

Université de Sfax  
Formation Doctorale en Informatique  
Faculté des Sciences Economiques et de Gestion de Sfax



# THESE DE DOCTORAT

Présentée en vue de l'obtention du

## DOCTORAT EN INFORMATIQUE

Présentée par  
**Amira REGAYEG TRIGUI**

---

### APPROCHE FORMELLE DE DEVELOPPEMENT DE SYSTEMES MULTI AGENTS : DE LA SPECIFICATION A LA CONCEPTION

---

Mr	Abdelmajid Ben Hamadou	Professeur à l'ISIM, Sfax	Président
Mme	Marie-Pierre Gleizes	Professeur à l'université Paul Sabatier, Toulouse III	Rapporteur
Mme	Hanene Ben Abdallah	Maître de conférences à la FSEG, Sfax	Rapporteur
Mr	Khaled Ghédira	Professeur à l'ISG, Tunis	Examineur
Mr	Mohamed Jmaiel	Maître de conférences à l'ENI, Sfax	Directeur de thèse
Mr	Ahmed Hadj Kacem	Maître de conférences à la FSEG, Sfax	Encadreur

---

Unité de Recherche en Développement et Contrôle d'Applications Distribuées

\*\*\*

ReDCAD

*A ma mère, mon père et ma grand-mère pour leur amour  
et leur patience*

*A mon mari Mohamed qui m'a toujours soutenue*

*A mon petit Ali et ma petite Fatma*

*A ma soeur et mes frères*

*A toute ma famille et ma belle-famille*

# Table des matières

Table des Figures	vi
Remerciements	viii
Introduction générale	1
<b>1 Les systèmes multi-agents : de la recherche vers l'application</b>	<b>5</b>
1.1 Les aspects des SMA . . . . .	6
1.1.1 Présentation des SMA . . . . .	6
1.1.2 Les méthodes de conception des SMA . . . . .	8
1.1.3 Les SMA et la simulation . . . . .	10
1.2 Les domaines d'application . . . . .	12
1.2.1 Les SMA et les applications grand public . . . . .	13
1.2.2 Les SMA et les applications coopératives . . . . .	15
1.2.3 Les SMA et les applications de gestion . . . . .	19
1.2.4 Les SMA et les applications temps réel . . . . .	22
1.3 Synthèse . . . . .	24
<b>2 État de l'art</b>	<b>27</b>
2.1 Les méthodes de conception des applications multi-agents . . . . .	29

2.1.1	Les méthodes semi-formelles . . . . .	30
2.1.2	Les méthodes formelles . . . . .	35
2.2	Étude comparative des principales méthodes formelles . . . . .	42
2.2.1	Le critère 1 : la couverture des aspects individuels et collectifs	43
2.2.2	Le critère 2 : le langage de spécification . . . . .	44
2.2.3	Le critère 3 : le processus de conception . . . . .	46
2.2.4	Le critère 4 : la vérification . . . . .	48
2.2.5	Le critère 5 : les aides méthodologiques . . . . .	50
2.3	Synthèse . . . . .	50
<b>3</b>	<b><math>\mathcal{T}_{temporal}Z</math> : un langage de spécification multi-formalisme</b>	<b>55</b>
3.1	Choix des formalismes de base . . . . .	56
3.2	La notation $Z$ . . . . .	60
3.2.1	Les langages de $Z$ . . . . .	61
3.2.2	Le raffinement de données dans $Z$ . . . . .	63
3.2.3	Les preuves dans $Z$ . . . . .	64
3.2.4	Les avantages et les limites de $Z$ . . . . .	66
3.3	La logique temporelle linéaire (LTL) . . . . .	67
3.3.1	Le Langage de LTL . . . . .	68
3.3.2	Le système déductif de LTL . . . . .	70
3.3.3	Les preuves dans LTL . . . . .	71
3.4	$\mathcal{T}_{temporal}Z$ : intégration de $Z$ et LTL . . . . .	71
3.4.1	Le modèle temporel . . . . .	72
3.4.2	La sémantique des formules temporelles . . . . .	73
3.4.3	La relation de raffinement . . . . .	75

3.4.4	Le pouvoir expressif de $\mathcal{T}_{\text{temporal}}\mathcal{Z}$ . . . . .	77
3.5	Conclusion . . . . .	79
<b>4</b>	<b><math>\mathcal{F}_{\text{or}}\mathcal{MAAD}</math> : une approche formelle de développement d'applications à base d'agents</b>	<b>81</b>
4.1	Une méthode formelle de développement des applications à base d'agent	83
4.1.1	Phase de spécification . . . . .	83
4.1.2	Phase de conception . . . . .	85
4.2	$\mathcal{F}_{\text{or}}\mathcal{MAAD}\text{Tools}$ : . . . . .	102
4.2.1	Environnement de développement . . . . .	102
4.2.2	Phase de spécification . . . . .	103
4.2.3	Phase de conception . . . . .	104
4.3	Conclusion . . . . .	105
<b>5</b>	<b>Études de cas</b>	<b>109</b>
5.1	Le jeu de poursuite . . . . .	109
5.1.1	Phase de spécification . . . . .	110
5.1.2	Phase de conception . . . . .	112
5.2	Le contrôle du trafic aérien . . . . .	115
5.2.1	Phase de spécification . . . . .	116
5.2.2	Phase de conception . . . . .	119
5.3	La gestion coopérative de pannes dans un réseau . . . . .	124
5.3.1	Phase de spécification . . . . .	126
5.3.2	Phase de conception . . . . .	128
5.4	Conclusion . . . . .	138

<b>Conclusion générale</b>	<b>141</b>
<b>Bibliographie</b>	<b>145</b>

# Table des figures

2.1	Aperçu sur la Conception des SMA . . . . .	27
2.2	Aperçu sur la classification des méthodes existantes . . . . .	29
2.3	Aperçu sur les méthodes formelles existantes . . . . .	36
2.4	Tableau récapitulatif pour l'étude comparative de cinq méthodes formelles selon cinq critères . . . . .	51
3.1	Le déroulement temporel d'un système dans une logique linéaire . . .	68
3.2	La représentation des actions des différentes entités par rapport à un axe de temps . . . . .	73
3.3	La description des principaux concepts des SMA avec $\mathcal{T}_{temporal}\mathcal{Z}$ . . .	78
4.1	Le processus de conception basé sur sept étapes de raffinement . . . .	86
4.2	Le graphe et/ou de l'objectif commun $A$ . . . . .	89
4.3	Les graphes de précédence des deux scénarios . . . . .	93
4.4	L'interface principale de la spécification des besoins . . . . .	103
4.5	La première interface de $\mathcal{F}_{or}\mathcal{MAAD}\mathcal{T}ools$ . . . . .	105
4.6	Un exemple de la génération automatique du graphe et/ou . . . . .	106
4.7	Un exemple de la proposition d'un scénario d'affectation des rôles . .	106
5.1	Le jeu de poursuite . . . . .	110

5.2	Le graphe et/ou généré par $\mathcal{F}_{or}\mathcal{MAAD} Tools$ . . . . .	113
5.3	La preuve par réduction du théorème VerifSpec avec Z-EVES . . . . .	115
5.4	Le plan de routage . . . . .	116
5.5	La preuve par réduction du théorème VerifSpec . . . . .	123
5.6	Le graphe et/ou de l'application CNFM généré par l'outil $\mathcal{F}_{or}\mathcal{MAAD} Tools$ . . . . .	129
5.7	Le graphe de précedence des buts locaux de CNFM . . . . .	132
5.8	La preuve par réduction du théorème VerifSpec de CNFM . . . . .	138

# Remerciements

Je tiens à remercier toutes les personnes qui ont participé à ce travail.

Tout d'abord je remercie M. Mohamed Jmaiel, maître de conférences à l'École Nationale d'Ingénieurs de Sfax et directeur de l'unité de Recherche en Développement et Contrôle des Applications Distribuées (ReDCAD) pour la confiance qu'il m'a témoigné tout au long de ce travail. Ce travail a grandement profité de son encadrement, de ses précieux conseils et de ses qualités humaines.

Je suis très reconnaissante à M. Ahmed Hadj Kacem, maître de conférences à la Faculté des Sciences Economiques et de Gestion de Sfax pour sa contribution dans le co-encadrement de ce travail. Ses conseils tant d'ordre méthodologique, technique que rédactionnel ont grandement contribué à la qualité de ce mémoire.

Mes remerciements vont également vers les membres de l'unité de recherche LARIS auquel j'ai fait partie ainsi que l'unité ReDCAD dont je suis actuellement membre.

Je tiens à remercier les étudiants de mastère qui, au cours de leurs projets, se sont intéressés à ma problématique de thèse et m'ont apporté du sang neuf et des idées nouvelles.

Je remercie aussi, mes collègues à l'Institut Supérieure de Gestion Industrielle ISGI Sfax et à la Faculté des Sciences Économiques et de Gestion FSEG Sfax pour les encouragements et les aides qui m'ont permis de finir cette thèse et surtout ceux qui m'ont fait profiter de leur grande expérience et avec qui j'ai eu de nombreuses discussions enrichissantes.

Je ne peux pas oublier mes amis et tous les membres de ma famille qui, tout au long de cette thèse, ont su supporter mes moments de doute.

# Introduction générale

L'informatique devient de plus en plus distribuée dans de multiples objets et fonctionnalités qui sont amenés à coopérer. Étant donnée cette distribution, la décentralisation des logiciels devient inévitable accentuant la complexité des applications informatiques. Pour réduire cette complexité, une *organisation* coopérative entre modules s'impose. De ce fait, et en vue de réduire l'intervention humaine, on est naturellement amené à donner plus d'*autonomie* et d'initiative aux différents modules.

Ces deux enjeux l'*autonomie* et l'*organisation* sont à la base du paradigme des systèmes multi-agents (SMA). Les SMA sont conçus comme un groupe d'entités intelligentes appelées "agents" qui interagissent par la coopération, la coexistence ou la compétition et simulant, ainsi, certaines formes de raisonnement, de connaissance et d'expertises. Ces systèmes ont été utilisés pour concevoir des applications (industrielles) complexes appartenant à des domaines variés tels que la gestion distribuée de réseaux [SN00] et les services WEB intelligents [Kep02]. La force de l'approche agent dans la résolution de problèmes complexes découle du fait que les agents grâce à leur autonomie et adaptabilité sont capables d'achever leurs buts de manière flexible en interagissant avec les autres à travers des protocoles et des langages de haut niveau [ZO04]. Néanmoins, cette flexibilité dans la résolution de problèmes révèle des difficultés dans la conception d'agents et de SMA. Ainsi, les développeurs doivent aborder des nouveaux concepts relatifs aux deux niveaux : (1) les aspects d'agent individuel tels que l'autonomie, le but, la connaissance, la capacité et le rôle ; et (2) les phénomènes résultant des aspects collectifs et sociaux, tels que l'organisation, la coopération, la négociation et la coordination.

Pour maîtriser cette complexité, des méthodes orientées-agent et des techniques de modélisation adéquates sont devenues essentielles. Plusieurs méthodes ont été proposées pour le développement des SMA. La majorité de ces méthodes sont semi-formelles. Ces dernières constituant soit une extension des méthodes orientées-objet, soit une extension des méthodes à base de connaissances, demeurent incomplètes et ne permettent pas de raisonner rigoureusement et de vérifier la satisfaction des propriétés jugées nécessaires dans la conception des SMA. Quant aux méthodes formelles [LdI01, BJT98, HKGM00], la plupart d’elles, bien qu’elles contribuent à un développement systématique des applications multi-agents, souffrent d’un manque d’aides pratiques et des issues méthodologiques dans la phase de conception.

Afin de tirer profit du potentiel d’adopter un processus rigoureux de conception, nous proposons une méthode, baptisée  $\mathcal{F}_{or}\mathcal{MAAD}$ , basée sur des raffinements successifs laissant développer une conception à partir des spécifications abstraites des besoins. Ces raffinements sont associés à un ensemble de principes et d’aides méthodologiques qui guident l’utilisateur pour construire par raffinement les aspects intra et inter-agents de manière systématique et incrémentale. La base de notre méthode  $\mathcal{F}_{or}\mathcal{MAAD}$  est le modèle conceptuel détaillé dans le travail de [LHKJ04] utilisant la notation Z [Spi92]. Étant donné ce modèle, nous procédons à décrire quelques directives dans le contexte du processus de développement afin de présenter comment nous pouvons atteindre les concepts définis à partir des besoins initiaux du système. Ainsi, nous intégrons la logique temporelle du premier ordre [MP92] dans les schémas Z [Spi92] afin de décrire l’évolution du système dans le temps. Ce multi-formalisme  $\mathcal{T}_{temporal}\mathcal{Z}$  nous permet de spécifier aussi bien les propriétés statiques que comportementales des SMA.

Le choix du langage de spécification permet la faisabilité du processus de vérification. Ce dernier consiste à prouver que sous certaines contraintes un système satisfait un ensemble de propriétés. Quand on considère un système (pris au sens large du terme), le langage permet d’exprimer pour un état donné du système, les propriétés de cet état. Par exemple, on peut donner une formule de la logique usuelle des entiers qui exprime qu’une variable contient un nombre premier. Cependant cette propriété peut-être vérifiée pour un état, mais peut être fausse pour un

autre état. Une propriété n'est, en général, pas toujours vérifiée. D'autre part, il peut être intéressant d'exprimer des relations sur les propriétés de différents états.

En effet, cette vérification est omise dans la plupart des approches existantes [RD00, DHK01, CSC04, KG96] ou, dans certains cas, nécessite une transition vers d'autres langages [MSH02].

Dans notre méthode  $\mathcal{F}_{or}\mathcal{MAAD}$ , nous proposons un processus de vérification compositionnel qui tient compte du processus de raffinement et qui accompagne les différentes étapes. Grâce au choix du langage de spécification Z et à l'intégration syntaxique et sémantique de la logique temporelle dans Z, le processus de conception peut être couplé avec un environnement qui supporte la vérification et le raisonnement tel que Z-EVES [MS99].

Une autre préoccupation dans ce travail est de développer un outil d'aide qui consiste à automatiser les différentes étapes de la méthode. Cette automatisation peut être totale ou partielle nécessitant l'intervention du concepteur. Nous initions le processus de développement par la description des besoins. Le passage d'une étape à une autre sera assuré de manière systématique en faisant recours à un outil de vérification syntaxique et sémantique.

En vue de s'assurer de la validité et la couverture de notre méthode, une phase essentielle dans ce travail est de l'illustrer avec des études de cas de domaines variés montrant différents aspects tels que la coopération, la négociation, la coordination d'action et l'organisation. Ainsi, nous avons développé trois études de cas. La première étude de cas concerne le jeu de poursuite. Le problème de jeu de poursuite connu sous le nom Proie Prédateurs constitue l'une des applications les plus utilisées en SMA qui met en évidence l'aspect coopération entre les différentes entités prédateurs. La deuxième étude de cas traite plutôt l'aspect négociation et ceci dans le cadre du contrôle de trafic aérien. Finalement, les aspects organisation et coordination d'actions sont abordés dans la troisième étude de cas, à savoir la gestion coopérative de pannes dans les réseaux informatiques.

Cette thèse se présente comme suit : le premier chapitre introduit les notions

d'agent et de SMA à travers un aperçu sur leur utilisation aussi bien dans les laboratoires de recherche que dans l'industrie.

Dans le deuxième chapitre, nous nous intéressons aux méthodes de développement des applications à base d'agents qui existent dans la littérature en présentant un survol de travaux existants suivi d'une étude comparative des méthodes formelles les plus connues.

Suite aux constatations recensées à partir de cette étude comparative, notre contribution se présente tout au long des chapitres 3, 4 et 5. Dans le chapitre 3, nous présentons notre langage  $\mathcal{T}_{temporal}\mathcal{Z}$  : un langage formel de spécification des applications à base d'agents. Il s'agit d'une intégration des opérateurs de la logique temporelle dans la notation  $\mathcal{Z}$ . Cette intégration porte aussi bien sur le niveau syntaxique que sémantique ce qui rend possible l'utilisation des outils de support de vérification de  $\mathcal{Z}$ .

Le chapitre 4 concerne la présentation de notre méthode  $\mathcal{F}_{or}\mathcal{MAAD}$  : une méthode formelle de développement des applications à base d'agents traitant aussi bien la phase de spécification des besoins que la phase de conception. Cette dernière s'intéresse aux deux niveaux qui caractérisent les SMA à savoir le niveau collectif et le niveau individuel à travers sept étapes de raffinement incrémental associées à un processus de vérification. Cette méthode est supportée par l'outil  $\mathcal{F}_{or}\mathcal{MAADTools}$  que nous avons développé afin d'assister le concepteur durant ces étapes successives.

Finalement, le chapitre 5 consiste à illustrer notre méthode à travers trois études de cas de domaines et de complexités variés. Dans le choix de ces études de cas, nous avons essayé d'aborder les différents aspects des SMA à savoir la coopération dans la première étude de cas, la négociation dans la deuxième et l'organisation et la coordination d'action dans la troisième. Cette diversification permet de s'assurer de la couverture des différents aspects par la méthode  $\mathcal{F}_{or}\mathcal{MAAD}$ .

# Chapitre 1

## Les systèmes multi-agents : de la recherche vers l'application

A la différence de l'Intelligence Artificielle (IA) qui modélise le comportement intelligent d'une seule entité, l'intelligence artificielle distribuée (IAD) s'intéresse à des comportements intelligents qui résultent de l'activité coopérative de plusieurs entités. Suite à la distribution de l'expertise sur un ensemble de composants qui communiquent pour atteindre un objectif commun ou résoudre un problème, il est nécessaire de décomposer le problème en sous-problèmes. Cette décomposition n'est pas toujours aisée. Ainsi, une extension des systèmes d'IAD est proposée : les entités doivent être capables de raisonner sur les connaissances et les capacités des autres dans le but d'une coopération effective. Pour ce faire, elles doivent être dotées de capacités de perception et d'action sur l'environnement et doivent posséder une certaine autonomie de comportement. On parle, alors, d'*agents* et par conséquent de *SMA* [AASD99].

Bien que les paradigmes *agent* et *SMA* représentent le sujet de plusieurs travaux de recherches, ils ont été adoptés dans plusieurs domaines d'applications montrant, ainsi, leur adéquation à résoudre plusieurs problèmes de domaines et de complexités variés.

Dans ce qui suit, nous proposons de citer des exemples, non exhaustifs, de certains concepts et aspects relatifs à ces deux paradigmes ainsi que de leur utilisation dans divers domaines d'application.

## 1.1 Les aspects des SMA

L'aspect *distribution* vient pour résoudre les problèmes de complexité des gros programmes monolithiques de l'intelligence artificielle : l'exécution est, alors, distribuée mais le contrôle reste centralisé. Au contraire, dans les SMA, chaque agent possède un contrôle total sur son comportement. En effet, pour résoudre un problème complexe, il est, parfois plus simple, de concevoir des programmes relativement petits (les agents) en interaction qu'un seul gros programme monolithique. L'autonomie permet au système de s'adapter, dynamiquement aux changements imprévus qui interviennent dans l'environnement.

D'une manière similaire, le génie logiciel a évolué vers des composants de plus en plus autonomes. Les SMA peuvent être vus comme la rencontre du génie logiciel et de l'intelligence artificielle distribuée, avec un apport très important des systèmes distribués. Par rapport à un objet, un agent peut prendre des initiatives, peut refuser d'obéir à une requête, peut se déplacer, etc.

