplets et ne puisse pas répondre à leurs attentes.

Le prototypage rapide "vivant"

Comme il vient d'être précisé, concevoir des systèmes devant évoluer dans des environnements difficiles nécessite des outils pour tester les composantes en cours d'exécution. Dans ce cadre, une fonctionnalité avancée, comme la simulation dans OpenTool, pourrait être une base de départ. En effet, en dotant OpenTool d'interface de modification du code des agents en direct (ce qui est possible), les conséquences des modifications des comportements des agents en cours de simulation pourraient être observées en direct. Cependant, OpenTool manque d'interfaces graphiques évoluées et adaptées. Il est peut être alors possible de se tourner vers des outils de simulation plus évolués, comme oRis[2] qui propose un langage de programmation dynamique que nous avons utilisé pour développer des systèmes de simulation robotique [Harrouet, 2000; Capera, 2001; Picard, 2001].

Idéalement, le prototypage rapide devrait permettre de compléter le comportement des agents (en leur ajoutant des règles de résolution de situations non coopératives par exemple) par observation de leurs erreurs. Le système serait alors vu comme une entité apprenant sa fonctionnalité en cours de fonctionnement et de déploiement.

Un pas plus loin : le *Coached Living Design*

Considérons maintenant que nous avons un système complexe particulier à développer. Nous sommes arrivés à identifier des entités de haut niveau comme étant les agents, mais nous sommes incapables de décrire leurs comportements ou leurs compétences complètement ou formellement. Très naturellement, dans une approche *top-down* classique appliquée à la résolution de problèmes multi-agents, nous allons descendre d'un niveau et décrire un agent comme étant un système multi-agent (comme le propose ADELFE par exemple). Les agents de nouveau système multi-agent sont plus simples mais, grâce à la coopération, ils seront capables d'exécuter la fonction attendue de l'agent de niveau supérieur qu'ils composent.

Mais les agents de niveau inférieur peuvent être trop difficiles à coder. Et ainsi, nous réitérons le pas précédent pour définir un autre niveau. Au bout d'un certain nombre d'itérations, nous retombons alors sur le niveau d'abstraction le plus bas : les instructions d'un langage de programmation. L'expressivité est maximale, mais leur organisation est très difficile à déterminer : faisons-la s'auto-organiser par coopération. Nous arrivons à l'idée de programmation émergente de Georgé [Georgé, 2004].

[2] `http://www.enib.fr/~harrouet/`

Nous avons alors un système multi-agent capable de simuler un programme classique. Comme nous voulons construire un système artificiel ouvert, les agents doivent s'auto-organiser pour trouver les organisations adéquates à leur environnement. Nous souhaitons un système capable d'apprendre le bon comportement grâce à la pression environnementale, et avons donc besoin d'agents coopératifs.

Bien sûr, nous ne pouvons coder la fonction du système directement dans les agents, par définition du système que nous voulons produire, mais nous pouvons influencer le collectif d'agents pour atteindre la fonction adéquate, comme un entraîneur (*coach*) peut le faire avec son équipe. Le concepteur est alors vu comme l'entraîneur de ces agents : nous parlons de *Coached Living Design*.

Ce type de conception est nécessaire pour guider le processus d'adaptation des composants lorsqu'ils sont codés en utilisant la programmation émergente. Néanmoins, cela peut être encore pertinent pour des niveaux d'abstraction plus élevés lorsque le concepteur est incapable de spécifier l'organisation de ces agents – comme il aurait pu être incapable de trouver le code pour obtenir une fonction donnée au niveau des instructions. Bien sûr, ces considérations sont à très long terme...

Vers une méthode plus adaptative : MétADELFE

Voici une perspective, encore à long terme, qui peut représenter le futur d'ADELFE : MétADELFE. L'idée est de produire une méthode de développement comme un système artificiel adaptatif en fonction des besoins des concepteurs. Les fonctions d'une méthode sont peu claires et incomplètes. ADELFE ne serait-elle pas adéquate pour résoudre ce problème ?

Le processus décrit dans ADELFE est relativement simple – en partie parce qu'il ne traite pas des travaux de tests, implémentation, etc. Cependant, le RUP, par exemple, est une méthode assez complexe dans sa construction, et quelque fois assez rigide. Comme nous avons pu le voir dans le chapitre 3, un processus peut être modélisé comme un ordonnancement de travaux, d'activités ou d'étapes produisant et consommant des documents ou des modèles. Une représentation possible est l'utilisation des diagrammes d'activités, faisant apparaître activités, produits et participants, qui représentent une organisation statique du processus. Pourquoi une telle organisation ne pourrait-elle pas être adaptative ? – dans le sens où les activités et les documents produits s'organisent dans le temps afin de répondre au mieux aux besoins des participants (analystes, concepteurs, etc.).

Le SPEM offre une bonne analyse du domaine méthodologique et propose une classification et un ensemble de mots-clés pouvant servir de point de départ à une analyse multi-agent, comme avec ADELFE par exemple.

Une méthode doit fournir un système construit suivant un processus particulier, grâce à des modèles manipulés par des outils dédiés. Comme pour ETTO, nous pouvons envisager un système de collaboration pour construire des systèmes. Les utilisateurs d'un tel système sont les participants, qui seront représentés par des agents. L'ordonnancement et la production des activités sera gérée par des agents de niveau inférieur, correspondant aux activités et états, qui devront trouver leur place dans l'échelle du temps et des productions en fonction des contraintes des utilisateurs – dates limites, niveau de détail, besoins d'outils

spécifiques, etc.

Une telle méthode adaptative peut s'incarner dans un outil comme AdelfeToolkit, plus collaboratif qu'actuellement et ayant des interactions plus riches avec les utilisateurs. Cependant, le cœur du problème réside dans la définition des règles locales de réorganisation à deux niveaux. Au niveau des agents représentant les participants, quelles sont les règles de coopération nécessaires ? Une étude plus approfondie du fonctionnement d'équipes de travail en entreprise pourrait peut-être y répondre. Au niveau des agents de bas niveau (les activités et les étapes), SPEM fournissant des schémas d'ordonnancement, des gardes, des flux de données précis, une analyse plus poussée pourrait faire apparaître des règles de coopération simples issues de ces définitions.

Conclusion

TOUT au long de ce travail, mon objectif a été de fournir les méthodes et les outils de développement de systèmes à fonctionnalité émergente. Des exemples d'applications ont été présentés afin d'illustrer mon propos. Ces travaux devaient connecter deux domaines particuliers : les systèmes émergents et l'ingénierie logicielle. Certains choix fondamentaux ont dû être pris pour accomplir cette tâche de diffusion et de compréhension.

Les choix scientifiques

Le premier choix est celui du cadre systémique adopté par ADELFE : les *AMAS*. Afin de construire des systèmes émergents, nous avons vu que la technique de l'auto-organisation offrait un cadre simple et reproductible dans les systèmes artificiels. Pour rendre l'auto-organisation implantable, il nous fallait trouver un critère de réorganisation pertinent : la *coopération*. La théorie des AMAS se posa donc comme base de réflexion pour une conception de systèmes émergents. En effet, dans ce cadre, pour construire un système émergent adéquat, il suffit de le peupler d'agents coopératifs. Ceci nécessite alors de définir clairement les règles de coopération et de réorganisation du collectif ; ce qui est le point critique de cette approche : comment trouver les situations non coopératives ? Ce choix possède ses avantages et ses inconvénients. Les inconvénients sont directement liés au manque de formalisation et de définition de ce qu'est la coopération. Elle reste, peut-être, à un trop haut niveau d'abstraction (on est loin d'une fonction de mesure de la coopération...). Il reste donc difficile de résoudre le problème de l'exhaustivité des règles de coopération afin de garantir l'adéquation fonctionnelle du système. Le principal avantage de l'approche par AMAS est certainement la facilité d'analyse pré-modélisation due, en partie, à son analogie aux comportements sociaux naturels.

Le deuxième choix est celui du cadre méthodologique : l'ingénierie orientée objet. Ce choix est majeur, et en mesurer les conséquences n'est pas évident. En effet, nous sommes partis de l'hypothèse qu'un agent (coopératif dans notre cas) pouvait être modélisé comme un objet, avec une sémantique particulière. Ce rapprochement n'est pas une nouveauté, de nombreux travaux antérieurs – mais pas tous – le tiennent comme acquis. Il est vrai que lorsqu'il faut coder, le choix de l'objet vient immédiatement à l'esprit, compte tenu des similitudes des propriétés des objets et des agents. Seulement, le gros point noir de ce choix vient du manque de définition claire du concept d'agent – contrairement au concept objet. Aucun consensus, malgré les efforts et les nombreuses manifestations orientées multi-agent, n'existe. En effet, les notions d'autonomie ou de capacité de raisonnement sont souvent trop floues pour être "correctement" exploitables.

Le troisième choix est directement lié au choix de conception objet : le RUP. Ce choix est

certainement celui qui porte le moins à conséquence, du moins du point de vue scientifique. En effet, ce choix a été judicieux pour assurer la bonne compréhension de la marche à suivre par des novices, compte tenu de la clarté du RUP. Les systèmes multi-agents restant des systèmes artificiels et/ou logiciels, ce genre de processus est tout à fait adapté, d'autant plus que nous avions fait le choix de l'objet comme paradigme de modélisation des agents.

Bilan et apport méthodologique

Mon travail sur ADELFE, et plus largement sur les méthodes multi-agents, a plusieurs retombées méthodologiques. Tout d'abord, la définition du processus ADELFE a été effectuée de manière claire en utilisant le SPEM. Ce travail a conduit à clarifier les concepts multi-agents nécessaires à la modélisation d'AMAS ; ce qui se traduit par la définition d'un glossaire dans ADELFE et la définition de stéréotypes UML et de règles de cohérence associées.

Le travail sur le SPEM était une première utilisation de cette notation. Aucun guide n'était réellement disponible et nous avons dû extraire, des premières spécifications du SPEM les règles de modélisation de processus. Ayant réalisé l'apport de son utilisation à la diffusion des nouvelles méthodes orientées objet, SPEM a été diffusé au sein de la FIPA afin de comparer les différentes approches en cours.

Un autre apport de ce travail est la réflexion sur la place qu'occupe A-UML dans la conception de systèmes multi-agents adaptatifs et l'intérêt de son utilisation à des fins de prototypage et de validation comportementale. En effet, une extension à A-UML a été développée afin de prendre en charge la coopération dans les interactions entre agents (à la réception de message). De plus, la transformation automatique de ces protocoles en machines à états finis permet d'ajouter à la notations A-UML une dimension d'exécutabilité jusqu'alors absente des propositions.

Enfin, le dernier apport est celui des outils associés à ADELFE. OpenTool a été étendu pour prendre en compte les modifications faites à UML et A-UML. AdelfeToolkit a été développé pour faciliter le suivi du processus en se basant sur une expression SPEM du processus et en illustrant chacune des activités et étapes par l'exemple d'ETTO. Bien sûr, ce dernier outil reste un prototype, mais valide le concept d'outil interactif de suivi de projet.

Cependant tout n'est pas arrêté et complet. De nombreuses perspectives s'ouvrent dans le sens du développement d'ADELFE, comme il a été souligné dans le chapitre 7. Le processus reste incomplet dans la couverture du cycle de vie logiciel. En effet, les phases préliminaires, d'analyse et de conception sont bien définies, mais les phases de développement, de tests/validations, de déploiement et de maintenance restent, pour l'instant identiques à celle du RUP. La problématique agent trouvera certainement sa place dans ces étapes, et requerra certainement des extensions plus adéquates et spécifiques. Il en est de même pour les notations qui restent limitées pour manipuler de larges collectifs ouverts ou pour vérifier la cohérence des modèles.

Bilan et apport aux AMAS

Comme tous les travaux se basant et utilisant les AMAS comme modèle de conception de systèmes artificiels, mon travail a permis de clarifier certains de leurs concepts. La réflexion sur le modèle d'agent coopératif et sur les stéréotypes a permis de présenter le fonctionnement des agents coopératifs de manière simple et modulaire. Comme du point de vue méthodologique, se pencher sur les définitions des termes ou sur les propriétés des systèmes émergents, nous a permis de clarifier certains points et de rendre plus clair et plus précis notre discours sur les AMAS – afin d'éviter les ambiguïtés sur les termes comme *coopération*, par exemple.

Mon travail propose d'utiliser la notion d'agent coopératif, propre aux AMAS, comme une extension des objets classiques grâce à l'utilisation de stéréotypes UML. Ceci n'est certainement pas la seule possibilité. Ici encore, les champs de perspectives sont larges. Concernant la modélisation de la dynamique ou de l'ouverture des AMAS, l'utilisation de langages plus fonctionnels semblerait adéquate et permettrait ainsi de garantir la propriété d'extensionnalité de la fonctionnalité d'un système émergent, par exemple. Concernant la systémique et la caractérisation des interactions inter- et intra-système, l'utilisation de notations de type automates ou produits d'automates est aussi une piste prometteuse. Enfin, concernant la définition exhaustive des types de situations non coopératives, une étude plus formelles – fonctionnelle ou ensembliste, par exemple – serait certainement envisageable. En effet, le point à éclaircir en priorité est l'exhaustivité des types de SNC mais aussi l'exhaustivité des SNC relatives à un agent. Est-il possible de répondre de manière sûre aux questions "*quelles sont les situations non coopératives de mes agents*" ou "*ai-je trouvé toutes les situations non coopératives de mes agents*" ?

Des exemples illustratifs et prometteurs

Ce travail et la pertinence de la méthode ADELFE ont été éprouvés grâce à ETTO et les robots transporteurs. Dans le premier cas, il a fallu développer un système de gestion coopérative des contraintes d'un emploi du temps. Le prototype réalisé montre des résultats très intéressants, tant au niveau méthodologique (identification des agents, des SNC, etc.), qu'au niveau fonctionnel (rapidité de résolution) ou non fonctionnel (ergonomie et pertinence d'un tel outil partagé). De plus, cet exemple a été intégré à ADELFE en tant qu'outil didactique. ETTO reste cependant un simple prototype, et le comportement des agents n'est pas encore complet – notamment, l'exploration des salles est aléatoire. Cette application représente l'une de mes perspectives de développement à très court terme.

La simulation robotique est une reprise de mon étude de DEA [Picard, 2001; Picard et Gleizes, 2002]. Le problème a été repris de zéro, car ADELFE n'existait pas à l'époque. Le résultat est différent, et cadre plus à la définition de systèmes émergents. En effet, dans la première version, l'architecture d'agent était totalement différente, et basée sur des graphes de comportements récursifs. Les robots déterminaient leur comportement par la construction de graphes, dont l'un des nœuds était, notamment, *traverser couloir X*. Ceci signifie que les robots avaient "conscience" de l'existence des couloirs. La solution actuelle, présentée dans le chapitre 6, est plus conforme aux directives issues de la définition de l'émergence : les ro-

bots n'ont aucune connaissance des couloirs. Pour eux le monde n'est que robots, boîtes, obstacles ou points d'attraction de retrait ou de dépôt. Les résultats des simulations montrent alors de meilleurs résultats en terme d'efficacité et d'apparition de sens de circulation, alors que dans la première version, tous les couloirs restaient fréquentés dans les deux sens, mais avec des préférences. Comme ETTO, cette application est une bonne illustration de ce qu'est un système émergent, même si tous les types de SNC (connus) ne sont pas manipulés – à cause de l'absence de communication directe entre agents, notamment.