### 1.1.1 Présentation des SMA

Le modèle de distribution qui s'applique aux SMA impose deux niveaux d'observation et d'analyse. Le premier niveau est celui de l'entité de base qu'on appelle *agent*. Un deuxième niveau, qu'on appelle *SMA*, concerne le système constitué par les agents et leurs interactions. Le premier niveau désigne l'aspect individuel. Le deuxième niveau s'intéresse, plutôt, à l'aspect collectif.

#### Aspect individuel

Dans la littérature, il n'y a pas une définition standard pour la notion d'*agent*. En effet, plusieurs définitions ont été présentées telles que “*un agent est une entité qui perçoit son environnement et agit sur celui-ci*” [RN03], “*un agent est un système informatique, situé dans un environnement, et qui agit d'une façon autonome pour atteindre les objectifs (buts) pour lesquels il a été conçu*” [JSW98], “*un agent est une entité qui fonctionne continuellement et de manière autonome dans*

*un environnement où d'autres processus se déroulent et d'autres agents existent*" [SY94], "*un agent est une entité autonome, réelle ou abstraite, qui est capable d'agir sur elle-même et sur son environnement, qui, dans un univers multi-agents, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec les autres agents*" [Fer95].

Comparant les définitions ci-dessus, nous pouvons identifier deux tendances principales dans la définition de l'*agent*. Certains chercheurs considèrent que nous pouvons raisonner sur un agent isolé, alors que d'autres considèrent les agents, principalement, comme entités agissant dans une société d'agents appelée *SMA*. Les deux tendances ont, déjà, porté des résultats. Nous pensons que c'est le paradigme des *SMA* qui va s'imposer comme prépondérant car il est plutôt difficile de considérer qu'un agent peut exister sans interagir avec d'autres agents (soit artificiels ou humains).

### **Aspect collectif**

On appelle *SMA* un système composé des éléments suivants [Fer95] :

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

de l'univers.

On dit que deux agents *coopèrent* s'ils contribuent à la réalisation d'une activité commune, après avoir fixé un objectif commun, afin d'accomplir les tâches difficiles (voire impossibles) à réaliser individuellement, améliorer la productivité des agents et optimiser l'utilisation des ressources. La coopération nécessite la mise en œuvre de mécanismes pour améliorer les performances du groupe afin d'accroître la survie de ses membres, mais elle nécessite en contre partie des structures sociales qui présentent des conséquences à la fois positives et négatives pour les agents.

Un aspect important pour faire coopérer des agents est de les *coordonner*. Par coordonner, on entend la gestion de tâches supplémentaires qui ne sont pas directement productives mais qui servent à améliorer l'accomplissement des activités qui le sont. La coordination est un aspect essentiel des SMA et une des raisons qui pousse à leur utilisation dans de nombreuses applications comme les systèmes d'information. Il existe plusieurs techniques de coordination, dont on peut citer la planification et la négociation.

Finalement, les différentes interactions entre deux ou plusieurs agents ne peuvent avoir lieu que grâce à la *communication*. Le rôle de base de la *communication* dans les SMA est de permettre des moyens d'échange d'informations (plans, résultats partiels, informations de synchronisation) afin d'enrichir la vue locale de chaque agent et de la faire étendre à une vue globale de son environnement.

### 1.1.2 Les méthodes de conception des SMA

Les SMA sont, souvent, des systèmes complexes qui demandent de longs efforts de développement. Du début des années 80 jusqu'au milieu des années 90, les efforts des chercheurs se sont, surtout, concentrés sur l'exploration de nouveaux concepts, la mise en place de théories et la réalisation de divers prototypes de SMA. Ce bouillonnement d'idées est tout à fait normal dans un jeune domaine technologique. Parmi les travaux de recherche, citons ceux de Decker et ses collègues [DL90] qui ont proposé un cadre de référence pour évaluer les recherches en résolution coopérative et distribuée de problèmes ("cooperative distributed problem solving"). Leur but était

de comprendre les relations entre des efforts de recherche et d'évaluer les progrès dans ce champ de recherche. Bien que ce cadre ne soit pas une méthode de conception, il souligna un besoin de prendre du recul par rapport au foisonnement des recherches. Werner [Wer89] proposa, lui aussi, un cadre de référence pour essayer de comprendre les "relations qui peuvent s'établir entre les usagers, les développeurs et le SMA en construction". Cette réflexion essaye de situer le développement traditionnel de systèmes informatiques par rapport au développement d'un SMA dans lequel les agents pourraient évoluer par eux-mêmes, choisissant leurs propres buts, créant leurs propres plans, etc. Ce cadre n'était pas une méthode de conception, proprement parlé, mais dénote une préoccupation méthodologique intéressante. Goodson et Schmidt [SSG78] ont proposé la première démarche méthodologique visant à décomposer un problème multi-agents en des unités fonctionnelles, d'identifier les unités fonctionnelles que la machine puisse traiter mieux que les usagers et de construire les modules de résolution de problèmes pour ces unités. Moulin et Cloutier [MC94] ont développé, quant à eux, la méthode MASB ("Multi-Agent Scenario-Based method"), une méthode de conception de SMA qui se base sur l'analyse et la conception de scénarios caractérisant les rôles joués par les agents humains et les agents artificiels et qui s'applique bien pour des applications de "travail collaboratif" (ex. système distribué de prise de rendez-vous). Burmeister [BDM97] a proposé une méthode dans laquelle on utilise trois modèles pour analyser un SMA : un modèle d'agent (les agents et leurs structures internes définies en termes de croyances, buts, plans, etc.) ; un modèle organisationnel (description des relations entre agents) ; un modèle de coopération (description des interactions entre agents). Cette méthode empruntait des techniques de modélisation aux méthodes orientées objet. Kinny et ses collègues [KG96] ont proposé une méthode qui distingue deux niveaux de conception (interne et externe) pour la création d'agents BDI. Les agents BDI mettent en oeuvre un modèle basé sur les notions de croyance (Belief), désir (Desire) et intention (Intention). Ce modèle a été popularisé par Rao et Georgeff [RG98]. Le niveau externe est spécifié par un modèle qui décrit les relations hiérarchiques entre agents et un modèle d'interaction qui définit les responsabilités, services et interactions entre les agents et leur environnement. Le niveau interne permet de modéliser chaque agent BDI grâce à trois modèles : un modèle de croyances, un modèle de buts et un

modèle de plans.

Récemment, nous assistons à une large gamme de méthodes d'analyse et de conception des SMA. La plupart d'entre elles s'appuient sur des techniques de modélisation empruntées à des méthodes connues en développement orienté objet ou en ingénierie des connaissances. Ces techniques de modélisation aident à la construction des modèles d'agents, de l'architecture du SMA et de la spécification des modèles d'organisation, d'interaction, etc.

La question qui se pose est : Peut-on s'attendre à la création d'une approche standard comme cela fut le cas en ingénierie des connaissances avec la méthode KADS [SW94] ou à une normalisation des modèles comme ce fut le cas en orienté objet avec le langage UML [BM04] ? Une telle standardisation apparaîtra, certainement, quand les technologies multi-agents seront adoptées, effectivement, par les applications industrielles.

### 1.1.3 Les SMA et la simulation

La simulation est utilisée dans divers domaines scientifiques quand la réalisation de l'ensemble des expériences concrètes nécessaires à la validation ou l'infirmité d'une théorie prendrait un temps trop grand ou demanderait des moyens dont on ne disposerait pas. Grâce aux résultats de la simulation, on peut ensuite orienter le choix des expérimentations à réaliser pour tendre vers les résultats désirés.

La modélisation est un processus particulier de simulation. Il consiste à donner une représentation simplifiée d'un objet, phénomène ou personne à modéliser, afin d'essayer de reproduire son comportement en simulation. Parmi les modèles informatiques, on trouve les modèles à base de SMA. Dans une modélisation par SMA, le phénomène est décrit de la perspective de ses constituants. On cherche ainsi à décrire des comportements globaux en ayant une représentation des agissements d'un ensemble d'entités autonomes et actives (les agents).

Grâce à ces SMA, on peut modéliser des systèmes réels dans lesquels des comportements très complexes émergent d'interactions relativement simples et locales entre de nombreux individus différents. De nombreux systèmes naturels ou créés

par l'homme sont complexes en ce sens - depuis l'économie, les sociétés, les normes culturelles et les embouteillages, jusqu'aux écosystèmes, conditions météorologiques, systèmes immunitaires et évolution - et les dynamiques non linéaires de ces systèmes sont difficiles à étudier en utilisant des modèles traditionnels.

Ainsi, les SMA sont le fruit de plusieurs disciplines présentant un tournant important pour l'informatique. Ce tournant peut être vu comme le passage de métaphores essentiellement basées sur des modèles centralisés, issues de la psychologie de l'individu par exemple, à des métaphores basées sur des modèles distribués, comme celles qu'on trouve en sociologie ou en psychologie sociale. Les raisons de ce passage, de modèles centralisés vers des modèles distribués, sont nombreuses. Parmi ces raisons, on peut citer :

- les problèmes que l'on cherche à résoudre sont, physiquement, distribués,
- les problèmes sont, fonctionnellement, très distribués et hétérogènes,
- la complexité des problèmes impose une vision locale,
- les réseaux imposent une version distribuée.

Le développement des réseaux, distribués par nature, des machines parallèles et l'évolution des langages à objets vers des langages à objets distribués ou langage d'acteurs ont, également, contribué à la mise en place de ces modèles.

Puisant ses fondements dans de nombreuses disciplines (e.g. le génie logiciel, l'intelligence artificielle, la programmation concurrente et répartie, etc.), la pluridisciplinarité de ce domaine fait sa richesse mais induit une grande complexité et une grande variété d'approches. Ainsi, de nombreux modèles d'agents, d'environnements, d'interactions et d'organisations sont élaborés et souvent combinés au sein d'un même SMA.

Ces modèles ont été appliqués, avec succès, pour la résolution de problèmes complexes et la simulation en vue d'expliquer les changements au sein d'un système naturel ou artificiel. En revanche, leur application dans des environnements complexes soulève de nombreux problèmes et fait encore l'objet de diverses recherches.

En effet, la complexité des nouveaux systèmes d'information (par exemple les systèmes accessibles depuis l'Internet), des applications traditionnelles vues

sous l'angle des SMA (logistique, transport, jeux, etc.) ainsi que des applications émergentes récentes (Grid computing, services Web, intelligence ambiante, ...) s'est trouvée considérablement accrue du fait de : leur distribution, la grande quantité d'informations qu'ils manipulent, leur aspect coopératif et adaptatif, et leur ouverture. Ces systèmes d'information et ces applications ont toutes les caractéristiques d'un domaine d'application des SMA. Ils permettent, en effet, de valider et de montrer les limites des modèles multi-agents proposés. De nombreux SMA opérationnels ont vu le jour. Les architectures logicielles découlant des modèles sont, donc, nombreuses et variées. Mais les systèmes résultants sont des systèmes dédiés à leurs applications. Or, un facteur qui déterminera la rapidité de diffusion industrielle et commerciale des SMA est la facilité avec laquelle ces nouveaux environnements complexes pourront être développés.

## 1.2 Les domaines d'application

Les SMA et les agents autonomes fournissent une nouvelle méthode pour analyser, concevoir et implémenter des applications de domaines récents de plus en plus complexes et distribués car ils sont caractérisés par la distribution tout en bénéficiant d'autres disciplines comme les sciences cognitives, la sociologie, et la psychologie sociale.

Aujourd'hui, la plupart des applications nécessitent de distribuer des tâches entre des "entités" autonomes (ou semi-autonomes) afin d'atteindre leurs objectifs d'une manière optimale. Puisque les approches classiques sont, en général, monolithiques et leur concept d'intelligence est centralisé, les applications actuelles sont établies à base de SMA.

Il y a, évidemment, plusieurs domaines d'application pour les agents dû au fait que les architectures basées sur les agents fournissent une manière bien particulière afin d'aborder, aisément, les problèmes. C'est pour cette raison que les agents sont, largement, utilisés dans des domaines tels que : l'automatisation des processus et de la production, la logistique, la gestion de réseaux, le commerce électronique, la recherche d'Information, la gestion du workflow, etc.

Vu la diversité de ces domaines, nous proposons de citer quatre catégories principales à savoir *les applications grand public*, *les applications coopératives*, *les applications de gestion* et *les applications temps réel*. Ces catégories ne sont pas, forcément, exclusives du fait qu'une application peut appartenir à plus qu'une catégorie de domaines.

Dans ce qui suit, nous citons, brièvement, quelque exemples de domaines à travers une liste non exhaustive d'applications variées.

### 1.2.1 Les SMA et les applications grand public

Pour montrer l'adéquation des SMA à répondre aux besoins des applications grand public, nous avons choisi de citer leur utilisation dans deux contextes, à savoir la recherche d'information et les télécommunications.

#### La recherche d'information

Une grande partie des applications de SMA est dans le domaine de recherche d'information dans des sources hétérogènes et réparties [CCDT01]. Dans de telles applications, les agents sont chargés de la collecte et du filtrage des informations, la gestion des ressources d'informations, la décomposition des tâches et du suivi du déroulement d'exécution des différentes sous-tâches.

Un *agent d'information* est défini comme le développement et l'utilisation efficace des entités informatiques autonomes, appelés les agents intelligents de l'information [Klu99], qui ont accès aux sources d'informations multiples et hétérogènes, et géographiquement distribuées comme dans l'Internet ou l'Intranet. La tâche principale de tels agents est d'exécuter des recherches afin de maintenir et négocier l'information appropriée au nom de leurs utilisateurs ou d'autres agents. Ceci inclut des qualifications telles que la recherche, l'analyse, la manipulation et la fusion de l'information hétérogène aussi bien que la visualisation et le guide de l'utilisateur. Ces agents fournissent, également, l'accès intelligent à une collection hétérogène de sources d'informations [Klu01].

Le nombre et la variété de sources et de services de données augmentent

considérablement chaque jour. Comme plus d'information devient disponible, il devient plus difficile d'accéder aux documents désirés. Le volume d'information disponible par l'intermédiaire de l'Internet et le Web représente un problème en soit. Le potentiel de l'Internet est passionnant mais la réalité est souvent décevante : car l'Internet est énorme et il n'est pas toujours facile de trouver la bonne information manuellement (ou même à l'aide des moteurs de recherche) [JSW98]. Le problème est, tout d'abord, que ces données fournies changent dans les formats et la représentation de structures (par exemple, les bases de données relationnelles) à semi-structuré (par exemple, email, newsgroup, pages de HTML) et à formats non structurés (par exemple, d'image de données) et la distribution de ces données dans le Web [SKL99]. D'autre part, il y a des facteurs humains et organisationnels derrière comme par exemple : nous nous sommes ennuyés par des temps de réponse lents, nous le trouvons difficile de lire le www rigoureusement, nous devenons fatigués, nous manquons facilement des choses, nous les comprenons mal, et d'ailleurs la structure sur Internet est seulement superficielle : (1) il n'y a aucune norme pour les "homepages", aucun mark-up sémantique pour vous dire ce qu'une page contient ; (2) la quantité de l'information mène à la surcharge de l'information. Ce que nous voulons est un genre de "secrétaire" : quelqu'un qui comprenne les éléments qui nous intéressent, (et les choses qui ne nous intéressent pas), qui peuvent agir comme un "Proxy", cachant l'information qui ne nous intéressent pas, et portant à notre connaissance l'information qui est d'intérêt. C'est là où les agents entrent dans le monde !

En résumé, aussi longtemps qu'Internet poursuivra son évolution, nous continuerons à être submergés par des données, sans que celles-ci soient toutefois structurées. La recherche d'information, dans ce cadre, devient une tâche difficile et les méthodes traditionnelles de recherche sur Internet ou sur des bases de données s'avèrent de plus en plus limitées. Les systèmes d'informations coopératifs, basés sur les agents logiciels, apportent de solutions prometteuses à cet épineux problème [CCDT01].

## Les télécommunications

Ces dernières années, les télécommunications ont, notamment, introduit une conception de services décentralisée dans le contexte du Web, créé de nouveaux services de médiation, tels que les portails et engendré l'apparition de nombreux fournisseurs de services réseaux qui ne disposent pas de leurs propres services réseaux [JCd02]. L'obtention de tels services décentralisés ne peut, bien entendu, être obtenue que grâce à des logiciels pour lesquels les données et le contrôle sont forcément distribués. De ce fait, il est clair que les SMA semblent convenir aux télécommunications. C'est pourquoi les principaux acteurs de télécommunications mènent, actuellement, d'intenses activités de recherche sur la technologie agent : British Telecom, France Télécom, Deutch Telekom, NTT, Nortel, Siemens, etc.

Dans ce cadre, nous citons, à titre d'exemple, un travail qui entre dans le cadre du projet réseau (Smart Net) de France Télécom [Gha92]. Le but du projet est d'introduire les techniques d'intelligence artificielle et des SMA dans la gestion et la supervision du réseau pour aider à traiter les alarmes et les notifications d'événements reçus par la plateforme de gestion du réseau. Actuellement, beaucoup de ces alarmes ont une utilité qui dépend de l'utilisateur et elles doivent être filtrées. D'autres événements deviennent significatifs quand ils sont associées à leurs contextes. Le contexte d'un événement renferme l'événement précédent, l'événement suivant et la date de détection de cet événement. La connaissance du contexte est, alors, nécessaire pour raisonner sur les événements, les actions et les changements à fin de pouvoir modéliser les faits (comme précédence, chevauchement, simultanéité) entre les événements.

### 1.2.2 Les SMA et les applications coopératives

L'objectif des recherches effectuées dans le domaine de la coopération et de la négociation entre agents est l'atteinte d'un état global du SMA en favorisant la synergie des agents. Ainsi, l'objectif peut être d'atteindre un état "meilleur", d'améliorer un résultat global tout en satisfaisant au maximum l'ensemble des résultats locaux. Parmi les équipes de recherche abordant la coopération dans les SMA, nous citons

l'équipe SMAC [COO].

On dira que plusieurs agents coopèrent ou encore qu'ils sont dans une situation de coopération si l'une des deux conditions est vérifiée :

1. l'ajout d'un nouvel agent permet d'accroître, différentiellement, les performances du groupe.
2. l'action des agents sert à éviter ou à résoudre des conflits potentiels ou actuels [Fer95].

De plus, l'introduction de nouveaux agents peut avoir des conséquences négatives sur le fonctionnement du groupe et les actions coopératives. Dans ce cas, les techniques de résolution de conflits proposent des solutions collectives à ces limitations de performances.

La négociation apparaît comme une technique de résolution de conflits. En effet, une situation de conflit peut être résolue, au sein d'un SMA, par le suivi de règles témoignant d'un arbitrage (dans le cas d'agents réactifs) ou bien l'obtention d'un accord résultant d'un processus de négociation (dans le cas d'agents cognitifs).

La coopération et la négociation entre agents connaissent des applications dans les domaines nécessitant l'accomplissement de tâches impossibles à réaliser seul, l'amélioration de la productivité de chacun des agents, l'augmentation de tâches réalisées dans un délai imparti, la diminution du temps de réalisation d'une tâche, l'amélioration de l'utilisation de ressources ou tout simplement le développement logiciel d'applications distribuées ouvertes et adaptatives [SN00].