Une évolution naturelle

Après avoir simulé des comportements collectifs d'agrégation de cadavres chez les fourmis en Maîtrise et après avoir étudié l'émergence comportementale dans des collectifs de robots en DEA, je me suis naturellement posé des questions sur la spécification et le développement de tels systèmes, ainsi que leur explication aux novices – *niveau méta*. ADELFE constitue l' une des réponses possibles. Cette réponse reste cependant, mais heureusement, ouverte et incomplète, laissant de nombreuses perspectives. ADELFE a elle-même soulevé des questions concernant la conception de méthodes, avec le SPEM ou l'idée MétADELFE – *niveau méta méta*.

Cependant, toutes les bonnes choses ont une fin, et cette évolution ne fait pas exception. Ces dernières années m'ont aussi conforté dans l'intérêt et les potentialités des théories de l'émergence pour les systèmes artificiels. C'est pourquoi, avant de continuer, ou de préciser, les travaux de cette thèse, il me semble nécessaire de revenir aux sources, en étudiant, notamment les systèmes émergents de manière plus analytique ; ce qui représente mes perspectives de recherches futures.

Bibliographie

S.M. ALI, R.M. ZIMMER et C.M. ELSTOB : The question concerning emergence : Implication for Artificiality. *Dans* DUBOIS, D.M., éditeur : *Computing Anticipatory Systems : CASYS'97 - First International Conference*, 1997.

F. ARLABOSSE, M.-P. GLEIZES et M. OCCELLO : Méthodes de Conception. *Dans Systèmes Multi-Agents*, volume 29 de *Arago*, pages 137–171. Editions Tech & Doc, 2004.

P. BALL : *The Self-Made Tapestry*. Oxford University Press, 1998.

C. BERNON, V. CAMPS, M.-P. GLEIZES et G. PICARD : Designing Agents' Behaviors and Interactions within the Framework of ADELFE Methodology. *Dans* A. OMICINI, P. PETTA et J. PITT, éditeurs : *Fourth International Workshop on Engineering Societies in the Agents World (ESAW'03)*, volume 3071 de *Lecture Notes in Computer Science (LNCS)*, pages 311–327. Springer-Verlag, 2003a.

C. BERNON, M. COSSENTINO, M.-P. GLEIZES, P. TURCI et F. ZAMBONELLI : A Study of Some Multi-Agent Meta-Models. *Dans* P. GIORGINI, J. P. MÜLLER et J. ODELL, éditeurs : *Agent-Oriented Software Engineering IV, 5th International Workshop, AOSE 2004, New-York, USA, July 19, 2003*, pages 113–130, 2003b.

C. BERNON, M.-P. GLEIZES, S. PEYRUQUEOU et G. PICARD : ADELFE : a Methodology for Adaptive Multi-Agent Systems Engineering. *Dans* P. PETTA, R. TOLKSDORF et F. ZAMBONELLI, éditeurs : *Third International Workshop on Engineering Societies in the Agents World (ESAW'02)*, volume 2577 de *Lecture Notes in Computer Science (LNCS)*, pages 156–169. Springer-Verlag, 2002a.

C. BERNON, M.-P. GLEIZES, G. PICARD et P. GLIZE : The ADELFE Methodology For an Intranet System Design. *Dans* P. GIORGINI, Y. LESPÉRANCE, G. WAGNER et E. YU, éditeurs : *Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002)*, volume 57, Toronto, Canada, May 2002b. CAiSE'02, CEUR Workshop Proceedings.

O. BOISSIER et Y. DEMAZEAU : ASIC : An Architecture for Social and Individual Control and its Application to Computer Vision. *Dans* J.W. PERRAM et J.-P. MÜLLER, éditeurs : *Distributed Software Agents and Applications, 6th European Workshop on Modelling Autonomous Agents, MAAMAW '94, Odense, Denmark, August 3-5, 1994, Proceedings*, volume 1069 de *Lecture Notes in Computer Science (LNCS)*, pages 135–149. Springer-Verlag, 1996.

O. BOISSIER, S. GITTON et P. GLIZE : Caractéristiques des systèmes et des applications. *Dans Systèmes Multi-Agents*, volume 29 de *Arago*, pages 25–54. Editions Tech & Doc, 2004.

E. BONABEAU, M. DORIGO et G. THERAULAZ : *Swarm Intelligence : From Natural to Artificial Systems*. Oxford University Press, 1999.

E. BONABEAU, G. THERAULAZ, J.-L. DENEUBOURG, S. ARON et S. CAMAZINE : Self-organization in social insects. *Trends in Ecology and Evolution*, 12:188–193, 1997.

G. BOOCH : *Conception orientée objets et applications*. Addison-Wesley, 1992.

C. BOURJOT, V. CHEVRIER et V. THOMAS : How Social Spiders Inspired an Approach to Region Detection. *Dans Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 426–433. ACM Press, 2002.

F. BRAZIER, C. JONKER et J. TREUR : Compositional Design and Reuse of a Generic Agent Model. *International Journal of Cooperative Information Systems*, 9(3):171–207, 2000.

F.M.T. BRAZIER, B.M. DUNIN-KEPLICZ, N.R. JENNINGS et J. TREUR : DESIRE : Modelling Multi-Agent Systems in a Compositional Formal Framework. *International Journal of Cooperative Information Systems*, 6(1):64–94, 1997.

F.M.T. BRAZIER, C.M. JONKER et J. TREUR : Principles of Compositional Multi-Agent System Development. *Dans* J. CUENA, éditeur : *Proceeding of the IFIP'98 Conference IT&KNOWS'98*. Chapman & Hall, 1998.

E. K. BURKE et S. PETROVIC : Recent Research Directions in Automated Timetabling. *European Journal of Operational Research (EJOR)*, 140(2):266–280, 2002.

G. CAIRE, W. COULIER, F. GARIJO, J. GOMEZ, J. PAVON, F. LEAL, P. CHAINHO, P. KEARNEY, J. STARK, R. EVANS et P. MASSONET : Agent Oriented Analysis Using Message/UML. *Dans* M. WOOLDRIDGE, G. WEISS et P. CIANCARINI, éditeurs : *Agent-Oriented Software Engineering II, Second International Workshop, AOSE 2001*, volume 2222 de *Lecture Notes in Computer Science (LNCS)*, pages 119–135. Springer-Verlag, 2001.

V. CAMPS, B. CARPUAT, M.-P. GLEIZES, P. GLIZE, A. MACHONIN, J. PEZET, C. RÉGIS et S. TROUILHET : Une théorie de la coopération pour l'auto-organisation des systèmes artificiels. Rapport interne IRIT/97-58-R, Institut de Recherche en Informatique de Toulouse (IRIT), 1997.

V. CAMPS, M-P. GLEIZES et P. GLIZE : Une théorie des phénomènes globaux fondée sur des interactions locales. *Dans* J-P. BARTHÈS, V. CHEVRIER et C. BRASSAC, éditeurs : *Systèmes multi-agents – de l'interaction à la socialité – Actes des 6èmes JFIADSMA*, pages 207–220. Editions Hermès, 1998.

D. CAPERA : Systèmes multi-agents adaptatifs et contrôle robotique. Mémoire de DEA Représentation de la connaissance et formalisation du raisonnement, Université Paul Sabatier - Toulouse III, 2001.

D. CAPERA, M-P. GLEIZES et P. GLIZE : Mechanism Type Synthesis based on Self-Assembling Agents. *Journal on Applied Artificial Intelligence*, 18(8–9), 2004a.

D. CAPERA, G. PICARD, M.-P. GLEIZES et P. GLIZE : Applying ADELFE Methodology to a Mechanism Design Problem. *Dans Third Joint Conference on Multi-Agent System (AAMAS'04)*, New York, USA, July 2004b. IEEE Computer Society.