Le domaine de l'informatique distribuée est très lié aux SMA et le problème de la décision y est un des problèmes majeurs. Les modèles de coopération et de négociation dans les SMA apportent des réponses à ce problème dans le domaine très récent du Grid Computing où des machines situées sur internet communiquent, se répartissent des tâches de calcul, de gestion de données, ... D'autre part, la majeure partie des applications actuelles résident dans le domaine du commerce électronique et des Web services [Kep02]. En effet, la négociation va permettre à des agents autonomes de pouvoir négocier les contrats pour les sociétés de commerce électronique. De plus, la coopération entre les workflows de plusieurs entreprises va permettre,

par exemple, d'accroître la rapidité et la sûreté d'une commande.

### **Les systèmes d'informations coopératifs (SIC)**

Les SIC sont, généralement, caractérisés par la grande variété et le grand nombre de sources d'informations. Ces sources d'informations sont hétérogènes et distribuées soit sur un réseau local (Intranet), soit sur l'Internet. De tels systèmes doivent être capables d'exécuter, principalement, les tâches suivantes :

- la découverte des sources : trouver la bonne source de données pour l'interroger ;
- la recherche d'informations : identifier les informations non structurées et semi structurées ;
- le filtrage des informations : analyser les données et éliminer celles qui sont inutiles ;
- la fusion des informations : regrouper les informations d'une manière significative.

Le SMA "Warren" [ZS96] pourrait constituer un exemple spécifique de l'utilisation des agents dans ce type d'application. C'est un système d'agents intelligents pour l'aide des usagers dans la gestion des portefeuilles. Ce système combine les données du marché financier, les rapports financiers, les modèles techniques et les rapports analytiques avec les prix courants des actions des compagnies. Toutes ces informations sont, déjà, disponibles sur le Web ; "Warren" ne fait que les intégrer via des agents spécialisés, les agents d'information, et ensuite les présenter aux usagers. Pour ce faire, "Warren" dispose de six agents ressources, deux agents de tâches et un agent utilisateur pour chaque usager. L'agent utilisateur affiche (via le web) les informations financières de son usager, lui permettant de faire des simulations d'achat et de vente des actions. Il affiche, également, les prix courants des actions et les nouvelles informations du marché financier. Le même agent permet, également, d'accéder aux rapports produits par les deux agents de tâches. Ces deux agents fournissent d'une part, une intégration graphique des prix et des nouvelles concernant les actions et, d'autre part, une analyse fondamentale des actions en tenant compte de leurs historiques. Les agents d'informations accèdent à différentes sources d'informations,

comme les pages Web. “Warren” n’est qu’un exemple et il existe, actuellement, plusieurs autres systèmes qui touchent à ce genre d’application. Parmi ces applications, nous pouvons citer :

- Infosleuth [NFK<sup>+</sup>00] : C’est un système multi-agent pour la recherche coopérative d’informations dans des bases de données distribuées. Ce système a été appliqué aux domaines médicaux.
- NetSA (pour “Networked Software Agents”) [CT98, Cot99] : C’est un système proche de Infosleuth et dédié aux environnements riches en informations.
- UMDL [UMD] : C’est un système d’informations coopératif pour la recherche des documents dans une librairie digitale.

## **Le Workflow**

Parmi les systèmes informatiques qui assistent les gestionnaires de grandes compagnies dans leur processus de gestion d’affaires (BPM) ou de prise de décision, nous pouvons citer le workflow [Woo00], qui est un système informatique de suivi automatique d’information entre les différents utilisateurs du système à travers le réseau. Les systèmes de gestion du workflow traditionnels sont gouvernés par des processus automatisés qui définissent le flux de travail partout dans les organisations. Cependant, tels processus n’ont qu’une vision locale, ils n’en possèdent pas les méta-données pour avoir une vision globale. En outre, ils ne sont pas conçus pour utiliser ou comprendre des ontologies ; ils ne sont pas dotés d’autonomie ou de comportement coopératif.

Vu que le workflow se caractérise par son aspect distribué et nécessite certaines capacités présentes chez les agents, il favorise, donc, une conception par agents.

Ainsi, le Workflow peut être vu comme une activité de résolution coopérative de problème. “Une résolution coopérative de problème se produit quand un groupe d’agents autonomes choisit de travailler ensemble pour accomplir un but commun”. Quand un problème coopératif se produit, chaque agent de la société doit reconnaître que le meilleur scénario pour résoudre ce problème est de coopérer avec les autres agents.

### 1.2.3 Les SMA et les applications de gestion

Le paradigme SMA s'applique, par exemple, pour la surveillance d'un processus industriel où elle met en œuvre la solution qui consiste à coordonner plusieurs surveillants spécialisés, plutôt qu'à envisager un seul surveillant omniscient.

Les recherches fondamentales concernent la représentation de la décision des agents, ou les protocoles de communication. Elles s'appliquent principalement aux télécommunications, à l'Internet avec le commerce électronique, à des agents physiques tels les robots ainsi qu'à l'optimisation des systèmes de transports.

#### **La gestion de grandes compagnies**

Pour pouvoir prendre une décision, les gestionnaires de grandes compagnies ont besoin de regrouper les avis et les informations provenant de plusieurs départements. Idéalement, toutes les informations pertinentes devraient être rassemblées avant qu'une décision ne soit prise. Cependant, le processus pour obtenir des informations, qui sont à jour et pertinentes, est très complexe et prend énormément de temps. Pour cette raison, plusieurs compagnies ont cherché à développer des systèmes informatiques afin de les assister dans leur processus d'affaires. A titre d'exemple, nous pouvons citer ADEPT [AGJ<sup>+</sup>94] qui aborde ce problème en considérant le processus d'affaires comme un ensemble d'agents qui négocient et qui offrent des services. Chaque agent représente un rôle distinct ou un département de l'entreprise et est en mesure de fournir un ou plusieurs services. Les agents qui requièrent les services d'autres agents le font par une négociation qui permet d'obtenir un coût, un délai temporel et un degré de qualité qui sont acceptables aux deux parties. Le résultat d'une négociation terminée avec succès constitue un engagement entre les deux parties.

#### **La gestion de production**

L'approche SMA a été appliquée aux systèmes de production [KP95]. Nous pouvons citer leur mise en œuvre dans :

- la conception et l'ingénierie simultanée [KP95] : il s'agit de développer les meilleurs produits en un temps court. A cet effet, toutes les étapes du cycle de vie du produit (définition du cahier des charges, conception, élaboration des méthodes, mises en fabrication, ...) sont considérées aussi tôt que possible.
- le système d'information [Woo00] : il est de plus en plus prouvé que la productivité des entreprises est plus limitée par l'information que par le travail ou le capital. Un exemple de gêne à la circulation de l'information provient des logiciels des différentes parties de l'entreprise qui ne peuvent pas "discuter" entre eux. Dans l'approche SMA, chacune de ces parties peut être vue comme un agent.
- la distribution du système de production (multi-sites) [KP95] : chaque site de production est autonome, mais doit tenir compte des décisions des autres sites.
- la gestion de la production [KP95] : le problème consiste à faire cohabiter des objectifs qui ont des termes plus ou moins longs.
- l'ordonnancement et le pilotage d'un système de production [Tra00] : il s'agit de comprendre comment ré-ordonnancer dynamiquement un atelier qui subit des événements perturbateurs. Ce ré-ordonnancement est une résolution coopérative et distribuée de problèmes entre les différentes entités de l'entreprise. La résolution part de la plus petite entité (le poste de travail), et tente de résoudre le problème en mettant en jeu progressivement de plus en plus d'entités (machines identiques, puis atelier, etc. ).
- les chaînes logistiques [TB96] : ce sont des réseaux de fournisseurs, d'usines, d'entrepôts, de centres de distribution et de détaillants à travers lesquels des matières premières sont acquises, transformées et livrées au consommateur. L'objectif est d'optimiser les performances de ces chaînes logistiques. Pour cela, les niveaux de décision stratégique, tactique et opérationnel sont mis en jeu pour que chaque maillon opère de manière intégrée à l'ensemble de la chaîne.
- les transports [FCDM<sup>+</sup>99] : le domaine du transport a, souvent, inspiré des travaux en intelligence artificielle et plus précisément dans la communauté multi-agent. Les propriétés d'autonomie, d'aptitudes sociales et de réactivité qui sont, usuellement, associées aux agents intelligents sont une réponse adaptée aux difficultés d'un domaine dont un des objectifs est l'amélioration de la

coordination entre entités, physiquement, distribuées. A titre d'exemple, une approche multi-agent a été appliquée pour résoudre les problèmes de planification et de négociation dans les systèmes de transport de marchandises par camions. Les conducteurs de camion et leur compagnie respective y sont vus comme des agents intelligents, autonomes et rationnels. L'objectif de l'étude est d'atteindre un compromis qui minimise le coût global des transports en appliquant, successivement, différentes méthodes de coopération. A cet effet, les conducteurs négocient à travers leur compagnie leurs propositions en effectuant des ventes et des achats de commandes.

### **L'auto-réparation**

La puissance de calcul est de plus en plus disponible sous la forme de machines en réseau (grappe, grille). Les applications réparties large échelle utilisent un grand nombre de ressources sur des durées qui rendent la probabilité de pannes non négligeable.

L'auto-curatif (Self-Healing) dénote la capacité du système à examiner, trouver, analyser et réagir aux dysfonctionnements. Les composants de tel système doivent être capables d'observer les pannes du système, d'évaluer les contraintes imposées par l'environnement extérieur et d'appliquer les corrections appropriées. Afin de découvrir automatiquement les dysfonctionnements du système ou les futures pannes possibles, il est nécessaires de savoir le comportement du système attendu. De tel système doit posséder des connaissances sur son propre comportement et il doit, aussi, avoir suffisamment de connaissances pour déterminer si le comportement courant est consistant et attendu en tenant compte de l'environnement.

Les SMA, caractérisés par leur autonomie, perception, distribution et coopération, se trouvent adéquats pour assurer l'auto-réparation. Ainsi, les agents représentent les composantes dont chacune sera chargée d'une partie des services et coopèrent avec les autres agents pour assurer la qualité demandée.

### 1.2.4 Les SMA et les applications temps réel

L'application des SMA dans un environnement temps réel ouvre de très intéressantes perspectives quant à l'élaboration et le développement de systèmes, réputés complexes et restreints, qui sont les *systèmes temps réels*. Ce développement, pour qu'il soit efficace et pratique, doit prendre en compte les caractéristiques intrinsèques des systèmes temps réel et, plus spécialement, les contraintes temporelles. De tels systèmes trouvent leur utilité dans de nombreux secteurs de l'industrie comme : l'automobile, l'aéronautique, les systèmes dits de contrôle-commande, la robotique, etc. Il est, donc, tout naturel qu'on ait cherché à utiliser les agents pour l'élaboration de ces systèmes.

Un agent dit *classique* (par opposition à agent *temps réel*) est une entité située dans un environnement ouvert particulier et capable d'actions autonomes visant à atteindre un objectif particulier. Un agent temps réel doit atteindre ses objectifs avec en plus des contraintes de temps qui sont susceptibles d'altérer la qualité de ces résultats. Un SMA temps réel doit fournir des services qui permettent aux agents temps réel de communiquer, coopérer et de se coordonner dans le but de vérifier leur objectif et les contraintes temps réel (qualité de service) spécifiées.

Dans ce qui suit, nous citons quelque exemples d'utilisation du paradigme SMA dans des applications temps réel à savoir la médecine, le commerce électronique et le contrôle du trafic aérien.

#### La médecine

Le modèle des soins d'un patient dans une unité de soins intensifs est, essentiellement, celui d'une équipe où un ensemble d'experts dans des domaines distincts coopèrent pour organiser les soins des patients. Le facteur le plus important pour donner de bons soins aux patients est le partage d'informations entre les membres de l'équipe de soins critiques. Une solution possible est de répartir le suivi des patients à un certain nombre d'agents de trois types différents. Les agents perception/action sont responsables de l'interface entre le SMA et le monde environnant, établissant la relation entre les données des senseurs et une représentation symbolique que le

système pourra utiliser, et traduisant les requêtes d'action du système en commandes pour les effecteurs. Les agents en charge du raisonnement sont responsables d'organiser le processus de prise de décision du système. Finalement, les agents en charge du contrôle (il n'y en a habituellement qu'un seul) assurent le contrôle de haut niveau du système [RHW<sup>+</sup>89].

### **Le commerce électronique**

Le commerce électronique (e-commerce) signifie les échanges qui se passent via Internet. D'où, les acheteurs et les vendeurs deviennent des entités électroniques. Le commerce électronique devient de plus en plus une réalité. Or, à chaque fois qu'un vendeur ou un client souhaite échanger des produits via Internet, il épuise plusieurs heures pour trouver une bonne solution car l'information est décentralisée. Donc, la négociation joue un rôle important pendant les échanges des produits ou des services. La capacité de négociation qui caractérise les SMA les rend, particulièrement, utiles pour le commerce électronique qui est riche en termes d'information et de processus. En résumé, il nous faut un système qui cherche les produits auxquels nous nous intéressons (une classification selon certains critères), et qui négocie avec les différents marchands d'une manière automatique. Les applications agents correspondent bien à ce besoin.

En outre, les enchères électroniques ou la bourse en ligne sont des systèmes qui montrent un caractère autonome (mise à jour automatique, prise de décision) ainsi que des contraintes sur quand et où les actions doivent être exécutées. Ce genre d'application est, typiquement, multi-agents et temps réel [DFWN<sup>+</sup>01].

### **Le contrôle du trafic aérien**

La progression très importante du trafic aérien dans les dernières décennies entraîne la congestion du ciel et alourdit la tâche des contrôleurs pour la gestion des conflits aériens. Pour cette raison, plusieurs projets ont été développés afin de résoudre le problème de conflit aérien en utilisant différentes méthodes et approches. Certains s'intéressent à améliorer le contrôle actuel en essayant d'éviter les

inconvénients préexistants ou de les limiter ; d'autres expriment leur refus total du contrôle centralisé créant ainsi de nouvelles approches indépendantes du contrôle de la base aérienne.

Dans ce contexte, certains projets se sont basés sur le concept agent, puisque ce dernier est apte à résoudre les problèmes complexes à cause de leur organisation et de leurs comportements collectifs. Dans ce principe, Henry et Tim [HH00] présentent un programme permettant à un agent autonome de surveiller le voisinage et de proposer la solution possible pour l'opérateur humain. Ce dernier peut ne pas interagir pendant un temps bien déterminé et la solution sera alors exécutée, comme il peut interrompre l'agent et publier sa propre commande. Dans [NDD04], les auteurs présentent une implémentation d'un simulateur multi-agents qui modélise chaque acteur (pilote, contrôleur, gestionnaire de flux) en tant qu'agents qui interagissent entre eux selon un modèle générique. Alors que John et Robert [WS99] proposent une nouvelle approche de coordination d'agents qui assure la sûreté et fournit à des agents une plus grande liberté pour optimiser ces opérations. Cette approche, dans laquelle les avions sont modélisés en tant qu'agents qui se coordonnent afin de résoudre le conflit, est appliquée avec succès à la gestion du trafic aérien.

### 1.3 Synthèse

Nous avons vu, tout au long de ce chapitre, que la technologie agent et multi-agent n'est pas un concept voué à rester sur les tablettes des laboratoires de recherche puisque plusieurs exemples d'applications existent déjà. Les personnes qui ont développé des SMA disent, toutefois, qu'il est difficile de concevoir et de bâtir un SMA. En effet, la construction de ce type de système comportent toutes les difficultés inhérentes aux systèmes répartis, auxquelles s'ajoute le caractère flexible des interactions entre agents. En plus, les concepteurs de SMA font face, de nos jours, à deux difficultés majeures : tout d'abord, l'absence de méthode systématique qui permet de spécifier et de structurer une application multi-agent, ensuite, le manque d'outils commerciaux pour construire des SMA.

Une des forces importantes derrière la croissance rapide que connaissent, aujourd'hui, les SMA est l'Internet où la population d'agents est sans cesse croissante. Ces agents devront savoir collaborer afin d'atteindre les objectifs de leurs concepteurs. Dans ce type d'environnement, les agents rencontrent deux défis majeurs : ils doivent être en mesure de se rencontrer et inter-opérer. Une première solution à ce problème est l'introduction d'agents intermédiaires (brokers, facilitateurs, etc.) [CT98, Cot99, Tro99]. Dans cette optique d'interopérabilité, un grand nombre de langages de communication pour les agents ont été développés ; basés, pour la plupart, sur la théorie des actes du discours. Bien que les performatives offertes par ces langages permettent de caractériser les types des messages, ils ne permettent pas encore aux agents de comprendre, explicitement, les concepts "discutés". Le problème des ontologies reste, donc, ouvert. Un autre problème critique est l'allocation de ressources limitées à un bon nombre d'agents. Des mécanismes basés sur les principes économiques ont été proposés pour résoudre ce problème. Dans de telles approches, on suppose que les agents sont égo-centrés et ne cherchent qu'à maximiser leur utilité. Les sous-domaines où on a appliqué ces principes économiques sont l'allocation de ressources, l'allocation de tâches et la négociation. Là aussi, la recherche n'a fait qu'entrevoir des techniques rudimentaires basés sur les lois du marché et il reste bien des domaines à couvrir comme par exemple les ventes aux enchères, les comportements acheteur(s) vendeur(s), le partage du marché, etc. Finalement, il convient de préciser que les chercheurs travaillant sur les SMA d'un point de vue formel se sont, presque toujours, heurtés à des agents omniscients, c'est-à-dire des agents ayant des capacités de raisonnement illimitées. Là, aussi, le problème est très ouvert et il y a pas mal d'équipes qui y travaillent tels que DESIR [DES] et INRIA [INR]. Comme nous pouvons encore le constater, le domaine des SMA demeure un domaine très ouvert pour la recherche.



# Chapitre 2

## État de l'art

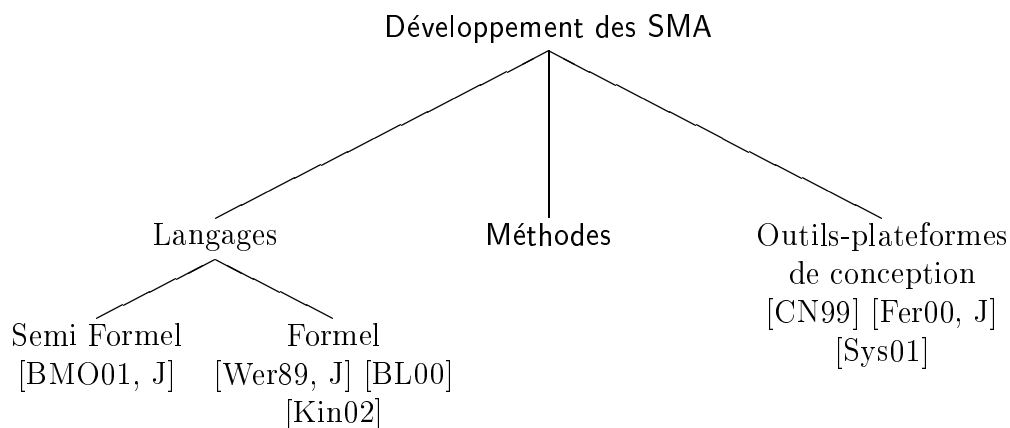


FIG. 2.1 – Aperçu sur la Conception des SMA

Le développement orienté agent est une approche prometteuse pour concevoir les systèmes logiciels complexes qui constituent maintenant une partie de l'infrastructure de la société moderne. L'approche agent est appropriée pour développer des applications industrielles dans divers champs, tels que la gestion des réseaux distribués [SN00] et les services WEB intelligents [Kep02]. La puissance de l'approche agent dans la résolution des problèmes complexes résulte du fait que les agents, grâce à leur autonomie et adaptabilité, peuvent réaliser leurs buts d'une manière flexible en se servant de *l'interaction* en terme de langages de haut niveau et de protocoles [ZO04]. Cependant, cette flexibilité dans la résolution de problèmes montre des difficultés dans la conception des agents et des sociétés d'agents. En effet, les

concepteurs doivent traiter de nouveaux concepts liés aux différents aspects individuels des agents, tels que l'*autonomie*, le *but*, la *connaissance*, les *capacités* et le *rôle*; ainsi que des concepts résultant des aspects collectifs et sociaux, tels que l'*organisation*, la *coopération*, la *négociation* et la *coordination des actions*.

Nous pouvons classer les travaux de recherche qui se sont consacrés au développement des SMA en trois axes (voir la figure 2.1).

Une première communauté de chercheurs ont présenté certains modèles décrivant les aspects des SMA. Leurs travaux ont cherché à proposer des formalismes pour décrire d'une part, l'aspect individuel d'un agent [KHCM02] [Wer89] qui se base principalement sur les notions de *goal* et *intention*; et d'autre part, l'aspect collectif des SMA qui se manifeste par les différentes formes d'interaction qui peuvent avoir lieu au sein d'un SMA [Woo00]. Les langages qui ont été proposés dans ce contexte sont orientés comportement et ont servi dans la description des protocoles pour certains aspects des SMA (coordination, communication, ...) tout en négligeant les attributs et les propriétés (concepts). Ainsi, ces langages souffrent d'un manque de couverture. Quant à l'aspect collectif, il existe plusieurs tentatives de Formalisation et de définition de langages de conception mais la plus part de ces tentatives n'offrent pas une méthode.

La deuxième communauté a proposé des plate-formes multi-agents permettant de faciliter l'implémentation et l'exploitation des SMA et de réduire le coût de développement. Leur but est de développer la plupart des composants de la technologie d'agent (communication, coopération etc.) pour permettre au développeur de résoudre les problèmes de son application. Plusieurs plate-formes ont été développées par des groupes de recherche universitaire et d'autres par l'industrie. Parmi les plate-formes les plus connues et utilisées nous citons AgentBuilder [Sys01], MadKit [Fer00], ZEUS [CN99] et Swarm [MBLA96].

Finalement, les derniers travaux de recherche ont essayé de faire face à la complexité de développement des systèmes à base d'agents en proposant les méthodes de développement appropriées. La majorité des solutions proposées sont des méthodes

informelles ou semi-formelles telles que Gaia [WJK00], Multi-agent System Engineering (MaSE) [DW01], Tropos [GMP02], Prometheus [PW02], Role Oriented Analysis and Design for Multi-Agent Programming (ROADMAP) [TPS02]. D'autres méthodes sont, plutôt, formelles telles que RIO [HKGM00], DESIRE [BJT98] et la méthode de Luck et d'Inverno [LdI01].

Dans ce travail, nous nous intéressons plus aux méthodes de conception des SMA. Par conséquent, dans ce qui suit, nous présentons, en premier lieu, des exemples représentatifs de chaque catégorie (semi-formelle, formelle) de méthodes de conception des applications multi-agents. En deuxième lieu, nous présentons une étude comparative de ces principales méthodes en se basant sur des critères que nous jugeons importants.

## 2.1 Les méthodes de conception des applications multi-agents

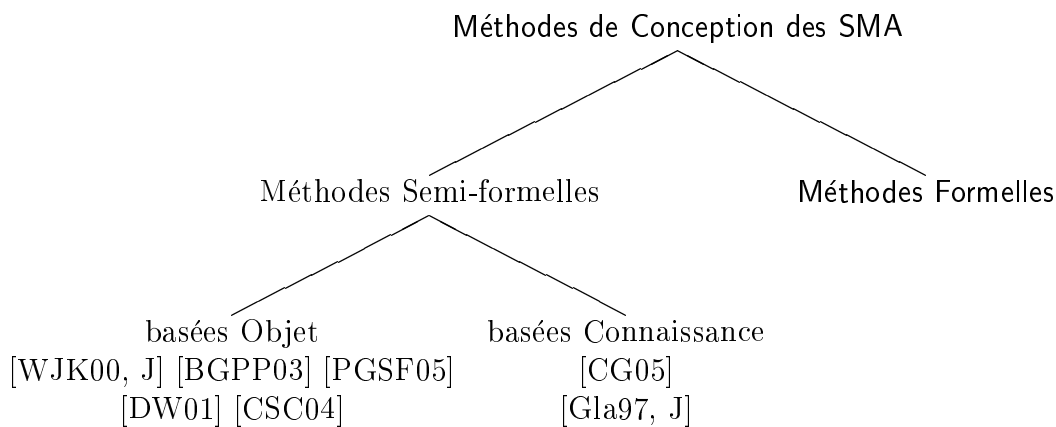


FIG. 2.2 – Aperçu sur la classification des méthodes existantes

La majorité des méthodes proposées pour la conception des SMA sont semi-formelles (voir figure 2.2). Nous distinguons celles qui représentent des extensions des méthodes orientées objet basées sur UML, comme ADELFE [BGPP03], AOAD [WJK99], MASB [BM96] et MaSE [DW01], et celles qui étendent des méthodes à base de connaissance, comme MAS-CommonKADS [CG05] et CoMoMAS [Gla97].

Ces méthodes, basées sur les notations semi-formelles, ne permettent pas un raisonnement formel sur les propriétés désirées.

D'autres approches ont adopté des méthodes formelles (voir figure 2.3), par exemple, le framework proposé par Luck et d'Inverno, qui utilise la notation Z [LdI01] et Concourant-Metatem [Fis94], basé sur la logique temporelle. A part le fait que ces travaux ne proposent aucune démarche de conception fixant les étapes et les passages entre elles, la plupart ignorent le raisonnement formel et en particulier la *vérification*.

### 2.1.1 Les méthodes semi-formelles

Plusieurs méthodes semi-formelles ont été proposées pour le développement des SMA. Nous distinguons des méthodes qui se basent sur la notion d'objet en l'étendant par des connaissances et des mécanismes d'interaction [WJK00, WJK99, DW01, BGPP03, PGSF05]. D'autres méthodes semi-formelles se préoccupent de présenter les connaissances employées pour décrire le comportement des agents [CG05, Gla97].

#### Les méthodes à base d'objets

Ces méthodes se basent sur la technologie objet avec des extensions pour pouvoir spécifier les agents. Pour décrire cette catégorie de méthodes, nous avons choisi de détailler INGENIAS [PGSF05], PASSI [CSC04] et ADELFE [BGPP03] vu qu'elles sont parmi les méthodes les plus complètes et les plus outillées.

#### INGENIAS

INGENIAS [PGSF05, PAV06] cherche à intégrer des résultats de différents domaines de recherche sur les agents. Pour y parvenir, INGENIAS définit un ensemble de méta-modèles. Ces méta-modèles décrivent différents aspects du SMA, concrètement : l'environnement, les interactions, les organisations, les agents et leurs buts et tâches. Le processus d'intégration consiste à faire évoluer ces méta-modèles

en les détaillant ou en incluant de nouveaux concepts.

INGENIAS élargit la proposition de MESSAGE [CCG<sup>+</sup>01] en définissant avec plus de détails les activités d'analyse, conception et implémentation pour élaborer un guide méthodologique aux développeurs de systèmes d'agents. Cet aspect est important car les concepts sont très divers et parce que le degré de complexité d'un système avec des agents multiples exige une définition soignée de l'organisation et des interactions de SMA.

Pour que la méthodologie soit applicable, le support d'outils est nécessaire. Les outils devraient aider à l'édition des spécifications des SMA par l'utilisateur, permettre de traduire les spécifications en composants logiciel, valider les modèles et générer la documentation. La spécification de SMA se construirait en utilisant des éléments définis dans les méta-modèles. Cela amène à définir le processus de génération partielle de code à partir des spécifications de modèles de SMA. Pour cela on propose la définition des schémas ou frameworks de SMA qui peuvent être spécialisés et configurés à partir de ces spécifications. Ces schémas peuvent être désignés pour la réalisation du SMA sur une plate-forme d'agents, comme Jade [BPR01], qui offrent les services basiques pour l'implantation de ce type de systèmes.

### **ADELFE**

ADELFE [BGPP03] (Atelier de Développement de Logiciels à Fonctionnalité Emergente) se centre sur les aspects adaptatifs et auto-organiseurs des SMA. Il propose une notation qui étend UML et une extension du Processus Unifié pour prendre en compte la modélisation et la réalisation de systèmes SMA adaptatifs. Son originalité réside dans le fait qu'il se focalise sur les aspects de dynamique en guidant le concepteur pour la réalisation de systèmes multi-agents à fonctionnalité émergente. Ce processus permet de prendre en compte l'ensemble des activités du cycle de vie en V du logiciel (recueil des besoins, analyse, conception, implémentation et tests) sous la forme de modèles UML. Le travail est facilité avec l'outil OpenTool [GP03].

L'organisation de SMA n'est pas spécifiée d'une façon explicite, mais elle est conséquence des interactions entre les agents. La méthode spécifie comment les

phases d'identification, de découpage du système en classes et composants peuvent être adaptées à une décomposition en agents auto-organiseurs.

Dans une première étape, un outil interactif permet au concepteur de savoir si un système multi-agent adaptatif est utile pour l'application proposée. Cet outil vérifie la pertinence de l'utilisation de la méthode. L'étape suivante consiste à adapter les phases d'une méthode orientée objet au domaine des agents adaptatifs.

La phase d'analyse comporte l'étape classique d'expressions des besoins c'est-à-dire exprimer ce que l'utilisateur souhaite que son système fasse. L'expression des besoins consiste à définir les fonctions globales que doit réaliser le système. On commence ensuite à définir l'environnement avec lequel le système va interagir. En effet, le système adaptatif ne pourra apprendre que s'il a des interactions avec son environnement. Pour ce faire, il faut s'intéresser à la description de l'environnement de manière exhaustive, c'est-à-dire déterminer ce qu'il est capable de faire, non seulement en ce qui concerne ses interactions avec les différents agents ou le système dans son ensemble, mais aussi ses comportements propres. La deuxième étape de cette phase d'analyse permet d'identifier les entités qui interviennent dans le comportement du système, entités qui seront ou non identifiées comme des agents dans la troisième et dernière étape de cette phase d'analyse.

La phase de conception consiste à donner un comportement aux agents et à l'environnement du SMA. Pour la conception d'un agent, ADELFE propose une description d'agent générique composée de sept modules : un module de communication avec les autres agents, un module de communication avec l'environnement, un module de croyances sur lui-même, un module de croyances sur les autres agents, un module de croyances sur son environnement, un module de compétences, un module d'attitude sociale lié à la coopération. Le comportement de chaque agent est décrit par le résultat des informations contenues dans ces sept modules.

## **PASSI**

PASSI [CSC04] (Process for Agent Societies Specification and Implementation) propose un processus itératif et incrémental pour passer des besoins au code, à travers cinq modèles. La notation pour décrire tous les modèles est UML mais avec

de nouveaux éléments des concepts d'agent qui sont spécifiés dans un méta-modèle structuré en trois niveaux d'abstraction différents.

Ainsi, le processus de PASSI est composé de cinq modèles : (1) le modèle des besoins du systèmes composé de quatre phases à savoir la description des besoins du domaine, l'identification d'agent, l'identification de rôle et la spécification de tâches; (2) le modèle de la société d'agents qui décrit les interactions sociales et les dépendances à travers les agents. Le développement de ce modèle nécessite trois étapes à savoir la description d'ontologie, la description de rôle et la description de protocole; (3) le modèle d'implémentation d'agent qui est un modèle classique de l'architecture de la solution en termes de classes et méthodes. Ce modèle est composé de la définition de la structure d'agent et la description du comportement d'agent; (4) le modèle de code qui nécessite la génération de code à partir du modèle grâce au PASSI ToolKit (PTK). Il est possible de générer non seulement les squelettes mais aussi des parties largement réutilisables de code des méthodes basées sur une librairie de patrons réutilisables et les descriptions de conception associées gérées par l'application AgentFactory; et (5) le modèle de déploiement qui est un modèle de distribution des parties du systèmes sur les unités matérielles et de leur migration entre ces unités.

### **Les méthodes à base de connaissances**

Les méthodes à base de connaissances fournissent des techniques pouvant prendre en compte l'état mental des agents. Plusieurs méthodes pour la modélisation des SMA, étendant la méthode CommonKADS [SWdH94], ont été proposées (CommonKADS est la méthode standard de modélisation des connaissances).

Parmi ces méthodes, nous proposons de présenter CoMoMAS [Gla97] comme étant un exemple clair et représentatif de cette catégorie de méthodes.

#### **CoMoMAS**

Dans la méthode CoMoMAS [Gla97], un agent est vu comme une entité ayant des compétences réactives, cognitives, coopératives et/ou sociales. Cette méthode

utilise la syntaxe de CML (Conceptual Modeling Language) [SW94] pour décrire ses différents modèles qui sont :

- le modèle d’agent décrit l’architecture et la structure des connaissances des agents (connaissance sociale, coopérative, de contrôle, cognitive et réactive) ;
- le modèle d’expertise décrit les compétences cognitives et réactives des agents ;
- le modèle de tâche décrit la décomposition des tâches du système et indique si ces tâches sont exécutées par un humain ou un agent informatique ;
- le modèle de coopération décrit la coopération entre agents, en utilisant les méthodes de résolution des conflits ;
- le modèle du système définit les aspects organisationnels de la société des agents et leurs aspects architecturaux ; et
- le modèle de conception intègre les modèles précédents dans un système global. Il décrit les exigences pour la conception.

La méthode CoMoMAS couvre presque toutes les étapes du processus de développement. De plus, CoMoMAS est supportée par le modèle incrémental et possède une approche mixte.

CoMoMAS est conçu avec la réutilisabilité et pour la réutilisabilité, c’est à dire que ses modèles sont réutilisables et sont conçus en utilisant des bibliothèques existantes.

CoMoMAS peut représenter les systèmes distribués. Le modèle du système définit les structures organisationnelles de la société que forment les agents.

Dans CoMoMAS, le mode de communication peut être direct, synchrone et asynchrone. Les langages de communication peuvent être basés sur les signaux (agents réactifs) et sur les actes de discours (agents cognitifs). On note l’utilisation dans le modèle de coopération de la méthode CoMoMAS des concepts de négociation, de délégation de tâche et de planification multi-agents.

CoMoMAS comme toutes les méthodes à base de connaissances possèdent une librairie d’outils pouvant être utilisés. Cependant, ces méthodes ne peuvent pas modéliser le comportement social des agents dans un SMA. En plus, nous notons l’absence de guides qui accompagnent le processus de conception.

## Synthèse

La majorité des méthodes proposées pour la conception des SMA sont semi-formelles. Ces méthodes suivent les principales phases : *spécification des besoins*, *analyse* et *conception*. En outre, elles partagent ensemble certains modèles construits durant ces phases mais elles ne couvrent pas simultanément les deux aspects collectif et individuel. Le premier aspect est défini en terme de coopération, organisation et interaction. Dans ce contexte, certains travaux se sont intéressés à l'organisation comme une caractéristique principale des SMA tels que AALAADIN [GF00], MAS-CommonKADS [CG05], CoMoMAS [Gla97] et ADELFE [BGPP03] décrivant la structure organisationnelle. La coopération, dans Gaia [WJK00], est sommairement définie. Par contre, dans d'autres méthodes telles que PASSI [CSC04] et CoMoMAS [Gla97], certains concepts ont été gérés dans le contexte du modèle de coopération tels que la négociation, la planification et la délégation de tâches. Concernant l'interaction, nous avons noté que certaines méthodes permettent une interaction dynamique telles que ADELFE [BGPP03], PASSI [CSC04] et MAS-CommonKADS [CG05]. L'aspect individuel, contrairement au collectif, nécessite un accord sur une définition universelle des principaux concepts tels que l'*état mental*.

Malgré l'utilisation de quelques notations graphiques ou semi-formelles, ces méthodes restent difficiles à utiliser dans l'industrie. Tout d'abord, la plupart entre elles souffrent d'un manque de couverture au niveau individuel ou collectif. De plus, les processus de conception ne définissent pas, dans la plupart des cas, des étapes claires et la manière de passer d'une étape à une autre. Les aides méthodologiques sont quasi absentes. Par ailleurs, le fait que ces méthodes se basent sur des notations semi-formelles, il est difficile (voire impossible) de raisonner rigoureusement sur les conceptions développées.

### 2.1.2 Les méthodes formelles

Dans le contexte de la conception des applications multi-agents, nous désignons par "méthodes formelles" celles utilisant des langages formels (Z, object-Z, LTL, Réseau de Petri, ...) pour décrire et concevoir ces applications.

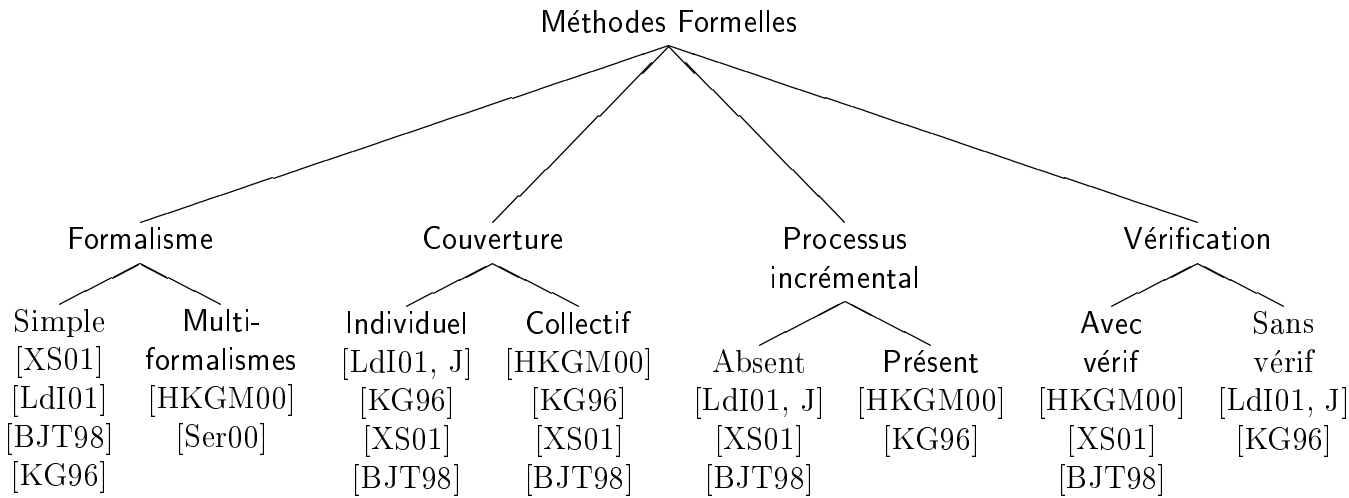


FIG. 2.3 – Aperçu sur les méthodes formelles existantes

En vue de couvrir les aspects collectif et individuel des SMA, certains travaux définissent un *modèle conceptuel* et accordent plus d'importance à l'aspect collectif. Dans ce contexte, nous pouvons citer les travaux concernant OperA [Dig03] qui proposent une sémantique formelle pour définir tous les concepts. La méthode Voyelles [RD00] s'intéresse à la décomposition de la vue système en quatre dimensions : Agent, Environnement, Interaction et Organisation. Le concepteur peut adopter différents formalismes, notations ou langages pour spécifier chaque dimension.