A. CARDON et Z. GUESSOUM : Systèmes multi-agents adaptatifs. *Dans* S. PESTY et C. SAYETTAT-FAU, éditeurs : *Systèmes multi-agents – Méthodologie, technologie et expériences – Actes des 8èmes JFIADSMA*, pages 101–116, Saint-Jean-la-Vêtre, Loire, France, Octobre 2000. Hermès.

C. CASTELFRANCHI : Through the Minds of the Agents. *Journal of Artificial Societies and Social Simulation*, 1(1):1–6, 1998.

J. CASTRO, M. KOLP et J. MYLOPOULOS : A Requirements-driven Development Methodology. *Dans* K.R. DITTRICH, A. GEPPERT et M.C. NORRIE, éditeurs : *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01)*, volume 2068 de *Lecture Notes in Computer Science (LNCS)*, pages 108–123. Springer-Verlag, 2001.

L. CERNUZZI et G. ROSSI : On The Evaluation Of Agent Oriented Methodologies. *Dans* J. DEBENHAM, B. HENDERSON-SELLERS, N. JENNINGS et J. ODELL, éditeurs : *Proceedings of the OOPSLA 02 - Workshop on Agent-Oriented Methodologies*, pages 21–30. COTAR, 2002.

A. COLLINOT et A. DROGOUL : Agent Oriented Design of a Soccer Robot Team. *Dans* M. TOKORO, éditeur : *Second International Conference on Multi-Agent Systems (ICMAS'96), Nara, Japan*, pages 41–47. AAAI Press, 1996.

A. COLLINOT et A. DROGOUL : La méthode de conception multi-agent CASSIOPEE : application à la robotique collective. *Revue d'intelligence artificielle*, 12(1):125–147, 1998.

A. COLORNI, M. DORIGO et V. MANIEZZO : Distributed Optimization by Ant Colonies. *Dans* F. VARELA et P. BOURGINE, éditeurs : *Proceedings of the First European Conference on Artificial Life, Paris, France*, pages 134–142. Elsevier Publishing, 1992.

C. CONTE : Emergent Functionality among Intelligent Systems : Cooperation within and without Minds. *AI and Society*, 6:78–93, 1992.

M. COSSENTINO : Different Perspectives in Designing Multi-Agent System. *Dans Designing Multi-Agent System, AgeS'02 (Agent Technology and Software Engineering) Workshop at NodE'02*, 2001.

M. COSSENTINO et C. POTTS : A CASE tool supported methodology for the design of multi-agent systems. *Dans The Proceedings of the International Conference on Software Engineering Research and Practice (SERP'02)*, 2002.

M. COSSENTINO, L. SABATUCCI et V. SEIDITA : SPEM description of the PASSI process. Rapport technique RT-ICAR-20-03, ICAR-CNR, 2003.

W. COULIER, F. GARIJO, J. GOMEZ, J. PAVON, P. KEARNEY et P. MASSONET : *MESSAGE : a methodology for the development of agent-based applications*. Kluwer, 2004.

J-L. T. da SILVA et Y. DEMAZEAU : Vowels Co-ordination Model. *Dans* M. GINI, T. ISHIDA, C. CASTELFRANCHI et W.L. JOHNSON, éditeurs : *The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings*, pages 1129–1136. ACM Press, 2002.

K. H. DAM et M. WINIKOFF : Comparing Agent-Oriented Methodologies. *Dans Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2003), Melbourne, Australia, at AAMAS'03*, volume 3030 de *Lecture Notes in Computer Science (LNCS)*, pages 78–93. Springer-Verlag, 2003.

K. A. DE JONG : *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*. Thèse de doctorat, University of Michigan, 1975.

S.A. DELOACH : Multiagent Systems Engineering : a Methodology and Language for Designing Agent Systems. *Dans Proceedings of Agent Oriented Information Systems '99 (AOIS'99)*, pages 45–57, 1999.

S.A. DELOACH et M.F. WOOD : Developping MultiAgent Systems with agentTool. *Dans* C. CASTELFRANCHI et Y. LESPERANCE, éditeurs : *The Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL-2000)*, volume 1986 de *Lecture Notes in Computer Science (LNCS)*, pages 46–60. Springer-Verlag, 2000.

S.A. DELOACH, M.F. WOOD et C.H. SPARKMAN : Multiagent Systems Engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.

Y. DEMAZEAU : From Interactions to Collective Behaviour in Agent-Based Systems. *Dans Proceedings of the First European Conference on Cognitive Science (ECCS'95), Saint Malo, France*, pages 117–132, 1995.

Y. DEMAZEAU : MAS Methodology. Présentation lors du Séminaire IRIT - Cycle Systèmes Multi-Agents, Institut de Recherche en Informatique de Toulouse - Université Paul Sabatier, 15 Novembre 2001a.

Y. DEMAZEAU : Voyelles. Mémoire d'habilitation à diriger des recherches, INP Grenoble, 2001b.

Y. DEMAZEAU : Créativité émergente centrée utilisateur. *Dans* J.P. BRIOT et G. KHALED, éditeurs : *Déploiement des systèmes multi-agents – Vers un passage à l'échelle (Journées Francophones sur les Systèmes Multi-Agents, JFSMA'03)*, pages 31–36. Hermès - Lavoisier (Revue RSTI Hors-série), 2003.

J. DENEUBOURG, S. GOSS, N. FRANKS, A. SENDOVA-FRANKS, C. DETRAIN et L. CHRETIEN : The Dynamics of Collective Sorting : Robot-like Ants and Ant-like Robots. *Dans Proceedings of the First International Conference on Simulation of Adaptive Behavior : From Animals to Animats*, pages 356–363, 1990.

P. DESFRAY : UML Profiles Versus Metamodel Extensions : An Ongoing Debate. *Dans OMG's UML Workshops : UML in the .com Enterprise : Modeling CORBA, Components, XML/XMI and Metadata Workshop*, 2000.

G. DI MARZO SERUGENDO, A. KARAGEORGOS, O.F. RANA et F. ZAMBONELLI, éditeurs. *Engineering Self-Organising Systems : Nature-Inspired Approaches to Software Engineering*, volume 2977 de *Lecture Notes in Artificial Intelligence (LNAI)*, 2004.

M. DORIGO, V. MANIEZZO et A. COLORNI : Ant System : Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man and Cybernetics- Part B : Cybernetic*, 26(1):29–41, 1996.

F. DOTTO : *Acheminement adaptatif auto-organisé par coopération*. Thèse de doctorat, Université Paul Sabatier - Toulouse III, 1996.

A. DROGOUL : Systèmes multi-agents situés. Mémoire d'habilitation à diriger des recherches, Université Paris 6, 2000.

A. DROGOUL et A. COLLINOT : Applying an agent-oriented methodology to the design of artificial organizations : A case study in robotic soccer. *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):113–129, 1998.

A. DROGOUL et J.-D. ZUCKER : Methodological Issues for Designing Multi-Agent Systems with Machine Learning Techniques : Capitalizing Experiences from the RoboCup Challenge. Rapport Interne 42/98, LIP6 - Université de Paris 6, 1998.

J. DRÉO, A. PÉTROWSKI, P. SIARRY et E. TAILLARD : *Métaheuristiques pour l'optimisation difficile*. Eyrolles, 2003.

E. H. DURFEE, V.R. LESSER et D. D. CORKILL : Coherent Cooperation Among Communicating Problem Solvers. *IEEE Transactions on Computers*, C36(11):1275–1291, 1987.

EURESCOM : MESSAGE : Methodology for Engineering Systems of Software Agents - Deliverable 1 - Initial Methodology. Project P907-GI, EURESCOM, 2000.

J. FERBER : *Les système multi-agents : Vers une intelligence collective*. InterEditions, 1995.

J. FERBER et O. GUTKNECHT : Aalaadin : a meta-model for the analysis and design of organizations in multi-agent systems. *Dans* DEMAZEAU, Y., éditeur : *3rd International Conference on Multi-Agent Systems*, pages 128–135, Paris, 1998. IEEE.