Peu d'autres méthodes se focalisent sur l'aspect individuel telle que la méthode de Luck et d'Inverno [LdI01]. Cette dernière consiste à définir une hiérarchie de quatre niveaux pour les entités qui composent un système à base d'agent. Cette hiérarchie commence par la spécification d'une *Entité* jusqu'à celle d'un *Agent autonome* passant par une suite de raffinements formels. En effet, ces raffinements consistent à manipuler des propriétés statiques et à omettre les propriétés comportementales. Ces dernières sont bien définies dans DESIRE [BJT98] grâce à l'utilisation de la logique temporelle. DESIRE est un framework de spécification et de conception qui supporte les agents en se basant sur la composition récursive des tâches interconnectées.

Parmi les travaux déjà cités dans le contexte des méthodes formelles, certains procèdent à valider leurs modèles conceptuels par la définition des modèles opérationnels adéquats. Dans ce contexte, nous citons le modèle organisationnel RIO

de Hilaire [HKGM00] dont la sémantique est présentée en terme d'un modèle formel. Ce modèle permet une description simple des deux aspects individuel et collectif. Le framework utilisé pour décrire ce modèle est basé sur un multi-formalisme intégrant Object Z et Statecharts. Malgré que ces travaux permettent une spécification et un prototypage des SMA, ils ne présentent aucune aide méthodologique guidant le processus de conception.

Dans ce qui suit, nous détaillons certaines de ces méthodes. Nous avons cherché à diversifier aussi bien les formalismes adoptés que les processus de conception suivis. Les méthodes que nous allons présenter sont celles de Hilaire et al. [HKGM00], de Brazier et al. [BJT98], de Luck et d'Inverno [LdI01], de Kinny et al. [KG96] et de Xu et al. [XS01]. Le choix de telles méthodes est guidé par la particularité de chacune que ce soit au niveau formalisme adopté ou processus suivi.

## **La méthode RIO**

Hilaire et al. [HKGM00] ont développé une approche multi-formalisme développée à partir des résultats de la composition de l'object-Z et Statechart. Ces deux langages ont des constructions qui permettent le raffinement des spécifications.

Afin de spécifier les protocoles d'interaction dans une organisation, cette méthode propose de raffiner la classe d'organisation en présentant des attributs de la logique temporelle. Les spécifications présentées dans cette approche peuvent être implémentées avec la plate-forme de développement multi-agent MadKit [Fer00] puisqu'elle est basée sur un modèle d'organisation. Cette méthode traite le système en tant qu'un ensemble de rôles.

Dans cette méthode, la spécification formelle des SMA est fondée sur les phases d'analyse et de conception et déclinée en trois étapes. La première considère le système en tant qu'une organisation ou société définie par un ensemble de rôles et d'interactions. La seconde présente les agents et leur assigne des rôles selon des critères de conception. La troisième se concentre sur la conception de l'architecture interne des agents. Cette approche de spécification emploie un modèle d'organisation basé sur trois concepts en corrélation : RIO (Rôle, Interaction et Organisation). Les

rôles sont des comportements génériques qui peuvent agir l'un sur l'autre, mutuellement, selon le modèle d'interaction.

Ce modèle présente de nombreux avantages. Chaque rôle constitue un grain plus fin que l'agent en terme de décomposition. Donc, on peut parler d'une large réutilisabilité. En effet, un même rôle peut être mis en œuvre par plusieurs agents.

Toutefois, il est à noter que cette méthode présente quelques limites vu qu'elle n'aborde pas certains problèmes soulevés par le développement de SMA tels que la négociation, la coopération, etc. Aussi, cette méthode ne présente pas de démarche claire, notamment, le passage d'une phase à une autre.

### **La méthode DESIRE**

Brazier et al. dans DESIRE [BJT98] (framework for DEsign and Specification of Interacting REasoning components) ont adopté la logique temporelle comme formalisme de base pour décrire les spécifications.

La méthode proposée est issue de l'ingénierie des connaissances basée sur un modèle traitant la connaissance, l'interaction, la coordination des tâches et les possibilités de raisonnement dans les SMA. Elle permet la composition, la décomposition, l'ordonnancement et la délégation des tâches, l'échange de l'information, et manipule les structures de la connaissance.

DESIRE propose deux modèles qui devraient être spécifiés par le concepteur de système à savoir le modèle *intra-agent* qui contient les descriptions des tâches, des connaissances et les possibilités de raisonnement et le modèle *inter-agent* décrivant la coordination, la coopération et d'autres types d'interaction entre les agents.

Cette méthode est basée sur les trois concepts qui sont (1) la représentation des connaissances sous forme d'une représentation graphique ou textuelle ; (2) le modèle d'agent où un agent est considéré comme un composant constitué de tâches qui peuvent être génériques ou spécifiques à l'agent ; et (3) la composition des tâches où les tâches sont décomposées en sous tâches et sont présentées sous formes de composants spécifiés par des informations d'entrée et de sortie.

Le processus de développement de DESIRE est très limité, il se concentre dans la phase de conception en présentant les différentes étapes sans aucune information sur le passage d'une phase à une autre. Cette méthode ne traite pas la notion de réutilisation des composants. DESIRE est parmi les méthodes les plus utilisées dans l'industrie, son principal avantage est l'utilisation des approches formelles intégrant un processus de vérification.

### La méthode de Luck et d'Inverno

Dans la méthode proposée par Luck et d'Inverno [LdI01], la spécification formelle des différentes entités est faite en se servant de la notation Z. La spécification des SMA repose sur la création d'un cadre général qui sert de point de départ pour la spécification. L'emploi de Z permet la spécification à différents niveaux d'abstraction. Le cadre général ou squelette de spécification propose, donc, un ensemble de concepts à raffiner pour obtenir une spécification d'un système particulier. Ces concepts définissent, d'une manière incrémentale, les différents aspects d'un SMA sous forme d'une structure hiérarchique (présentée en Z par l'inclusion de schéma). Cette structure hiérarchique se base sur l'héritage des propriétés des composants de bas niveau. Cet aspect est, facilement, modélisé par les langages orientés objets en utilisant la notion d'héritage.

La hiérarchie d'inclusion de ces concepts est spécifiée par un ensemble de schémas Z. Au sommet de la hiérarchie, se trouve l'*Environnement*. L'environnement est l'univers de toutes les entités. Il représente les attributs ou les caractères potentiellement perceptibles par toute entité (du SMA) capable de les percevoir. La deuxième entité de la hiérarchie est l'*Objet*. Il est défini par ses attributs et ses capacités. Les attributs de l'objet fournissent une description de son identité. C'est un sous-ensemble des attributs de l'environnement. Les capacités de l'objet sont les actions qu'il peut effectuer et qui ont pour effet la modification de l'environnement. Après l'*Objet*, on trouve l'*Agent* pour lequel Luck et d'Inverno ont adopté la vision de Shoham [SY93] : l'agent est toute entité à la quelle on peut associer un état mental. Les agents de Luck et d'Inverno possèdent des tâches à réaliser. La tâche est, par définition, une obligation que l'on doit remplir. C'est l'adoption d'un but (une tâche à réaliser) qui

transforme un objet en agent. Un but est vu comme une description d'un état de l'environnement qu'il faut atteindre. L'*Agent Autonome* constitue le dernier niveau de l'hierarchie d'agents. Il possède ses propres motivations. Dans ce contexte, les motivations désignent tout désir ou préférence capable de générer un but. Elles affectent, par la suite, tout raisonnement, comportement et toute perception et les orientent vers la réalisation de ce but.

Après avoir spécifié les agents autonomes, les auteurs enrichissent, sous forme de schémas Z, les perceptions, les actions ainsi que l'état actuel et futur de ces agents.

L'approche proposée par Luck et d'Inverno est de spécifier, par raffinements successifs, à partir du squelette proposé dans [dL97]. Cette approche est, donc, incrémentale et permet la définition pas à pas des aspects d'un SMA. De plus, la spécification d'un SMA est, naturellement, structurée ce qui la rend plus lisible.

L'approche de Luck et d'Inverno possède quelques inconvénients : l'accent est mis sur le comportement d'un seul agent et néglige le comportement collectif d'un SMA. Cette approche ne s'intéresse pas aux différents types d'interaction et au comportement intelligent des agents. Elle ne prend en considération que les agents autonomes. Encore plus, cette approche ne présente pas une démarche. Finalement, nous notons l'absence de la vérification qui permet de garantir la satisfaction des propriétés du système lors des étapes de raffinement.

## **La méthode de Kinny**

Kinny et al. [KG96] présentent une méthode qui adopte, d'une part, certaines terminologies et notations à partir de l'analyse et la conception orientées-objet (précisément, FUSION [PH97]) et d'autre part, un ensemble de concepts spécifiques à l'agent permettant au concepteur de comprendre et de modéliser un système complexe. Les différents concepts introduits sont divisés en deux catégories : concepts abstraits et concrets. Les concepts abstraits sont ceux utilisés durant la phase d'analyse pour concevoir le système, mais qui n'ont, nécessairement, aucune réalisation directe dans le système. Les concepts concrets, au contraire, sont utilisés dans le processus de conception et auront une directe implémentation dans le système à

exécuter.

Le principe de cette méthode est de transformer les modèles décrits dans la phase d'analyse en un niveau d'abstraction suffisamment bas de telle sorte que les techniques de conception traditionnelles peuvent être appliquées. Le processus d'analyse et de conception orientées agent est concerné par : (1) comment une société d'agents coopère pour réaliser les objectifs au niveau système? et (2) quelles sont les tâches à attribuer à chaque agent pour les atteindre.

Cette méthode propose une analyse et une conception orientées agent. Elle est générale vue qu'elle est applicable pour une large gamme de SMA. En outre, elle est compréhensive traitant les aspects du système relatifs aux deux niveaux : le niveau macroscopique (social) et le niveau microscopique (agent). Cette méthode est fondée sur une vue du système comme une organisation computationnelle se composant de plusieurs rôles interagissants.

### **La méthode de Xu**

Xu et al. [XS01] proposent une approche qui consiste à utiliser un modèle formel dans la conception des agents. Cette approche est basée sur G-net, qui est un type du réseau de Petri défini pour supporter la modélisation du système en terme d'un ensemble de modules indépendants. Le modèle de base G-net est ajusté pour définir le "G-net orienté agent" qui peut servir comme un modèle générique de la conception d'agent.

Le principe de cette approche consiste à instancier un G-net selon deux étapes : générer un identifiant unique de l'agent, et initialiser l'état mental de l'agent résultant. En plus, cinq modules spéciaux sont introduits pour décrire un agent autonome. Ces modules sont : (1) le module *Buts*, (2) le module *Plan*, (3) le module à *Base de connaissances*, (4) le module *Environnement* et (5) le module *Planificateur*. Les modules *But*, *Plan* et à *Base de connaissances* sont basés sur le modèle agent BDI [RG98], tandis que le module *Environnement* représente un modèle abstrait de l'environnement. Le module *Planificateur* représente la partie la plus essentielle d'un agent qui lui permet de décider d'ignorer un message reçu, de débiter une nouvelle

conversation, ou de continuer avec la conversation courante. Dans le module *Planificateur*, les plans sont achevés et les modules de *But*, *Plan* et à *Base de connaissance* d'un agent sont mis à jour après chaque acte communicative ou si l'environnement change. Pour supporter la conception orientée agent, Xu et al. ont senti le besoin d'adopter le mécanisme de l'*héritage*.

## Synthèse

Les méthodes formelles sont sans doute nécessaires à la modélisation des systèmes complexes comme les SMA. Ces méthodes permettent d'avoir une expression claire des propriétés du SMA à modéliser. Étant donné que les méthodes formelles de conception des applications à base d'agents adoptent de telles méthodes, elles permettent de décrire les différents aspects des SMA en se basant sur des formalismes ayant des sémantiques claires et précises aussi bien que sur des processus bien définis.

Dans les méthodes formelles présentées, nous notons que la plupart de ces méthodes n'adoptent pas un processus de développement systématique et incrémental des applications multi-agents. En effet, elles souffrent d'un manque d'aides pratiques et des issues méthodologiques dans la phase de conception. En outre, malgré qu'elles se sont basées sur des méthodes formelles, certaines ne présentent pas des processus de vérification.

Vu que nous nous intéressons particulièrement aux techniques formelles, et afin de mieux appréhender ces méthodes, nous proposons de mener dans la section suivante une étude comparative de ces méthodes.

## 2.2 Étude comparative des principales méthodes formelles

La diversité des aspects révélant des SMA et la complexité inhérente du développement des applications multi-agents nous incitent à noter l'importance de

certains critères devant être présents dans les méthodes de conception de telles applications. Ces critères se rapportent sur la couverture des différents aspects caractérisant les SMA, l'utilisation d'un formalisme adéquat capable de décrire ces aspects et l'adoption d'un processus incrémental se basant sur des étapes guidées et accompagnées par un processus de vérification bien défini.

Ainsi, nous proposons, dans ce qui suit, de présenter une analyse comparative des cinq principales méthodes existantes en se basant sur cinq critères de comparaison à savoir la *couverture des aspects multi-agents*, la *langage de spécification*, le *processus de conception*, le *processus de vérification* et les *aides méthodologiques*.

Dans cette étude, nous proposons de comparer les principales méthodes formelles existantes. Pour ce faire, nous considérons les méthodes brièvement présentées dans la section suivante à savoir les méthodes de Luck et d'Inverno [LdI01], de Kinny et al. [KG96] et de Brazier et al. [BJT98] pour la première catégorie de méthodes adoptant un formalisme simple ; la méthode de Xu et al. [XS01] pour la deuxième catégorie qui ont proposé une extension d'un formalisme existant ; et la méthode de Hilaire et al. [HKGM00] pour la troisième catégorie qui se base sur une combinaison de formalismes existant.

### **2.2.1 Le critère 1 : la couverture des aspects individuels et collectifs**

En étudiant les méthodes sujet de cette comparaison, nous notons que certaines se focalisent sur les concepts relatifs à l'organisation des SMA. A titre d'exemple, la méthode de Hilaire et al. [HKGM00] considère le système en tant qu'une organisation définie par un ensemble de rôles et d'interactions. Les rôles sont des comportements génériques qui peuvent interagir selon un modèle d'interaction. En outre, la méthode de Kinny [KG96] traite les aspects relatifs au niveau macroscopique (social). En plus, cette méthode, et plus précisément, le processus d'analyse et de conception orientées-agent s'intéresse à l'aspect coopération qui permet aux agents de réaliser les objectifs communs et aux différentes compétences qui doivent être présentes chez ces agents pour mener à bien les dits objectifs.

Plusieurs aspects liés à la coopération, tels que la négociation et la délégation de tâches, ont été bien traités dans d'autres méthodes formelles. En effet, la méthode de Brazier et al. [HKGM00] est basée sur un modèle inter-agent traitant la connaissance, l'interaction, la coordination des tâches et les possibilités de raisonnement dans les SMA.

Contrairement aux aspects collectifs, les aspects individuels nécessitent un accord sur des définitions universelles des concepts clés spécifiques à l'agent, tels que *compétence*, *but*, *intention*, etc. Dans ce contexte, Brazier et al. [HKGM00] proposent le modèle intra-agent qui contient les descriptions des tâches, des connaissances et les possibilités de raisonnement. En outre, la méthode de Luck et d'Inverno [LdI01] met l'accent sur le comportement d'un agent et néglige le comportement collectif d'un SMA. Aussi, la méthode de Xu [XS01] a mis l'accent sur l'aspect individuel. Elle décrit un agent par un état mental et des mécanismes de raisonnement. Par contre, la méthode de Kinny [KG96], aborde les deux niveaux macroscopique et microscopique. Ce dernier consiste à définir, selon un processus d'analyse et de conception, ce qui est nécessaire à chaque agent pour pouvoir coopérer avec les autres.

### 2.2.2 Le critère 2 : le langage de spécification

Dans la littérature, les méthodes formelles présentent trois catégories : certaines ont adopté des langages de spécification simples (un seul formalisme) tel que Z. D'autres ont proposé des extensions de formalismes existants en vue d'arriver à décrire les aspects des SMA telles que les extensions apportées aux réseaux de Petri [XS01]. Finalement, la troisième catégorie consiste plutôt à combiner deux formalismes ou plus en vue de définir un langage de spécification composé de plus qu'un formalisme (multi-formalisme) tel que le multi-formalisme de Hilaire et al. [HKGM00].

La méthode de Luck et al. [LdI01] appartient à la première catégorie utilisant un formalisme simple à savoir la notation Z.

Aussi, la méthode de Brazier et al. (DESIRE) [HKGM00] fait partie de la première catégorie. La philosophie de DESIRE se contredit avec celles des langages

de spécification généralement proposés tels que Z ou VDM, précisément dans le fait qu'elle ne suggère pas un type spécifique d'architecture de spécification et de conception de SMA. DESIRE permet plus de structures et, ainsi, plus de supports. L'un des buts du framework DESIRE est de permettre des constructeurs avec lesquels les patrons de raisonnement peuvent être, explicitement, modélisés. DESIRE a l'avantage additionnel que les spécifications et leurs sémantiques peuvent être décrites, formellement, utilisant la logique temporelle comme étant une base. Ceci permet de prouver diverses propriétés relatives au système.

La méthode de Xu et al. [XS01] fait partie de la deuxième catégorie (extension d'un formalisme existant). Cette approche est basée sur les G-nets, qui sont une extension des réseaux de Petri, définie pour supporter la modélisation du système en terme d'un ensemble de modules indépendants. Le "G-net orienté-agent" peut servir comme un modèle générique de la conception d'agent. Il présente un ensemble de modules sous forme de places et reliés entre eux à travers des transitions. Dans un niveau plus détaillé de conception, ces transitions sont raffinées en sous-réseaux.

Enfin, la méthode de Hilaire (RIO) [HKGM00] appartient à la troisième catégorie (multi-formalismes). Une approche de multi-formalismes est développée en se basant sur la composition de l'object-Z et le statechart. L'object-Z est une extension de la notation Z intégrant la notion d'objet. Il est utilisé afin de spécifier les évolutions des entités alors que statechart est une extension des automates à états finis avec des constructions pour définir le parallélisme et la communication (les aspects réactifs des SMA). L'approche multi-formalismes adoptée pour la spécification de SMA consiste en l'utilisation de deux langages pour spécifier un système. Le principe consiste à ajuster un langage formel en vu de pouvoir lui intégrer d'autres formalismes permettant, ainsi, d'augmenter son pouvoir expressif. L'approche de composition est basée sur deux étapes : (1) la première étape consiste à intégrer les notations (intégration syntaxique) qui consiste à encapsuler un comportement dans une classe Object-Z. Cette encapsulation se fait par un schéma particulier qui inclut un statechart. (2) La deuxième étape doit fournir une sémantique formelle à l'ensemble des spécifications partielles écrites en utilisant les deux formalismes. Cette étape consiste à définir des règles de traduction vers un domaine sémantique

commun à savoir les systèmes de transitions, ce qui confère aux spécifications une sémantique opérationnelle.

### 2.2.3 Le critère 3 : le processus de conception

Hilaire et al. [HKGM00] présentent plusieurs mécanismes qui permettent de décomposer et hiérarchiser des schémas d'interactions décrits par des rôles et des organisations. Le premier mécanisme consiste à la généralisation/spécialisation des éléments caractérisant un rôle à la manière de l'héritage pour les langages orientés objet. Le second mécanisme assimile la notion de rôle à celle d'organisation d'un grain plus fin. La spécification d'un agent peut se résumer, grossièrement, à la construction d'un agent à partir des rôles qu'il met en œuvre et, éventuellement, par certains autres mécanismes (comme la coordination de ces rôles, par exemple).

Pour faciliter la mise en pratique du squelette de spécification, Hilaire et al. proposent une méthode. Selon cette méthode, le processus de développement d'un SMA peut être décomposé en trois grandes phases : *analyse*, *conception* et *réalisation*. En cela, il est proche du processus classique adopté en génie logiciel. La première phase d'*analyse* consiste à la description informelle du problème. On doit, pour un problème donné, identifier : la nature des entités, leurs différents liens, les interactions occurring entre ces entités, et conduire une analyse fonctionnelle. La phase de *conception* est entièrement basée sur le méta-modèle rôle/organisation. À partir de la phase d'analyse, il faut identifier les rôles, les interactions et les organisations spécifiques au problème. La dernière phase (la *réalisation*), consiste essentiellement à préciser l'architecture des agents, les choix d'implémentation, les protocoles d'interaction, les modes de communication, ... Concernant ce dernier point, Hilaire et al. pensent établir un certain nombre de règles de transformation permettant le passage de la spécification à une implémentation sous MadKit [Fer00]. Ainsi, la spécification du méta-modèle est conçue comme un squelette qu'on peut raffiner et spécialiser de manière à obtenir la spécification d'un SMA.

Kinny et al. [KG96] proposent, dans leur méthode, de traiter deux phases : la phase d'*analyse* et la phase de *conception*. La phase d'*analyse* peut être résumée dans les étapes suivantes : (1) identifier les rôles dans le système, (2) pour chaque

rôle, identifier et documenter les protocoles associés, (3) élaborer le modèle des rôles en utilisant le modèle de protocole, et (4) itérer les étapes (1)–(3).

Le processus de *conception* génère trois modèles : le modèle agent, le modèle services et le modèle d'acointances.

La phase de *conception* de la méthode se résume par : (1) créer un modèle agent : cette étape consiste à agréger les rôles en types d'agent et les raffiner pour former une hiérarchie de type d'agent ; (2) développer un modèle services en examinant les protocoles et les propriétés de sûreté et de vivacité de chaque rôle ; et (3) développer un modèle d'acointances à partir du modèle d'interaction et du modèle d'agent.

Contrairement à Hilaire et al. qui proposent un processus de développement assez complet traitant les différentes phases, le processus de développement proposé par Brazier et al. dans DESIRE est très limité. Ce processus se concentre sur la phase de conception en présentant les différentes étapes sans aucune information sur le passage d'une phase à une autre. Cette méthode ne traite pas la notion de réutilisation des composants.

Le processus de développement de DESIRE est présenté par (1) la décomposition des tâches qui consiste à définir la hiérarchie de tâche et la description des tâches individuelles (entrées, sorties, relations etc.), (2) la définition d'un modèle d'échange de l'information, (3) l'ordonnancement des sous-tâches (l'ordre temporel des tâches, l'effort pour résoudre la tâche, etc.), (4) la délégation des sous-tâches en assignant les tâches aux agents, et (5) la définition de la structure de la connaissance en fournissant les concepts qui décrivent les objets et les relations qui présentent comment les concepts se relient entre eux.

Xu et al. proposent un processus de raffinement assurant le passage à un niveau de conception plus détaillé : les transitions abstraites seront raffinées en sous-réseaux. Pour supporter la conception orientée agent, Xu et al. proposent d'incorporer certaines capacités de modélisation héritée. Mais, l'héritage dans la conception orientée agent est plus compliqué que dans la conception orientée objet vu que, contrairement à un objet (passif), un agent a un état mental et des mécanismes de raisonnement. Ainsi, ces auteurs pensent que tant que l'héritage se présente au niveau classe, une sous classe agent peut être initialisée avec l'état mental d'une super classe agent,

mais la nouvelle connaissance acquise, nouveaux plans construits, et les nouveaux buts générés dans un agent individuel (comme une instance de la super classe agent), ne peuvent pas être hérités par un agent lors de la création d'une instance de la sous classe. Pour simplifier, ils assument qu'une instance d'une sous classe agent utilise, toujours, ses propres mécanismes de raisonnement, et, ainsi, les mécanismes de raisonnement dans la super classe seront désactivés.

## 2.2.4 Le critère 4 : la vérification

Les preuves formelles servent à montrer qu'une spécification est consistante et qu'une spécification raffine une autre permettant, ainsi, d'ajouter de la qualité au développement de logiciel et de garantir la fiabilité de l'application développée.

La construction de preuves sur les propriétés des spécifications, sujet de raisonnement, permet de détecter les problèmes le plutôt possible durant le processus de développement. Aussi, les preuves construites peuvent aider à saisir les besoins du système et assister à identifier les différentes suppositions.

Dans la phase de conception, une preuve peut nous montrer non seulement que la conception est correcte, mais aussi pourquoi elle est correcte.

Les domaines d'application des SMA sont, souvent, des domaines critiques nécessitant un processus de spécification clair et une conception correcte. De ce fait, nous notons la nécessité d'appliquer les méthodes formelles suivies par les preuves appropriées.

Il est à noter que la majorité des méthodes de développement des applications à base d'agents souffre de l'absence d'un processus de vérification. En effet, la vérification est complètement omise dans la méthode de Kinny [KG96]. Par contre, Hilaire et al. proposent une approche de vérification basée sur la traduction de spécifications en systèmes de transitions et l'utilisation du logiciel STeP [MP92] qui permet d'automatiser les preuves. Le prototypage et la vérification bénéficient alors d'une approche incrémentale qui est plus efficace que le prototypage ou la vérification de l'intégralité d'un système dont l'espace d'états peut être infini. Ainsi, les actions collectives et les comportements des agents sont spécifiés. À partir d'une

telle spécification, et avant de la raffiner pour spécifier les agents, cette approche consiste à mettre en œuvre des processus de prototypage et de vérification. Cette approche de prototypage est basée sur l'exécution des statecharts avec l'environnement STATEMATE. Les comportements des rôles et des organisations peuvent être simulés.

Pour DESIRE (Brazier et al.), une méthode de vérification compositionnelle est décrite et appliquée en utilisant le système de raisonnement de la logique temporelle. Une méthode compositionnelle de vérification de SMA prend en considération les niveaux d'abstraction du processus et la structure compositionnelle en relation. Le processus de vérification est réalisé par des preuves mathématiques assurant que la spécification du système intègre les propriétés souhaitées. Les besoins sont formulés, formellement, en terme de sémantiques temporelles. Durant le processus de vérification, les besoins du système, comme un tout, sont dérivés à partir des propriétés des agents. Ces propriétés sont à leur tour dérivées à partir des propriétés des composants des agents. Les composantes primitives qui ne sont pas composées d'autres peuvent être vérifiées en utilisant des méthodes de vérification plus traditionnelles pour les systèmes à base de connaissances ou d'autres méthodes de vérification accordant au type de vérification utilisé.

Xu et al., à leur tour, voient que l'un des avantages de la construction des modèles formels dans la conception orientée agent est d'aider à s'assurer que la conception soit correcte. Une conception correcte d'agent doit garantir certains besoins clés tels que la vivacité (sous certaines conditions un évènement finira par avoir lieu), le non blocage (le système ne se trouvera jamais dans une situation où il ne peut plus progresser) et la concurrence (les participants sont en concurrence dans les évènements ou dans le partage des ressources). Aussi, certaines propriétés, telles que l'héritage, ont besoin d'être vérifiées pour assurer leurs fonctionnalités correctes. Les réseaux de Petri offrent une technique prometteuse pour vérifier que la conception est correcte. Dans cette méthodes, Xu et al. utilisent un outil, appelé INA (Integrated Net Analyzer) [SR98], pour analyser et vérifier leurs modèles d'agent. Afin de prouver des propriétés comportementales additionnelles du modèle révisé, ils utilisent certaines capacités de model checking permises par l'outil INA.

### 2.2.5 Le critère 5 : les aides méthodologiques

Les méthodes de développement orientées agents, telles que celles présentées dans ce chapitre, essaient principalement de proposer des approches claires pour analyser, concevoir et développer les SMA utilisant des méthodes et des techniques spécifiques. Bien que ces approches formelles proposent une description formelle et rigoureuse des applications à base d'agents, elles souffrent de l'absence d'aides méthodologiques.

Afin d'être applicable avec succès, chaque phase d'un processus doit être enrichie par certains guides (ou aides) méthodologiques. Par "guides ou aides méthodologiques", nous désignons toute assistance pratique durant le processus de conception. Cette assistance consiste à aider et guider le concepteur à suivre l'approche selon un ensemble d'étapes et de règles de passage d'une étape à la suivante.

Dans la phase de conception des SMA, la définition d'une méthode convenable nécessite la proposition d'un ensemble d'aides méthodologiques spécifiques à cette phase. Un tel ensemble doit définir comment les différentes étapes de conception doivent être organisées et enchaînées les unes aux autres, quelles activités de l'ingénierie et de développement à accomplir dans chaque étape et quand les techniques et les artifices sont à appliquer pour ces activités. L'absence de ces aides signifie l'absence de contrôle pour le processus et ainsi ce dernier peut devenir un effort chaotique avec une faible probabilité d'atteindre les résultats voulus.

## 2.3 Synthèse

A partir de l'étude comparative faite selon les cinq critères de comparaison déjà fixés (figure 2.4), nous avons noté l'absence d'une méthode formelle qui répond à la totalité de ces critères. En effet, d'une part, il n'existe aucune méthode formelle qui garantit la couverture de tous les aspects des SMA (individuel et collectif, statique et comportemental) et l'utilisation d'un langage de spécification assez complet capable de décrire ces différents aspects et de vérifier les propriétés jugées importantes. D'autre part, nous notons l'absence d'un processus de conception à base de

Methodologies	Couverture					L'engagement de spécification (formalisme) complet	Vérification	Processus clair	Aides méthodologiques
	Individuel	Collectif							
		Organisation	Coopération	Négociation	Interaction				
Luck et d'Inverno	Oui	Non	Non	Non	Non	Non (formalisme simple)	Non	Non	Non
Hilaire et al.	Non	Oui	Non	Non	Non	Oui (multi-formalisme)	Oui (Avec transition)	Oui	Non
Kimmy et al.	Oui	Oui	Oui	Non	Non	Non (Simple)	Non	Oui	Non
Xu et al.	Oui	Non	Non	Non	Oui	Oui (Extension d'un formalisme)	Oui (Directe)	Non	Non
Brazier et al.	Oui	Non	Oui	Oui	Oui	Non (Simple)	Oui (Directe)	Non	Non
FormAAD	Oui	Oui	Oui	Oui	Oui	Oui (multi-formalisme)	Oui (Intégré)	Oui	Oui

FIG. 2.4 – Tableau récapitulatif pour l'étude comparative de cinq méthodes formelles selon cinq critères

raffinements successifs avec des étapes claires accompagnées d'une assistance pratique aidant l'utilisateur à concevoir de manière systématique et incrémentale les différents aspects de l'application à spécifier.

En se basant sur les synthèses faites à propos des différents travaux existants, notre travail consiste à proposer, dans un premier lieu, un langage de spécification formel orienté agent qui permet de couvrir les différents aspects (statique et comportemental) des agents tels que la connaissance, le but et le rôle, aussi bien que les aspects collectifs d'une application multi-agent, tels que les protocoles d'interaction et la structure d'organisation. Ce langage intègre la logique temporelle linéaire dans la notation  $Z$ . En effet, le langage  $Z$  possède toutes les caractéristiques nécessaires pour manipuler des aspects structuraux et fonctionnels des agents (l'état mental et les traitements associés), tandis que la logique temporelle est considérée en tant qu'un des formalismes les plus éminents pour décrire des comportements réactifs des agents [MP92]. Afin de fournir une interprétation formelle pour les opérateurs temporels définis, nous développons une sémantique opérationnelle pour des applications multi-agents en termes de séquences des états du système. La définition de ce modèle temporel dans la notation  $Z$  permet d'employer des outils soutenant la notation  $Z$ , tel que  $Z$ -EVES [MS99] que nous adoptons pour la vérification. Ces outils nous ont permis d'effectuer la vérification de la syntaxe, du type et du domaine de nos spécifications et de raisonner au sujet de l'exactitude en prouvant les propriétés désirées.

En deuxième lieu, nous proposons une méthode de conception d'applications multi-agents, basée sur les raffinements successifs [San00] où nous essayons de satisfaire les différents critères sujet de la comparaison (la figure 2.4). Nous définissons un ensemble d'étapes de raffinement permettant de développer, étape par étape, les comportements des différents agents à partir des spécifications abstraites de l'objectif commun. Nous fournissons les directives méthodologiques qui aident (1) à définir une stratégie de coopération pour réaliser l'objectif commun donné, (2) à décrire une structure d'organisation, puis (3) à identifier les actions nécessaires de communication et en conclusion, (4) à dériver les comportements appropriés des agents.

Afin de garantir l'exactitude des spécifications de conception, nous formulerons,

pour chaque étape de raffinement, une obligation de preuve. Celle ci permet de s'assurer que les spécifications raffinées préservent les propriétés de l'application à développer.



# Chapitre 3

## $\mathcal{T}_{temporal}\mathcal{Z}$ : un langage de spécification multi-formalisme

Selon les constatations dégagées de l'étude comparative des méthodes existantes, présentée dans le chapitre précédent, nous avons noté la nécessité d'avoir un formalisme suffisamment expressif pour décrire les différents aspects des SMA. Ainsi, nous proposons, dans ce chapitre, un langage de spécification des applications à base d'agents. Ce langage doit être capable de spécifier aussi bien l'aspect individuel, l'aspect collectif que les propriétés structurelles et comportementales d'un SMA.

Généralement, un SMA est considéré comme une collection de composants qui évoluent, continuellement, dans un environnement contenant des agents actifs et des objets passifs [SLL02]. En conséquence, les spécifications d'une application à base d'agents incluent des descriptions de l'environnement, de la structure et des comportements des différents agents (intra-agent) et les primitives de communication aussi bien que les protocoles d'interaction (inter-agent). En outre, nous ajoutons à la partie collective une description des structures d'organisation et des activités de planification.

Afin de couvrir tous ces aspects, nous avons opté pour l'intégration des formules temporelles dans des schémas Z. Cette intégration nous permet de couvrir des aspects inter et intra-agent dans un cadre unifié. En fait, la partie Z vise à décrire les composants (passifs et actifs) en termes d'attributs et de propriétés afférentes. D'autre part, la partie logique temporelle vise à enrichir cette description par les

propriétés liées aux comportements et à l'interaction.

Ainsi, ce chapitre débute par justifier le choix des formalismes adéquats pour parvenir à présenter un langage de spécification des applications à base d'agents qui répond aux critères envisagés. Ensuite, nous passons à introduire les formalismes retenus. En premier lieu, nous commençons par présenter quelques éléments essentiels de leur syntaxe ; puis, nous donnons une définition de la sémantique associée à une spécification écrite avec l'un et l'autre des formalismes.

### 3.1 Choix des formalismes de base

L'étude comparative portant sur les critères *langage de spécification* et *vérification* de la section 2.2 du chapitre précédent nous permet de mieux comprendre quels éléments devront être pris en considération pour parvenir à un formalisme permettant de mieux exprimer les divers éléments qui se présentent dans la conception des SMA. Les formalismes, adoptés dans les travaux existants, manquent de pouvoir d'expression. Certains critères comme l'aspect comportemental ne sont pas pris en compte par la plupart des formalismes actuels. Les problèmes des contraintes de temps et de coordination d'actions ne sont pas, souvent, suffisamment traités.

Étant données ces constatations, nous allons, dans ce qui suit, choisir les formalismes adéquats et expressifs pour arriver à décrire les différents aspects des SMA.

Les questions qui se posent sont : Quelle est la stratégie la plus simple : créer un nouveau formalisme de toutes pièces ; utiliser deux formalismes préexistants d'une manière séparée pour compléter la couverture et décrire l'aspect statique et l'aspect comportemental ; manipuler un formalisme déjà existant et améliorer sa couverture ; utiliser un formalisme hybride, issu de la fusion (intégration) de deux formalismes plutôt que de les utiliser conjointement [Bru98] ?

Créer un nouveau formalisme de toutes pièces est une stratégie longue et coûteuse si on veut vraiment obtenir un taux de couverture assez large du problème et un formalisme cohérent. On doit, d'abord, définir la syntaxe du formalisme ; puis définir sa sémantique et éventuellement une sémantique opérationnelle. On n'est pas obligé

de définir de sémantique, mais, alors, on se place dans le cadre de notations non formelles ou semi-formelles ce qui nous ramène aux problèmes que nous avons évoqués dans les sections précédentes sur les méthodes semi-formelles. Or certains problèmes nécessitent des modèles formels solides pour être traités correctement.

Utiliser deux formalismes complémentaires afin de mieux couvrir les différents aspects du problème est une autre possibilité. La solution judicieuse est alors d'utiliser un formalisme pour la description de l'aspect statique et un autre pour la description de l'aspect comportemental. Cette stratégie semble séduisante, dans un premier temps, puisqu'elle permet d'utiliser des formalismes déjà existants. Le problème majeur est la définition d'une sémantique unifiée ainsi que l'existence d'outils couvrant les deux formalismes à la fois. Ainsi, il devient nécessaire d'utiliser deux démarches séparées ce qui engendre des difficultés de preuve de cohérence de la conception.

Enrichir un formalisme existant est la stratégie qui paraît la plus raisonnable. Deux cas sont, alors, envisageables : on peut partir d'une notation ne possédant pas de sémantique ou au contraire un formalisme à la sémantique forte. Dans le premier cas, il est facile de greffer à de tels formalismes de nouvelles capacités de description. De tels formalismes sont généralement ambigus et il sera plus difficile de leur attribuer une sémantique par la suite. Néanmoins, il reste possible, mais difficile, de leur attribuer une sémantique opérationnelle [GJ95]. Dans le cas d'un formalisme à la sémantique forte, toute extension nécessite un long effort pour décrire la nouvelle sémantique. Mais, on reste dans un cadre formel, ce qui permet d'espérer pouvoir obtenir une sémantique opérationnelle, de la génération automatique de toute ou partie de l'interface (prototypage), ainsi que de la vérification de certaines propriétés, éventuellement de manière automatique. Dans ce cas, il faut aussi choisir un formalisme de départ "réaliste", avec un taux acceptable de couverture du problème. Dans le cas où sa sémantique est simple faisant appel à un minimum de connaissances mathématiques, ça ne pourra que faciliter son intégration dans le monde réel.

On peut, aussi, considérer des stratégies hybrides comme la fusion de deux formalismes plutôt que leur utilisation conjointe. Par exemple, on peut citer un formalisme qui fusionne Z et une logique temporelle. Z est fondé sur l'algèbre ce qui lui confère

son aspect formel et il n'est pas difficile d'ajouter des extensions à  $Z$  tant qu'on le fait dans le cadre de l'algèbre : c'est un moyen simple d'améliorer un formalisme existant.