J. FERBER, O. GUTKNECHT et F. MICHEL : From Agents to Organizations : An Organizational View of Multi-agent Systems. *Dans* P. GIORGINI, J. P. MÜLLER et J. ODELL, éditeurs : *Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, 2003, Revised Papers*, volume 2935 de *Lecture Notes in Computer Science (LNCS)*, pages 214–230. Springer-Verlag, 2003.

L. J. FOGEL : *Artificial Intelligence through Simulated Evolution*. Wiley, N.Y.C., 1966.

S. FORREST : Emergent Computation : Self-Organizing, Collective and Cooperative Phenomena in Natural and Artificial Computing Networks. *Special issue of Physica D*, 1991.

L. GASSER, C. BRAGANZA et N. HERMAN : MACE : A Flexible Testbed fo Distributed AI Research. *Distributed Artificial Intelligence, Reserach Notes on Artificial Intelligence*, pages 119–152, 1987.

J.-P. GEORGÉ : L'émergence. Rapport interne IRIT/2003-12-R, Institut de Recherche en Informatique de Toulouse (IRIT), 2003.

J.-P. GEORGÉ : *Résolution de problèmes par émergence – Étude d'un Environnement de Programmation Émergente*. Thèse de doctorat, Université Paul Sabatier - Toulouse III, 2004.

J.-P. GEORGÉ, M.-P. GLEIZES et P. GLIZE : Conception de systèmes adaptatifs à fonctionnalité émergente : la théorie AMAS. *Revue des sciences et technologie de l'information*, 17(4):591–626, 2003a.

J.-P. GEORGÉ, M.-P. GLEIZES, P. GLIZE et C. RÉGIS : Real-time Simulation for Flood Forecast : an Adaptive Multi-Agent System STAFF. *Dans Proceedings of the AISB'03 symposium on Adaptive Agents and Multi-Agent Systems, University of Wales, Aberystwyth*, 2003b.

J.-P. GEORGÉ, G. PICARD, M.-P. GLEIZES et P. GLIZE : Living Design for Open Computational Systems. *Dans* M. FREDRIKSSON, A. RICCI, R. GUSTAVSSON et A. OMICINI, éditeurs : *International Workshop on Theory And Practice of Open Computational Systems (TAPOCS) at 12th IEEE International Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE'03)*, pages 389–394, Linz, Austria, June 2003c. IEEE Computer Society.

M-P. GLEIZES et P. GLIZE : ABROSE : Des systèmes multi-agents pour le courtage adaptatif. *Dans* S. PESTY et C. SAYETTAT-FAU, éditeurs : *Systèmes multi-agents – Méthodologie, technologie et expériences – Actes des 8èmes JFIADSMA*, pages 117–132. Hermès, Octobre 2000.

M.-P. GLEIZES, J. LINK-PEZET et P. GLIZE : An Adaptive Multi-Agent Tool for Electronic Commerce. *Dans Second International Workshop on Knowledge Media Networking, Gettysburg, USA*, 2000.

M.-P. GLEIZES, T. MILLAN et G. PICARD : ADELFE : Using SPEM Notation to Unify Agent Engineering Processes and Methodology. Rapport interne IRIT/2003-10-R, Institut de Recherche en Informatique de Toulouse (IRIT), 2003.

P. GLIZE : L'adaptation des systèmes à fonctionnalité émergente par auto-organisation coopérative. Mémoire d'habilitation à diriger des recherches, Université Paul Sabatier - Toulouse III, 2001.

P. GLIZE, M.-P. GLEIZES et V. CAMPS : Coopérer pour apprendre sans présupposer. *Dans Conference on Learning : from Nartural Principles to Artificial Methods, Genève, Suisse*, 1997.

D .E. GOLDBERG : *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

J. GOLDSTEIN : Emergence as a Construct : History and Issues. *Journal of Complexity Issues in Organizations and Management*, 1(1), 1999.

J. GOMEZ SANZ et R. FUENTES : Agent oriented system engineering with ingenias. *Dans Fourth Iberoamerican Workshop on Multi-Agent Systems, Iberagents'02*, 2002.

O. GUTKNECHT et J. FERBER : Vers une méthodologie organisationnelle de conception. *Dans* M.-P. GLEIZES et P. MARCENAC, éditeurs : *Ingénierie des systèmes multi-agents – Actes des 7èmes JFIADSMA*, pages 93–104. Hermès, 1999.

H. HAKEN : *Synergentics : an Introduction*. Springer Verlag, 1978.

D. HALES et B. EDMONDS : Can Tags Build Working Systems ? From MABS to ESOA. *Dans* G. DI MARZO SERUGENDO, A. KARAGEORGOS, O.F. RANA et F. ZAMBONELLI, éditeurs : *Engineering Self-Organizing Applications – First International Workshop (ESOA) at the Second International Joint Conference on Autonomous Agents and Multi-Agents Systems (AAMAS'03)*, volume 2977 de *Lecture Notes in Artificial Intelligence (LNAI)*, pages 186–194. Springer-Verlag, 2003a.

D. HALES et B. EDMONDS : Evolving Social Rationality for MAS using "Tags". *Dans* J. S. ROSENSCHEIN, T. SANDHOLM, M. WOOLDRIDGE et M. YOKOO, éditeurs : *Proceedings of the 2nd International Conference on Autonomous Agents and Multiagent Systems, Melbourne, July 2003 (AAMAS 2003)*, pages 497–503. ACM Press, 2003b.

F. HARROUET : *oRis : s'immerger par le langage pour le prototypage d'univers virtuels à base d'entités autonomes*. Thèse de doctorat, Université de Bretagne Occidentale, 2000.

F. HEYLIGHEN : *The Science of Self-organization and Adaptivity*. EOLSS Publishers, 1999.

J. H. HOLLAND : *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.

J. H. HOLLAND : The effect of labels (tags) on social interactions. SFI Working Paper 93-10-064, Santa Fe Institute, 1993.

J.H. HOLLAND : *Emergence : From Order to Chaos*. Oxford University Press, 1998.

J. HORN : Autonomic Computing : IBM's Perspectives on the State of Information Technology. Rapport technique, IBM Research, 2001. URL `http://www.ibm.com/research/autonomic`.

C. A. IGLESIAS, M. GARIJO, J. C. GONZÁLEZ et J. R. VELASCO : Analysis and design of multiagent systems using MAS-CommonKADS. *Dans* M. P. SINGH, A. RAO et M. J. WOOLDRIDGE, éditeurs : *Intelligent Agents IV : Agent Theories, Architectures and Languages*, volume 1365 de *Lecture Notes in Artificial Intelligence (LNAI)*. Springer-Verlag, 1998.

I. JACOBSON, G. BOOCH et J. RUMBAUGH : *The Unified Software Development Process*. Addison-Wesley Longman Publishing, 1999.

D. KINNY, M. GEORGEFF et A. RAO : A Methodology and Modelling Technique for Systems of BDI agents. *Dans* W. Van de VELDE et J. W. PERRAM, éditeurs : *Agents Breaking Away : Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a MultiAgent World*, volume 1038 de *Lecture Notes in Artificial Intelligence (LNAI)*, pages 51–71. Springer-Verlag, 1996.

T. KOHONEN : *Self-Organising Maps*, volume 30 de *Springer Series in Information Sciences*. Springer-Verlag, 2001.

C. R. KUBE et H. ZHANG : Collective robotics intelligence : From social insects to robots. *Dans Proceedings, 3rd International Conference on Simulation of Adaptive Behavior*, 1993.

K. LAGUS, T. HONKELA, S. KASKI, et T. KOHONEN : WEBSOM for textual data mining. *Artificial Intelligence Review*, 13:345–364, 1999.

C. LANGTON : Computation at the Edge of Chaos – Phase Transition and Emergent Computation. *Physica D*, 4, 1990.

V. R. LESSER et D. G. CORKILL : DVMT : The Distributed Vehicle Monitoring Testbed : A Tool for Investigating Distributed Problem Solving Networks. *AI Magazine*, 4(3):15–33, 1983.

J. LIND : *Iterative Software Engineering for Multiagent Systems : the MASSIVE Method*, volume 1994 de *Lecture Notes in Artificial Intelligence (LNAI)*. Springer Verlag, 2001.

E. MACCHION : Plate-forme « système multi-agent adaptatif » pour une exploration comportementale transcriptionnelle – Application à la levure Saccharomyces cerevisiae à partir de données métaboliques et de puces à ADN. Mémoire d'Ingénieur C.N.A.M. Informatique, Conservatoire National des Arts et Métiers de Toulouse, 2004.

M. MANELA et J. A. CAMPBELL : Designing Good Pursuit Problems as Testbeds for Distributed AI : A Novel Application of Genetic Algorithms. *Dans* C. CASTELFRANCHI et J.-P. MÜLLER, éditeurs : *From Reaction to Cognition, 5th European Workshop on Modelling Autonomous Agents, MAAMAW '93, Neuchatel, Switzerland, August 25-27, 1993, Selected Papers*, volume 957 de *Lecture Notes in Computer Science (LNCS)*, pages 231–252. Springer-Verlag, 1995.

J. P. MANO et P. GLIZE : Self-adaptive Network of Cooperative Neuro-agents. *Dans AISB'04 Symposium on Adaptive Multi-Agent Systems*, 2004.

H.R MATURANA et F.J. VARELA : *The Tree of Knowledge*. Addison-Wesley, 1994.

P. MORAITIS, E. PETRAKI et N. SPANOUDAKIS : Engineering JADE Agents with Gaia Methodology. *Dans* R. KOWALCZYK, J. MULLER, H. TIANFIELD et R. UNLAND, éditeurs : *Agent Technologies, Infrastructures, Tools and Applications for E-Services – Best (revised) papers of NODe 2002 Agent-Related Workshops*, volume 2592 de *Lecture Notes in Artificial Intelligence (LNAI)*, pages 77–92. Springer-Verlag, 2002.

J-P. MÜLLER : Emergence of collective behaviour : simulation and social engineering. *Dans* A. OMICINI, P. PETTA et J. PITT, éditeurs : *Fourth International Workshop on Engineering Societies in the Agents World (ESAW'03)*, volume 3071 de *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, 2003.

P.-A. MÜLLER et N. GAERTNER : *Modélisation objet avec UML*. Eyrolles, 2000.

H. NWANA, D. NDUMU, L. LEE et J. COLLIS : ZEUS : A Tool-Kit for Building Distributed Multi-Agent Systems. *Applied Artifical Intelligence Journal*, 13(1):129–186, 1999.

M. OCCELLO, Y. DEMAZEAU et C. BAEIJS : Designing Organized Agents for Cooperation in a Real Time Context. *Dans* A. DROGOUL, M. TAMBE et T. FUKUDA, éditeurs : *Collective Robotics*, volume 1456 de *Lecture Notes in Artificial Intelligence (LNAI)*, pages 25–37. Springer-Verlag, 1998.

M. OCCELLO, J.-L. KONING et C. BAEIJS : Conception de SMA. Quelques éléments de réflexion méthodologique. *Revue Technique et Science Informatiques*, 20(2), 2001.

J. ODELL : Objects and Agents Compared. *Journal of Object Technology*, 1(1):41–53, 2002.

J. ODELL, H. VAN DYKE PARUNAK et B. BAUER : Extending UML for Agents. *Dans Proceedings of the Agent-Oriented Information Systems (AOIS) Workshop at the 17th National Conference on Artificial Intelligence (AAAI)*, pages 3–17, 2000.

J. ODELL, H. VAN DYKE PARUNAK et B. BAUER : Representing Agent Interaction Protocols in UML. *Dans First International Workshop, AOSE 2000 on Agent-Oriented Software Engineering*, pages 121–140. Springer-Verlag, 2001.

OMG : Software Process Engineering Metamodel Specification Version 1.0. Rapport technique Version 1.0, formal/02-11-14, Object Management Group, 2002.

L. PADGHAM et M. WINIKOFF : Prometheus : a Pragmatic Methodology for Engineering Intelligent Agents. *Dans* J. DEBENHAM, B. HENDERSON-SELLERS, N. JENNINGS et J. ODELL, éditeurs : *Proceedings of the OOPSLA 02 - Workshop on Agent-Oriented Methodologies*. COTAR, 2002.

L. PADGHAM et M. WINIKOFF : Prometheus : A Methodology for Developing Intelligent Agents. *Dans* F. GIUNCHIGLIA, J. ODELL et G. WEISS, éditeurs : *Agent-Oriented Software Engineering III, Third International Workshop, AOSE 2002, Bologna, Italy, July 15, 2002, Revised Papers and Invited Contributions*, volume 2585 de *Lecture Notes in Computer Science (LNCS)*, pages 174–185. Springer-Verlag, 2003.

A. PERINI, M. PISTORE, M. ROVERI et A. SUSI : Agent-oriented modeling by interleaving formal and informal specification. *Dans* P. GIORGINI, J. P. MÜLLER et J. ODELL, éditeurs : *Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, 2003, Revised Papers*, volume 2935 de *Lecture Notes in Computer Science (LNCS)*, pages 36–52. Springer-Verlag, 2003.

L. PETIT : Essai sur l'inférence adaptative dans JRules fondée sur la théorie des AMAS. Mémoire de Maîtrise IUP Ingénierie des Systèmes Informatiques, Université Paul Sabatier, 2004.

S. PEYRUQUEOU : Approche et applications de l'ingénierie des systèmes multi-agents. Mémoire de diplôme de recherche technologique (drt) en ingénierie des systèmes informatiques et automatiques, Université Paul Sabatier, 2002.

G. PICARD : Étude de l'émergence comportementale d'un collectif de robots par auto-organisation coopérative. Rapport technique, IRIT - Université Paul Sabatier, 2001.

G. PICARD, C. BERNON et M.-P. GLEIZES : Cooperative Agent Model within ADELFE Framework :An Application to a Timetabling Problem. *Dans Third Joint Conference on Multi-Agent System (AAMAS'04)*, pages 1506–1507, New York, USA, July 2004. IEEE Computer Society.

G. PICARD et M.-P. GLEIZES : An Agent Architecture to Design Self-Organizing Collectives : Principles and Application. *Dans* D. KAZAKOV, D. KUDENKO et E. ALONSO, éditeurs : *AISB'02 Symposium on Adaptive Multi-Agent Systems (AAMASII)*, volume 2636 de *Lecture Notes in Artificial Intelligence (LNAI)*, pages 141–158. Springer-Verlag, 2002.

G. PICARD et M.-P. GLEIZES : The ADELFE Methodology – Designing Adaptive Cooperative Multi-Agent Systems. *Dans* F. BERGENTI, M-P. GLEIZES et F. ZAMBONELLI, éditeurs : *Methodologies and Software Engineering for Agent Systems (Chapter 8)*, pages 157–176. Kluwer Publishing, 2004.

I. PRIGOGINE et G. NICOLIS : *Self Organization in Non-Equilibrium Systems*, chapitre III & IV. Wiley and Sons, 1977.

A. S. RAO et M. P. GEORGEFF : BDI-agents : from theory to practice. Rapport technique 56, Australian Artificial Intelligence Institute, Melbourne, Australia, 1995.

I. RECHENBERG : *Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog, Stuttgart, 1973.

P-M.. RICORDEL : *Programmation Orientée Multi-Agents : Développement et Déploiement de Systèmes Multi-Agents Voyelles*. Thèse de doctorat, Institut National Polytechnique de Grenoble, 2001.