Ainsi, notre choix consiste à adopter une stratégie hybride permettant, à la fois, l'utilisation de deux formalismes et l'amélioration d'un formalisme existant. Nous allons considérer un formalisme existant simple et lisible et nous allons l'améliorer en lui intégrant un autre formalisme existant. Nous obtenons, alors, un formalisme assez expressif et capable de spécifier les aspects requis. Il s'agit d'intégrer ou de fusionner ces formalismes plutôt que de les utiliser séparément. Il reste, maintenant, de choisir les formalismes préexistants sujet de cette intégration.

Comme synthèse de l'étude comparative sur les méthodes de conception des applications à base d'agents faite dans la section 2.2 du chapitre précédent, nous avons relevé un certain nombre de critères que devra prendre en compte le choix de formalisme :

- Il est important de pouvoir spécifier l'*aspect statique* et l'*aspect comportemental* caractérisant les applications à base d'agents.
- Le formalisme doit être simple à utiliser et facile à comprendre. En effet, il ne doit pas être réservé à des spécialistes, il doit pouvoir être utilisé par un grand nombre d'utilisateurs. En outre, il faudrait augmenter sa sémantique afin de pouvoir traiter les différents aspects évoqués précédemment à savoir l'aspect statique et l'aspect comportemental. Pour ces deux raisons, sa sémantique doit être relativement aisée à manipuler et nous devons pouvoir utiliser le formalisme en "oubliant" sa sémantique.
- Le formalisme choisi doit garantir le pouvoir incrémental permettant de passer d'une spécification abstraite des besoins vers des spécifications plus raffinées et, ainsi, plus proche de l'implémentation.
- Des outils de support pour la vérification automatique des propriétés décrites avec ce formalisme doivent être disponibles.

Étant donnés ces critères, nous proposons, en premier lieu, l'utilisation du *langage Z*. Ce choix est motivé par trois raisons principales. D'abord, c'est une notation standard assez expressive pour permettre les représentations compréhensibles et unifiées

des applications à base d'agents. Ensuite, ce langage intègre une approche rigoureuse pour la structuration et la composition des spécifications. Enfin, il est soutenu par plusieurs outils, tels que Z-EVES [MS99] et Isabelle-HOL [KSW96], qui offrent une vérification syntaxique et des types aussi bien que des preuves de théorèmes. En outre, Z est fondé sur l'algèbre ce qui lui confère son aspect formel et il n'est pas difficile d'ajouter des extensions à Z tant qu'on le fait dans le cadre de l'algèbre : c'est un moyen simple d'étendre un formalisme existant.

Dans le contexte du développement des applications à base d'agents, plusieurs travaux se sont basés sur l'utilisation de Z dont nous pouvons citer les travaux de Luck et d'Inverno [LdI01]. Ces derniers ont pu décrire les aspects internes des agents à savoir l'état mental, les tâches à réaliser, l'autonomie, etc. contrairement aux aspects collectifs et comportementaux liés à l'interaction, l'organisation, ... qui ont été omis vu l'incapacité de Z à expliciter de tels aspects.

Pour combler ce manque et être capable de décrire les aspects comportementaux et les interactions qui caractérisent un agent, nous avons opté pour l'utilisation de la *logique temporelle linéaire*. Ce choix est motivé par le fait que cette logique est appropriée pour capturer les propriétés de *sûreté* et de *vivacité* liées au comportement réactif des agents. Par rapport à d'autres techniques, la logique temporelle permet une distinction claire entre certaines propriétés telles que celles de sûreté et de vivacité. En plus, elle soutient le raisonnement formel basé sur un système rigoureux de preuve. Contrairement à d'autres techniques, telles que les machines à états finis, où la taille des preuves augmente exponentiellement avec le nombre d'états, les preuves de logique temporelle demeurent compactes et claires [DB01]. Les formalismes basés sur la logique temporelle ont une sémantique relativement facile à manipuler. Les spécifications obtenues à l'aide d'une logique temporelle sont proches du langage naturel, ce qui les rend aisément compréhensibles même par les non spécialistes.

Dans la littérature, le langage Z a été combiné avec différents formalismes tels que CSP [MS01], CCS [GS97], ... Par ailleurs, Z a été combiné avec la logique temporelle tel que dans les travaux de Duke [DG89]. Ces travaux consistent à étendre Z en vu d'inclure les opérateurs de la logique temporelle. Ces opérateurs ne sont utilisés que pour raisonner sur la séquence d'états des schémas Z sans aucune intention de les

appliquer dans les formules  $Z$ . Cette combinaison a été faite seulement au niveau de la syntaxe en intégrant les opérateurs temporels dans les schémas  $Z$ . Cependant aucune intégration sémantique n'a été faite. Ce manque a empêché particulièrement l'utilisation des supports pour la vérification des spécifications.

Comme notre choix s'est porté sur l'utilisation combinée et intégrée des langages formels  $Z$  et logique temporelle aussi bien au niveau syntaxique que sémantique, nous présentons les bases nécessaires à la compréhension de spécifications écrites avec ces formalismes et nous définissons notre langage  $\mathcal{T}_{temporal}Z$  de spécification de SMA comme une intégration de  $Z$  et de la logique temporelle linéaire.

## 3.2 La notation $Z$

La notation  $Z$ , comme elle est présentée dans [Spi92], est un langage formel de spécification orienté modèle qui est basé sur la théorie des ensembles et la logique de prédicats de premier ordre. Ce langage est employé pour décrire une application en terme d'états et d'opérations sur ces états. Afin de structurer des spécifications et de les composer,  $Z$  emploie un *langage de schémas*. Ce dernier permet de rassembler des objets et de les encapsuler pour une réutilisation.

$Z$  est un langage qui a, déjà, été utilisé pour spécifier de nombreux systèmes. Wooldridge et Jennings [WJK99] soulignent l'intérêt que peut représenter, pour les SMA, l'utilisation d'un tel langage. D'une part, l'expertise sur ce langage est très importante et d'autre part, les outils développés pour ce langage sont très nombreux. De fait,  $Z$  [Spi92] a déjà été utilisé pour spécifier des SMA [BAL05, dL97, LdI01]. La spécification des SMA proposée par Luck et d'Inverno [LdI01] repose sur la création d'un cadre général qui sert de point de départ pour la spécification. L'emploi de  $Z$  permet la spécification à différents niveaux d'abstraction. Le cadre général ou squelette de spécification propose, donc, un ensemble de concepts à raffiner pour obtenir une spécification d'un système particulier. Ces concepts définissent de manière incrémentale les différents aspects d'un SMA. La hiérarchie d'inclusion de ces concepts est spécifiée par un ensemble de schémas  $Z$ .

### 3.2.1 Les langages de Z

La théorie des ensembles utilisée dans Z inclut un ensemble d'opérateurs standards et un ensemble de produits cartésiens. La logique mathématique utilisée est une logique de prédicats de premier ordre. Ensemble, ils forment un langage mathématique qui est facile à lire et à appliquer. Cependant, ce langage n'est qu'un seul aspect de Z. Un autre aspect est la manière dont les mathématiques peuvent être structurées. Les objets mathématiques et leurs propriétés peuvent être collectés ensemble dans *un schéma* : un patron de déclarations et de contraintes. Le langage schéma peut être utilisé pour décrire l'état d'un système et la manière dont cet état peut changer. Il peut être, aussi, utilisé pour décrire les propriétés du système et raisonner à propos des raffinements possibles d'une conception.

Un schéma Z se compose de "déclarations" et de "prédicats" :

<i>SchemaName</i>
<i>Declarations</i>
<i>Predicates</i>

Pour décrire une opération sur un état, nous employons deux copies de cet état : la première représente l'état avant d'appliquer l'opération (*Schema*) et la seconde représente l'état (*Schema'*) après avoir effectué l'opération :

<i>OperationName</i>
<i>Schema</i>
<i>Schema'</i>
<i>Predicate</i>

Une caractéristique importante dans Z est l'utilisation de *Types*. Chaque objet dans la logique mathématique a un type unique, est représenté comme étant l'ensemble maximal dans la spécification courante.

Le type de tout schéma peut être considéré comme le produit cartésien des types de chaque variable, sans aucune notion d'ordre, mais contraint par les prédicats de

ce schéma. La modularité est facilitée dans  $Z$  par l'autorisation de l'inclusion de schémas. On peut sélectionner une variable *var* d'un schéma *schema* en écrivant *schema.var*. Pour introduire un type dans  $Z$ , abstraction faite de son contenu et des éléments de ce type, on utilise la notion de *givenset*. On écrit  $[NODE]$  pour représenter l'ensemble de tous les nœuds. Si on veut indiquer qu'une variable englobe certains ensembles de valeurs ou une paire ordonnée de valeurs, on écrit, respectivement,  $x : \mathbb{P} \text{ NODE}$  et  $x : \text{NODE} \times \text{NODE}$ .

Le type *Relation* exprime des relations entre deux types existants connus par les types source et cible. Le type d'une relation ayant la source  $X$  et la cible  $Y$  est  $\mathbb{P}(X \times Y)$ . Une relation est ainsi un ensemble de paires ordonnées. Dans le cas où aucun élément du type source ne peut être relié à deux éléments ou plus du type cible, la relation est une *Fonction*. Une fonction totale ( $\rightarrow$ ) est celle dont chaque élément de l'ensemble source est relié, tandis que une fonction partielle ( $\mapsto$ ) est telle que il existe des éléments de la source non reliés. Le *Domaine* (*dom*) d'une relation ou fonction renferme les éléments de l'ensemble source qui sont reliés, et le *Range* (*ran*) renferme les éléments de l'ensemble cible qui sont reliés.

Le langage schéma est utilisé pour structurer et composer les descriptions : présenter les informations, les encapsuler, et les nommer pour une réutilisation. La réutilisabilité est nécessaire pour la réussite d'une application des techniques formelles. En identifiant et partageant les composantes communes, on maintient des descriptions flexibles et facile à gérer. Dans le langage de schémas, on observe les spécifications qui partagent des parties, les preuves qui partagent des arguments, les théories qui partagent des abstractions et les problèmes qui partagent des aspects communs. On pense que l'utilisation de schémas aide à proposer un meilleur style de spécification. Cependant, comme c'est le cas pour toute notation, le langage de schémas nécessite une application attentive et judicieuse. On doit faire attention lors de développement de théories simples et lors de l'utilisation de schémas pour les présenter de façon élégante et compréhensible.

### 3.2.2 Le raffinement de données dans Z

Écrire une spécification formelle est une activité dont la réussite nécessite une simple description et une bonne compréhension. Cependant, nous devons, aussi, développer une spécification de manière à ce qu'elle nous amène à une implémentation convenable. Ce processus de développement est appelé *raffinement*. Nous raffinons une spécification formelle en ajoutant plus de détails relatifs à l'implémentation. Par exemple, nous devons être plus précis dans la façon d'organiser les données, ou dans la manière de mettre en œuvre certains calculs. Précisément, il est important que notre nouvelle description plus détaillée soit consistante avec la spécification originale : le raffinement doit être correct.

Dans notre contexte, le raffinement sert à améliorer les spécifications. Ce processus d'amélioration permet d'enlever le non déterminisme ou l'incertitude. Une spécification abstraite peut laisser certains choix de conception non résolus ; un raffinement peut résoudre certains de ces choix et éliminer le non déterminisme. Plusieurs étapes de raffinement peuvent être accomplies, chacune enlève un degré d'incertitude, jusqu'à ce que la spécification s'approche du code exécutable des programmes.

Ainsi, nous développons un système par la construction d'un modèle de conception utilisant des types de données mathématiques simples pour identifier le comportement désiré. Nous devons, ensuite, raffiner cette description par la construction d'un autre modèle qui respecte les décisions de conception faites et qui est proche de l'implémentation. Quand c'est nécessaire, ce processus de raffinement peut produire un code exécutable. La notation Z est, alors, un langage mathématique avec un mécanisme de structuration robuste.

Dans le contexte de la notation Z, l'unité de spécification est *un schéma*. Le raffinement d'une telle spécification exige le *raffinement de données (data)* [DE94] [Spi92] et le *raffinement des opérations (operation)* respectivement pour le schéma de données et le schéma d'opération. Puisque nous n'employons pas les schémas d'opération ( $\Delta$  schemas), nous ne nous sommes pas intéressés par leur raffinement (raffinement d'opération). Par contre, nous introduisons une relation de raffinement

de comportement qui permet de substituer des comportements abstraits par des comportements concrets.

### 3.2.3 Les preuves dans Z

Une spécification Z peut être écrite dans une variété de styles. Cependant, il est convenable d'utiliser, dans plusieurs cas, une approche basée sur des états ou des modèles. Un système peut être modélisé comme un état abstrait et une séquence d'opérations sur cet état. Les opérations sont considérées atomiques. Une fois une conception est faite, il est utile de prouver certaines propriétés au comportement du système. Ceci aide à vérifier la conception et repérer les erreurs. Cette vérification consiste à effectuer certaines obligations de preuves dont chacune se présente sous la forme d'une règle d'inférence. L'ensemble de ces règles d'inférence forme le système de preuve qui se base sur la déduction [Spi92]. Chaque règle est écrite dans la forme suivante :

$$\frac{\textit{premiss}_1, \dots, \textit{premiss}_n}{\textit{conclusion}} \quad [ \textit{name} ]$$

La signification d'une telle règle est que la validité de la *conclusion* découle de la validité des *prémisses* : une fois les *prémisses* sont vraies, il en est, alors, de même pour la *conclusion*. La liste des prémisses est parfois vide. Les règles sont utilisées pour deux raisons. Pour un opérateur logique *op*, la règle d'élimination de *op* décrit qu'est ce qu'il peut être déduit à partir de  $p \textit{ op } q$ ; et la règle d'introduction de *op* décrit sous quelles conditions  $p \textit{ op } q$  peut être conclue. En utilisant ces règles pour introduire et éliminer différents opérateurs, on peut commencer à partir d'un ensemble de propositions ou d'hypothèses et dériver d'autres propositions. Si l'ensemble des hypothèses est vide, alors on appelle la proposition dérivée *un théorème*.

Ce processus peut être fastidieux et surtout s'il est en totalité manuel; mais, une fois qu'il est implémenté, il est très pertinent pour réduire les erreurs et comprendre les opérations du système. Finalement, les preuves formelles servent à montrer qu'une spécification raffine une autre permettant, ainsi, d'ajouter de la qualité

au développement de logiciel et de garantir la faisabilité de l'utilisation industrielle d'une méthode formelle. Il est possible de raffiner une conception abstraite en une implémentation concrète en appliquant une série d'étapes de raffinement d'états et de comportements. Chaque étape de raffinement est reliée à la précédente par une relation mathématique. Il y a un ensemble de règles de preuves gouvernant les étapes de raffinement valides.

Divers outils pour vérifier les types et guider les preuves dans Z sont disponibles. A titre d'exemples, nous en citons :

- le package Fuzz [Spi00], un vérificateur de type et de syntaxe avec une option de style et de fontes LaTeX [Lam94] qui est disponible à partir du Spivey Partnership. Il est compatible avec la seconde édition de Spivey's Z Reference Manual [Spi92]. C'est une collection d'outils qui aident à composer et imprimer les spécifications Z et à les examiner pour assurer la conformité avec les règles du langage Z mais sans vérifier le sens de la spécification. Une partie de la collection est une option de style qui présente des commandes supplémentaires de LaTeX [Lam94] pour les spécifications Z, et une police de caractères qui contient les symboles spéciaux de Z. L'autre partie est un programme pour analyser et vérifier les spécifications écrites en utilisant les commandes LaTeX. Fuzz est écrit en C, il est commercialisé et il n'accepte que le format LaTeX en entrée.
- ZTC [FT98] permet de vérifier la syntaxe des spécifications Z tout en ignorant le texte informel. Il autorise deux formats d'entrées : LaTeX et ZSL (version ASCII de Z). ZSL est conçu pour être lisible et essaye de maintenir l'aspect visuel des spécifications Z autant que possible sous forme d'une représentation graphique. ZTC peut effectuer des traductions entre LaTeX et ZSL.
- FORSITE [Bow87] est développé pour l'utilisation des méthodes formelles afin de spécifier les systèmes en temps réel. Il permet, essentiellement, l'analyse et la vérification des spécifications Z et l'indexation des schémas.
- ZETA [BG98] est un environnement ouvert pour le développement des spécifications Z en format LaTeX. Il fournit un cadre d'intégration des outils pour éditer et analyser les spécifications Z. ZETA contient le vérificateur

et l'analyseur qui met en application tout le langage Z et fournit en plus de nouvelles extensions. ZETA contient aussi l'environnement Hol-Z [KSW96] qui permet la preuve des spécifications Z en se basant sur le tireur de preuve de théorème générique Isabelle [NPW04]. ZETA fournit une représentation graphique en langage Java et implémente le moteur d'exécution en C++, mais l'intégration de nouveaux composants ne peut être faite qu'en Java.

Le plus grand inconvénient de ces outils est qu'ils ne supportent que la vérification de la syntaxe et du type des schémas Z. Ils ne s'intéressent pas à la preuve des propriétés de spécifications. En plus, certains de ces outils ne supportent pas tous les composants de la notation Z, alors que d'autres ont essayé d'améliorer les spécifications Z par de nouveaux concepts mathématiques.

Afin de vérifier nos spécifications, nous utilisons l'outil Z-EVES [MS99] qui permet de vérifier la syntaxe selon la notation Z et de prouver des propriétés au sujet du comportement du système spécifié. Z-EVES est un logiciel libre, une version gratuite est disponible pour toute utilisation universitaire. Il est portable, et il est disponible pour Linux, Windows et Solaris. En plus, une base de documentation assez riche est développée pour faciliter l'utilisation de Z-EVES et pour mieux comprendre son système de raisonnement de vérification et de preuve des théorèmes. Le système Z-EVES supporte presque toute la notation Z. L'interaction avec Z-EVES utilise la notation Fuzz [Spi00]. Z-EVES supporte le Toolkit mathématiques dont les théorèmes ont été formulés afin d'être utilisés automatiquement dans le vérificateur de théorème (`theorem prouver`).

### 3.2.4 Les avantages et les limites de Z

Z est utile pour la spécification des systèmes, permettant une meilleure compréhension avant l'exécution, et réduisant le nombre d'erreurs. Z peut être employé pour produire des spécifications lisibles en utilisant une notation mathématique. Les mathématiques fournissent la précision, à la différence du langage naturel et des diagrammes qui sont souvent employés. Les spécifications Z sont présentées sous une forme bien déterminée selon la notation schématique. Il est, aussi, possible de produire des spécifications hiérarchiques. La notation Z gagne

graduellement l'acceptation dans l'industrie.

Par contre,  $Z$  n'est pas adéquat pour la description des propriétés non fonctionnelles, telle que la performance. De plus, il ne permet pas la description des comportements réactifs temporels ou concurrents. Cependant, il y a d'autres méthodes formelles qui sont convenables pour de telles descriptions. Ainsi, une solution envisageable consiste à combiner ces méthodes avec  $Z$  en vue de bien spécifier les différentes propriétés du système [MM05].

### 3.3 La logique temporelle linéaire (LTL)

La logique temporelle [MP92] est une extension de la logique conventionnelle (propositionnelle). Elle intègre de nouveaux opérateurs qui expriment la notion du temps. En effet, par exemple, il n'est pas possible d'exprimer dans la logique classique une assertion liée au comportement d'un programme telle que "après exécution d'une instruction  $i$ , le système se bloque". Dans cette assertion, les actions s'exécutent suivant un axe de temps : à l'instant  $t$ , exécution de l'instruction  $i$ , et à  $t + 1$  blocage du système. Il faut donc une logique qui modélise les expressions du passé et du futur.