P-M. RICORDEL et Y. DEMAZEAU : From Analysis to Deployment : A Multi-agent Platform Survey. *Dans* A. OMICINI, R. TOLKSDORF et F. ZAMBONELLI, éditeurs : *Engineering Societies in the Agent World, First International Workshop, ESAW 2000, Berlin, Germany, August 21, 2000, Revised Papers*, volume 1972 de *Lecture Notes in Computer Science (LNCS)*, pages 93–105. Springer-Verlag, 2000.

P-M. RICORDEL et Y. DEMAZEAU : Volcano, a Vowels-Oriented Multi-agent Platform. *Dans* B. DUNIN-KEPLICZ et E. NAWARECKI, éditeurs : *From Theory to Practice in Multi-Agent Systems, Second International Workshop of Central and Eastern Europe on Multi-Agent Systems, CEEMAS 2001 Cracow, Poland, September 26-29, 2001, Revised Papers*, volume 2296 de *Lecture Notes in Computer Science (LNCS)*, pages 253–262. Springer-Verlag, 2002.

S. RUSSEL et P. NORVIG : *Artificial Intelligence : a Modern Approach*. Prentice-Hall, 1995.

C. RÉGIS, T. SONTHEIMER, M-P. GLIEZES et P. GLIZE : STAFF : Un système multi-agent adaptatif en prévision de crues. *Dans* P. MATHIEU et J-P. MÜLLER, éditeurs : *Systèmes multi-agents et systèmes complexes – ingénierie, résolution de problèmes et simulations – Actes des 10èmes JFIADSMA*, pages 87–98. Hermès, 2002.

O. SHEHORY et A. STURM : Evaluation of Modeling Techniques for Agent-Based Systems. *Dans Proceedings of the Fifth International Conference on Autonomous Agents*, pages 624–631. ACM Press, 2001.

O. SIMONIN et J. FERBER : Un modèle multi-agent de résolution collective de problèmes situés multi-échelles. *Dans* J.P. BRIOT et G. KHALED, éditeurs : *Journées Francophones sur les Systèmes Multi-Agents, JFSMA'03*, pages 317–330, Hammamet, Tunisia, 27-29 November 2003. Hermès.

K. SOCHA, J. KNOWLES et M. SAMPELS : A $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System for the University Timetabling Problem. *Dans* M. DORIGO, G. DI CARO et M. SAMPELS, éditeurs : *Proceedings of ANTS 2002 – Third International Workshop on Ant Algorithms*, volume 2463 de *Lecture Notes in Computer Science (LNCS)*, pages 1–13. Springer-Verlag, September 2002.

K. SOCHA, M. SAMPELS et M. MANFRIN : Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art. *Dans Proceedings of EvoCOP 2003 – 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization*, volume 2611 de *Lecture Notes in Computer Science (LNCS)*, pages 334–345. Springer-Verlag, April 2003.

R. SOLEY et THE OMG STAFF STRATEGY GROUP : Model driven architecture. White paper Draft 3.2, OMG, 2002.

T. SONTHEIMER et P. GLIZE : STAFF : La prévision de crues par modèle adaptatif. *Dans* S. PESTY et C. SAYETTAT-FAU, éditeurs : *Systèmes multi-agents – Méthodologie, technologie et expériences – Actes des 8èmes JFIADSMA*, pages 249–252. Hermès, Octobre 2000.

L. STEELS : Cooperation between Distributed Agents through Self-Organization. *Distributed Artificial Intelligence*, 1, 1996.

A. STURM, D. DORI et O. SHEHORY : Single-Model Method for Specifying Multi-Agent Systems. *Dans* J. S. ROSENSCHEIN, T. SANDHOLM, M. WOOLDRIDGE et M. YOKOO, éditeurs : *Proceedings of the 2nd International Conference on Autonomous Agents and Multiagent Systems, Melbourne, July 2003 (AAMAS 2003)*, pages 121–128. ACM Press, 2003.

A. STURM et O. SHEHORY : A Framework for Evaluating Agent-Oriented Methodologies. *Dans Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2003), Melbourne, Australia, at AAMAS'03*, volume 3030 de *Lecture Notes in Computer Science (LNCS)*, pages 94–109. Springer-Verlag, 2003.

R. TOLKSDORF et R. MENEZES : Using Swarm Intelligence in LINDA Systems. *Dans Fourth International Workshop on Engineering Societies in the Agents World (ESAW'03), Imperial College London, UK, 29-31 October*, volume 3071 de *Lecture Notes in Artificial Intelligence (LNAI)*. Springer-Verlag, 2004.

X. TOPIN, V. FOURCASSIÉ, M.-P. GLEIZES, G. THERAULAZ, C. RÉGIS et P. GLIZE : Theories and experiments on emergent behaviour : From natural to artificial systems and back. *Dans Proceedings on European Conference on Cognitive Science, Sien*, 1999a.

X. TOPIN, C. RÉGIS, M.-P. GLEIZES et P. GLIZE : Comportements individuels adaptatifs dans un environnement dynamique pour l'exploitation collective de ressources. *Dans* A. DROGOUL et J.-A. MEYER, éditeurs : *Intelligence Artificielle Située, cerveau, corps et environnement*. Hermès, 1999b.

A.M. TURING : The Chemical Basis of Morphogenesis. *Philosophical Transactions of the Royal Society B 237*, 1952.

F. VAN AEKEN : *Les systèmes multi-agents minimaux*. Thèse de doctorat, Institut National Polytechnique de Grenoble, 1999.

G. Van de VIJVER : Emergence et explication. *Intellectica : Emergence and explanation*, 1997.

H. VAN DYKE PARUNAK : "Go to the Ant" : Engineering Principles from Natural Agent Systems. *Annals of Operations Research*, 75:69–101, 1997.

E. VAREILLES : Modélisation formelle des systèmes multi-agents adaptatifs. Rapport de dea programmation et systèmes, Université Paul Sabatier, 2002.

R. VAUGHAN, K. STØY, G. SUKHATME et M. MATARIĆ : Blazing a trail : Insect-inspired resource transportation by a robotic team. *Dans Proceedings of 5th International Symposium on Distributed Robotic Systems*, 2000a.

R. VAUGHAN, K. STØY, G. SUKHATME et M. MATARIĆ : Go ahead make my day : Robot conflict resolution by aggressive competition. *Dans Proceedings of the 6th International Conference on Simulation of Adaptive Behaviour*, 2000b.

L. von BERTALANFFY : *Théorie générales des systèmes*. Editions Dunod, 1993.

H. von FOERSTER : *On Self-Organizing Systems and their Environments*, pages 31–50. Pergamon, 1960.

J. von NEUMANN : *Theory of Self-Reproducing Automata*. Urbana : University of Illinois Press, 1966.

G. WAGNER : The Agent-Object-Relationship Metamodel : Towards a Unified View of State and Behavior. Rapport technique, Eindhoven Univ. of Technology, Fac. of Technology Management, 2002.

G. WAGNER : A UML Profile for External AOR Models. *Dans* F. GIUNCHIGLIA, J. ODELL et G. WEISS, éditeurs : *Agent-Oriented Software Engineering III, Third International Workshop, AOSE 2002, Bologna, Italy, July 15, 2002, Revised Papers and Invited Contributions*, volume 2585 de *Lecture Notes in Computer Science (LNCS)*, pages 138–149. Springer-Verlag, 2003.

M. WOOLDRIDGE : On the Sources of Complexity in Agent Design. *Applied Artificial Intelligence*, 14(7):623–644, 2000.

M. WOOLDRIDGE, N. R. JENNINGS et D. KINNY : The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

M. WOOLDRIDGE et N.R. JENNINGS : Agent theories, architectures, and languages : a survey. *Dans Proceedings of the Workshop on Agent Theories, Architectures, and Languages on Intelligent Agents (ATAL'95)*, pages 1–39. Springer-Verlag, 1995.

E. YU et J. MYLOPOULOS : Towards Modelling Strategic Actor Relationships for Information Systems Development – With Examples from Business Process Reengineering. *Dans Proceedings of the 4th Workshop on Information Technologies and Systems*, pages 21–28, 1994.

Liste des figures

1.1 La métaphore de la boîte noire : le proto-émergentisme (en haut) et le néo-émergentisme (en bas). 9

1.2 Adaptation à un environnement dynamique et auto-organisation. 12

1.3 Un exemple d'auto-organisation : l'expérience des cellules de Bénard. 13

1.4 La réaction de Belousov-Zhabotinski : un exemple d'auto-organisation par réaction-diffusion. 14

1.5 Le théorème de l'adéquation fonctionnelle : les relations d'inclusion entre systèmes adéquats et coopératifs. 21

2.1 Représentation d'un protocole d'interaction grâce à un diagramme de séquence A-UML. 34

2.2 Représentation de l'état interne d'un agent lors d'un protocole d'interaction grâce à un diagramme d'états/transitions. 34

2.3 Les différents éléments de modélisation externe d'AOR. 35

2.4 Le métamodèle UML d'AGR 40

2.5 Généalogie des méthodes orientées agent et les différentes influences 41

2.6 Un exemple de modélisation orientée intention avec *i*⋆ [Castro *et al.*, 2001]. . . 43

2.7 Un exemple de diagramme de classes et d'instances pour le problème du trafic aérien dans AAII. 48

2.8 Les diagrammes de structure organisationnelle (à gauche) et d'organisation concrète (à droite) dans Aalaadin. 50

2.9 Les cinq calques de Cassiopée (en gris le micro-niveau, en blanc le macro-niveau). 52

2.10 Le modèle d'agent générique de DESIRE – décomposition en tâches. 55

2.11 Le processus de développement de Gaia et les modèles manipulés. 58

2.12 Les différentes étapes et les modèles de MaSE. 61

2.13 Le processus itératif de la méthode MASSIVE. 63

2.14 Les deux niveaux d'analyse de MESSAGE et les modèles associés. 65

2.15 Les cinq modèles/phases de la méthode PASSI. 68
2.16 Les trois phases de Prometheus et les modèles manipulés. 72
2.17 L'approche générale de la méthode Voyelles. 77

3.1 Les quatre premières définitions de travaux du processus d'ADELFE. 89
3.2 La définition de travaux WD_1 : les besoins préliminaires. 93
3.3 La définition de travaux WD_2 : les besoins finals. 97
3.4 Diagramme de collaboration entre le système et le plan pédagogique national (ou NPP). 98
3.5 Diagramme de collaboration entre le système et les salles. 98
3.6 Les cas d'utilisation d'ETTO. 100
3.7 Cas d'utilisation entre le gestionnaire de cours et le système. 102
3.8 Cas d'utilisation entre le gestionnaire de salles et le système. 102
3.9 Cas d'utilisation entre un enseignant et le système. 102
3.10 Cas d'utilisation entre un groupe d'étudiants et le système. 102
3.11 La définition de travaux WD_3 : l'analyse. 105
3.12 Le diagramme de classes préliminaire d'ETTO. 106
3.13 Diagramme de séquences d'ETTO représentant les interactions entre un enseignant et le gestionnaire contraintes. 113
3.14 Diagramme de séquences d'ETTO représentant les interactions entre le gestionnaire de salles et les salles. 113
3.15 La définition de travaux WD_4 : la conception. 115
3.16 Les différents modules d'un agent coopératif et leurs dépendances 116
3.17 Diagramme de classes du paquetage agent. 120
3.18 Diagramme de classes du paquetage grid. 120
3.19 Diagramme de classes du paquetage constraint. 120
3.20 Diagramme de classes du paquetage interface. 120
3.21 Un exemple de protocole d'interactions entre deux agents de réservation (BookingAgents) d'ETTO. 121
3.22 Diagramme de classes d'agents pour ETTO. 124

4.1 Un exemple de protocole dans ADELFE. 142
4.2 Un exemple de protocole de requête indirecte avec clause IF (rectanggle de couleur à gauche de la ligne de vie de AMA. 143
4.3 Machine à états finis pour la classe Buyer du protocole de la figure 4.1. 145
4.4 Machine à états finis pour la classe Seller du protocole de la figure 4.1. 145

4.5 Sous-machine de l'état ContractInitiation de la classe Buyer pour le rôle Buyer Role. 146
4.6 Sous-machine de l'état Seller_cfp de la classe Seller pour le rôle Seller Role. . . 146
4.7 Sous-machine de l'état SelectIndirectRequest de la classe AMA du protocole de la figure 4.2. 147
4.8 Diagramme de classes d'agents dans OpenTool UML1.4 / Adelfe 1.2. 149
4.9 Gestion des erreurs d'utilisation des stéréotypes ADELFE dans OpenTool. . . 150
4.10 L'outil d'aide au suivi du processus d'ADELFE : AdelfeToolkit. 152
4.11 L'outil d'adéquation aux AMAS d'AdelfeToolkit. 153

5.1 Variation des contraintes en cours de résolution pour la variante 1. 161
5.2 Variation du temps de résolution en fonction du nombre d'agents. 162
5.3 Variation des contraintes en cours de résolution pour la variante 2. 163
5.4 Variation des contraintes en cours de résolution pour la variante 4, avec suppression d'un BookingAgent après stabilisation du système. 163
5.5 Variation des contraintes en cours de résolution pour la variante 4, avec suppression de huit BookingAgents après stabilisation du système. 164

6.1 L'environnement de transport multi-robot de ressources. 168
6.2 Le diagramme de classes préliminaire d'analyse pour le problème de transport collectif. 170
6.3 Nombre de boîtes rapportées pour 15 simulations (300 robots, 2 couloirs, 5 cases de perception), du comportement nominal (individualiste) et des deux comportements coopératifs : le comportement de déblocage (voir 6.3.1) et le comportement d'anticipation avec déblocage (voir 6.3.2) 175
6.4 Courbe de fréquentation d'un couloir pour les deux comportements coopératifs : le comportement de déblocage uniquement (à droite) et le comportement d'anticipation avec déblocage (à gauche). 176
6.5 Placement des balises (en bleu) pour les deux buts. 176
6.6 Environnement "difficile". Placement des balises (en rouge) dans un environnement "difficile" pour le but *atteindre la zone de dépôt*. 178
6.7 Fréquentation d'un couloir dans un environnement "difficile". 178
6.8 Placement des balises virtuelles dans un environnement dynamique à quatre couloirs dont deux sont obturés, pour 300 robots. 180
6.9 Fréquentation des couloirs pour les deux buts : *atteindre la zone de retrait* (à gauche) ou *atteindre la zone de dépôt* (à droite), pour 300 robots. 180
6.10 Nombre de boîtes rapportées pour un environnement dynamique à quatre couloirs, pour 300 robots, comparé au nombre de boîtes rapportées dans un environnement statique à deux couloirs. 180

7.1 Décalage de processus objet-agent par chevauchement pour les systèmes artificiels ouverts (*Open Computational Systems* ou OCS). 190

Liste des tableaux

2.1 L'évolution des approches de programmation : de la programmation monolithique à la programmation objet [Odell, 2002]. 31
2.2 Synthèse de la comparaison des différentes méthodes 78

3.1 Les quatre niveaux du MOF. 86
3.2 Un schéma d'analyse des agents pour l'identification des SNC. 125
3.3 Analyse des SNC pour un BookingAgent dans ETTO. 125

5.1 Une solution à la variante 1 d'ETTO. 158
5.2 Une solution à la variante 2 d'ETTO (en gras apparaissent les partenariats avec relâchement de contraintes). 159

6.1 Exemple de formulation de la situation non coopérative d'inutilité 174

7.1 Evaluation d'ADELFE et comparaison aux autres méthodes. 187