La logique temporelle linéaire ou LTL (Linear Temporal Logic) permet de représenter le comportement des systèmes réactifs au moyen de propriétés qui décrivent l'évolution du système. Dans le cadre de la logique temporelle linéaire le temps se déroule, comme son nom l'indique, linéairement. Ainsi, nous spécifions le comportement attendu d'un système tel que dans la figure 3.1 où une séquence d'actions qui se suivent.

Les logiques temporelles permettent de représenter et de raisonner sur certaines propriétés de sûreté des systèmes. En ce sens, elles sont bien adaptées à la spécification et à la vérification des systèmes réactifs et concurrents. Elles pourraient donc être adaptées à la spécification des SMA. Néanmoins, il ne faut pas oublier que ces logiques ont été conçues pour raisonner sur des propriétés usuelles des systèmes concurrents comme l'invariance ("de mauvaises choses ne peuvent pas

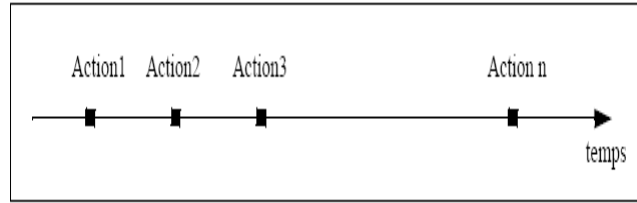


FIG. 3.1 – Le déroulement temporel d’un système dans une logique linéaire

se produire”), la vivacité (“les bonnes choses vont effectivement se produire”), ou encore la précédence (“il faut avoir préalablement demandé une ressource pour espérer l’obtenir”).

Dans la section suivante nous présentons une logique temporelle linéaire proposée par Manna et Pnueli [MP92]. Cette logique a, depuis, été grandement améliorée d’un point de vue expressif par l’ajout de nombreux opérateurs. Mais, les principes de base restent les mêmes.

### 3.3.1 Le Langage de LTL

On peut résumer la sémantique des logiques temporelles par la description du modèle de la logique et de sa sémantique :

1. Les opérateurs de la logique : la logique contient :
  - l’ensemble des connecteurs booléens :  $\vee, \wedge, \neg, \Rightarrow, \Leftrightarrow$  ;
  - les quantificateurs du premier ordre  $\forall$  et  $\exists$  ;
  - les opérateurs temporels :  $\circ$  (‘next’=prochain),  $\square$  (‘always’=toujours),  $\diamond$  (‘eventually’= inévitablement ou finalement) et  $U$  (‘until’=jusqu’à).
2. Les règles de formation des formules
  - les termes de la logique sont formés par l’application de fonctions à des constantes et des variables.
  - les formules sont construites par l’application sur les termes de connecteurs

- booléens, d'opérateurs temporels ou de quantificateurs.
- les formules atomiques sont les propositions et les prédicats appliqués aux termes de types compatibles.
3. Le modèle de LTL : Un modèle  $(I, \alpha, \delta)$  consiste en une interprétation globale  $I$ , une affectation globale  $\alpha$ , et une séquence d'états  $\delta$ .
- $I$  désigne un domaine de validité correspondant à chaque type et affecte des éléments concrets de ce domaine aux symboles.
  - $\alpha$  affecte une valeur d'un domaine approprié aux variables libres globales et aux propositions.
  - $\delta = s_0, s_1, s_2, \dots$  est une séquence infinie d'états où chaque  $s_i$  affecte des valeurs aux variables libres locales et aux propositions.
4. Interprétation des formules : Dans cette sous-section, nous présentons une définition inductive de l'interprétation de la valeur de vérité d'une formule temporelle  $t$  dans le modèle  $(I, \alpha, \delta)$ . La valeur d'un terme ou d'une sous-formule  $w$  est notée  $w \mid_{\delta}^{\alpha}$  c'est à dire  $w$  dans le modèle  $(I, \alpha, \delta)$  où  $I$  est sous-entendu.
- Interprétation générale des expressions
    - Pour une variable ou une proposition locale  $y : y \mid_{\delta}^{\alpha} = y_{s_0}$ . La valeur de  $y$  dans l'état  $s_0$  est celle qui lui est affectée dans le premier état de  $\delta$ .
    - Pour une variable ou une proposition globale  $u : u \mid_{\delta}^{\alpha} = \alpha[u]$ . La valeur de  $u$  est celle affectée à  $u$  par  $\alpha$ .
    - Pour une constante  $c$ , l'évaluation est déterminée par  $I : c \mid_{\delta}^{\alpha} = I[c]$ .
    - Pour une fonction,  $f(x_1, \dots, x_k) : f(t_1, \dots, t_k) \mid_{\delta}^{\alpha} = I[f](t_1 \mid_{\delta}^{\alpha}, \dots, t_k \mid_{\delta}^{\alpha})$ . La valeur de la fonction est celle de la fonction interprétée,  $I[f]$ , appliquée aux valeurs de  $t_1, \dots, t_k$  dans le modèle  $(I, \alpha, \delta)$ .
    - Pour un prédicat  $p(x_1, \dots, x_k) : p(t_1, \dots, t_k) \mid_{\delta}^{\alpha} = I[p](t_1 \mid_{\delta}^{\alpha}, \dots, t_k \mid_{\delta}^{\alpha})$ .
    - Pour une disjonction :  $(w_1 \vee w_2) \mid_{\delta}^{\alpha} = \text{true}$  : si et seulement si  $w_1 = \text{vrai}$  ou  $w_2 = \text{vrai}$ .
    - Pour une négation :  $\neg w \mid_{\delta}^{\alpha} = \text{vraie}$  si et seulement si  $w \mid_{\delta}^{\alpha} = \text{faux}$ .
  - Interprétation des opérateurs temporels
    - Pour l'application  $\circ : \circ w \mid_{\delta}^{\alpha} = w \mid_{\delta^1}^{\alpha}$ . La valeur de  $\circ w$  (next  $w$ ) dans

$\delta = s_0, s_1, s_2, \dots$  est donnée par la valeur de  $t$  dans la séquence décalée

$\delta^1 = s_1, s_2, \dots$

- Pour l'application  $\Box : \Box w \mid_{\delta}^{\alpha} = \text{vrai}$  si et seulement si pour tout ( $k \geq 0$ ),  $w \mid_{\delta}^{\alpha} = \text{vraie}$ . 'always  $w$ ' est vrai si et seulement si  $w$  est vrai pour toute séquence suffixe de  $\delta$ .
- Pour l'application  $\Diamond : \Diamond w \mid_{\delta}^{\alpha} = \text{vrai}$  si et seulement si il existe ( $k \geq 0$ ),  $w \mid_{\delta}^{\alpha} = \text{vraie}$ . 'Eventually  $w$ ' est vrai si et seulement si  $w$  est vrai sur au moins une séquence suffixe de  $\delta$ .
- Pour l'application  $U : w_1 U w_2 \mid_{\delta}^{\alpha} = \text{vrai}$  si et seulement si pour ( $k \geq 0$ ),  $w_2 \mid_{\delta^k}^{\alpha} = \text{vrai}$  et que pour tout  $i$ ,  $0 \leq i < k$ ,  $w_1 \mid_{\delta^i}^{\alpha} = \text{vrai}$ . ' $w_1$  Until  $w_2$ ' est vrai si et seulement si  $w_1$  est vrai continûment à partir de  $s_i$  et jusqu'à ce que  $w_2$  soit vraie.

### 3.3.2 Le système déductif de LTL

Dans la logique temporelle, un système déductif se constitue des éléments suivants :

- un ensemble d'*axiomes* : c'est un ensemble de formules valides prises comme des propriétés de base des opérateurs de ce langage.
- un ensemble de *règles* : il s'agit des règles à travers lesquelles de nouvelles formules valides peuvent être dérivées à partir d'autres formules dont la validité a été précédemment établie. Une règle possède la forme suivante :

$$p_1, p_2, \dots, p_m \vdash q$$

Elle consiste en une liste de formules  $p_1, p_2, \dots, p_m$  appelées *prémises*, et une formule  $q$  appelée la *conclusion* de la règle. Une telle règle exprime que si nous avons, déjà, établi la validité de  $p_1, p_2, \dots, p_m$ , alors nous pouvons inférer la validité de  $q$ . Un exemple de règle est la règle fréquemment utilisée de *modus ponens*  $p, p \rightarrow q \vdash q$ , qui infère la validité de  $q$  à partir de la validité de  $p$  et de  $p \rightarrow q$ .

### 3.3.3 Les preuves dans LTL

Étant donné un système déductif  $H$  constitué de plans d'axiome et de règles, nous exprimons la construction d'une dérivation, appelée *une preuve*, qui établit la validité d'une formule utilisant les axiomes et les règles.

Un élément important dans la construction de preuves à partir d'axiomes et de règles est la notion d'*instantiation*. Soit  $\psi$  une formule (plan) et  $p_1, p_2, \dots, p_r$  certains symboles qui apparaissent dans  $\psi$ . Un remplacement de  $p_1, p_2, \dots, p_r$  qu'on représente par  $\alpha : [p_1 \longleftarrow \varphi_1, p_2 \longleftarrow \varphi_2, \dots, p_r \longleftarrow \varphi_r]$ , spécifie pour chaque  $p_i$ ,  $i = 1, \dots, r$ , une formule remplaçante  $\varphi_i$ . On dénote par  $\psi[\alpha]$  la formule obtenue à partir de  $\psi$  en remplaçant toute occurrence de  $p_1, p_2, \dots, p_r$  par  $\varphi_1, \varphi_2, \dots, \varphi_r$ , respectivement. On considère  $\psi[\alpha]$  comme une instantiation de  $\psi$ . Par exemple, la formule  $\Box \diamond q \vee \diamond \neg \diamond q$  est une instantiation de la formule  $\Box p \vee \diamond \neg p$ , utilisant le remplacement  $[p \longleftarrow \diamond q]$ .

Une preuve dans  $H$  est une séquence de lignes dont chacune contient une formule  $p$  (éventuellement temporelle). Une telle ligne présente la validité prouvée de  $p$ . Dans la représentation de preuves, on ajoute, habituellement, à chaque ligne une courte *justification* qui exprime la base de l'inclusion de cette ligne dans la preuve. Chaque ligne doit être supportée par un axiome ou l'application d'une règle.

## 3.4 $\mathcal{T}_{temporal}\mathcal{Z}$ : intégration de Z et LTL

La logique temporelle linéaire, comme présentée par Manna et Pnueli [MP92], est un outil approprié pour spécifier et vérifier les systèmes concurrents et réactifs. En fait, il y a une variété d'opérateurs temporels qui peuvent être candidats pour exprimer les propriétés comportementales d'un agent. Ces opérateurs peuvent être définis en terme de deux opérateurs de base. Dans notre travail, nous proposons l'utilisation, seulement, des opérateurs nécessaires pour le développement des applications à base d'agent. Dans ce qui suit, nous présentons brièvement ces opérateurs avec une explication intuitive. Soit  $P$  et  $Q$  deux formules logiques ou temporelles :

$\nabla P$   $P$  s'effectue "*maintenant*" ( $\nabla$  peut être omis);

- $\square P$  “*toujours*”  $P$ , c’est-à-dire  $P$  s’effectue dans le présent et dans tous les futurs points de temps ;
- $\diamond P$  “*inévitablement*”  $P$ , c’est-à-dire  $P$  aura lieu à un certain point de temps actuel ou futur ;
- $\circ P$  “*temps suivant*”  $P$ , c’est-à-dire  $P$  aura lieu au prochain point de temps.

Afin d’intégrer ces opérateurs temporels dans le framework de  $Z$ , nous avons présenté dans [RKJ04b] un nouveau type libre de  $Z$  appelé *Formula*. Nous distinguons les formules atomiques (*Action*), qui sont étroitement liées à l’application à spécifier et les formules temporelles (*Formula*) qui relient des formules de prédicats aux opérateurs temporels.

$$Formula ::= \nabla \langle\langle Action \rangle\rangle \mid \circ \langle\langle Formula \rangle\rangle \mid \square \langle\langle Formula \rangle\rangle \mid \diamond \langle\langle Formula \rangle\rangle$$

Nous prouverons, plus tard, que ces opérateurs sont suffisants pour exprimer les propriétés souhaitées des applications à base d’agents relatives aux aspect statique et comportemental.

Ainsi, dans  $\mathcal{T}_{temporal}Z$ , un schéma se présente comme suit :

<i>SchemaName</i> _____
<i>Declarations</i>
$F_1 \dots, F_n$

où les  $F_i$  ( $i = 1, 2, \dots, n$ ) sont des disjonctions de formules de type *Formula*.

Cette intégration syntaxique est accompagnée par une intégration sémantique, présentée dans la section suivante. De ce fait, ces deux niveaux d’intégration vont permettre d’utiliser les outils de supports de  $Z$  tout en maintenant la puissance et le pouvoir expressif de la logique temporelle.

### 3.4.1 Le modèle temporel

En plus de l’intégration syntaxique, nous proposons une intégration sémantique qui assure l’évaluation des formules selon un modèle temporel bien défini. Ce modèle

est spécifié, en  $Z$ , sous forme d'une abréviation présentant une fonction totale qui associe à chaque instant de temps ( $Time == \{x : \mathbb{N}\}$ ) l'ensemble des actions ( $\mathbb{F} Action$ ) exécutées par les agents du système. En se basant sur ce modèle temporel, nous pourrions interpréter nos formules temporelles.

$$Model == Time \rightarrow \mathbb{F} Action$$

Dans notre spécification, nous supposons que les actions produites par chaque agent entre deux instants successifs  $t_i$  et  $t_{i+1}$  sont exécutées, approximativement, à l'instant  $t_{i+1}$  afin de faciliter l'utilisation et la spécification du modèle (la figure 3.2).

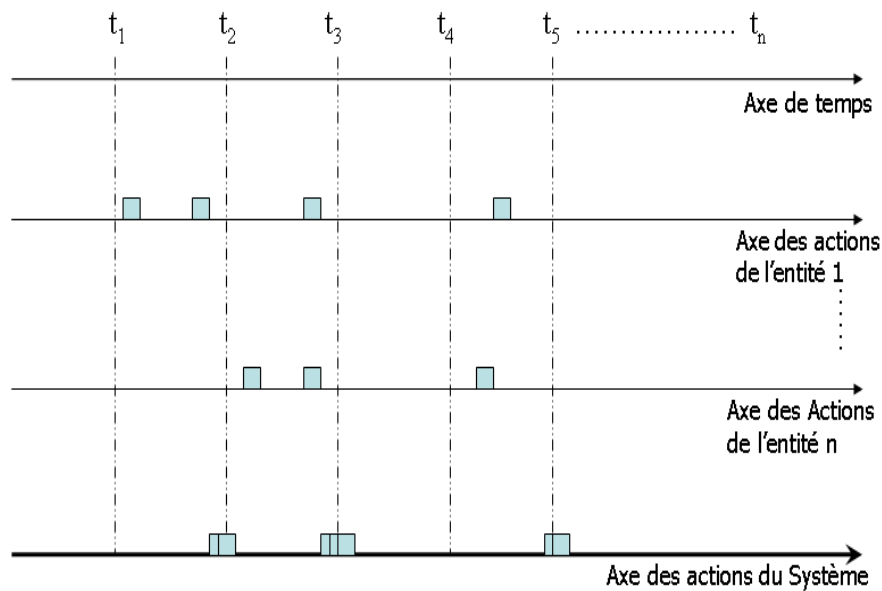


FIG. 3.2 – La représentation des actions des différentes entités par rapport à un axe de temps

### 3.4.2 La sémantique des formules temporelles

Dans cette section, nous définissons la sémantique de notre logique temporelle en terme de fonctions axiomatiques. Cette étape est très significative puisqu'elle nous permet de traduire des formules temporelles en langage  $Z$ . Ainsi, il devient facile d'exploiter les outils automatiques de vérification, tels que Z-EVES ou Isabelle-HOL, qui acceptent simplement la syntaxe standard de  $Z$ .

La fonction axiomatique ( $E$ ) interprète une formule temporelle dans un modèle donné et à un point de temps donné.

Les définitions axiomatiques suivantes décrivent, respectivement, les évaluations de  $\Box f$ ,  $\Diamond f$  et  $\circ f$  :

$$\left| \begin{array}{l} E : Formula \times Model \times Time \\ \hline \forall f : Formula \bullet \forall m : Model \bullet \forall t : Time \bullet \\ E((\Box f), m, t) \Leftrightarrow (\forall t1 : Time \mid t1 \geq t \bullet (f, m, t1) \in E) \end{array} \right.$$

$$\left| \begin{array}{l} E : Formula \times Model \times Time \\ \hline \forall f : Formula \bullet \forall m : Model \bullet \forall t : Time \bullet \\ E((\Diamond f), m, t) \Leftrightarrow (\exists t1 : Time \mid t1 \geq t \bullet (f, m, t1) \in E) \end{array} \right.$$

$$\left| \begin{array}{l} E : Formula \times Model \times Time \\ \hline \forall f : Formula \bullet \forall m : Model \bullet \forall t : Time \bullet \\ E((\circ f), m, t) \Leftrightarrow (\exists t1 : Time \mid t1 = t + 1 \bullet (f, m, t1) \in E) \end{array} \right.$$

Nous généralisons la fonction  $E$  en faisant une première abstraction du paramètre temps. Ainsi, la fonction ( $Eva$ ) interprète une formule temporelle dans le cadre d'un modèle donné :

$$\left| \begin{array}{l} Eva : Formula \times Model \\ \hline \forall f : Formula \bullet \forall m : Model \bullet \\ Eva(f, m) \Leftrightarrow (\forall t : Time \bullet (f, m, t) \in E) \end{array} \right.$$

Nous pouvons généraliser davantage la fonction d'interprétation par une abstraction du modèle. La fonction suivante définit une interprétation générale des opérateurs temporels :

$$\left| \begin{array}{l} Eval : Formula \\ \hline \forall f : Formula \bullet Eval(f) \Leftrightarrow (\forall m : Model \bullet (f, m) \in Eva) \end{array} \right.$$

Finalement, afin d'employer les opérateurs temporels dans leurs notations habituelles (par exemple,  $\square$  pour “toujours”) dans le schéma  $Z$ , nous les présentons comme des fonctions axiomatiques définies avec la fonction d'interprétation *Eval*. Ainsi, nous établissons une équivalence logique entre un opérateur temporel et le prédicat correspondant spécifié dans la fonction ci-dessus *Eval*.

Cette équivalence est décrite pour les opérateurs  $\square$ ,  $\diamond$  et  $\circ$ , respectivement, comme suit :

$$\square f \Leftrightarrow Eval(\square f)$$

$$\diamond f \Leftrightarrow Eval(\diamond f)$$

$$\circ f \Leftrightarrow Eval(\circ f)$$

### 3.4.3 La relation de raffinement

#### Les principes de base

Afin de définir la relation de raffinement entre les spécifications en  $\mathcal{T}_{temporal}\mathcal{Z}$ , nous adoptons, avec quelques restrictions, la relation de  $\sqsubseteq$  définie dans [DE94]. Pour ce qui concerne le raffinement de données, déjà présenté dans la section 3.2.2, un schéma raffine un autre schéma (noté  $S_i \sqsubseteq S_j$ ) si et seulement si les attributs de  $S_i$  sont inclus dans la partie déclaration de  $S_j$ .

Un schéma raffiné peut soit :

1. renfermer tout le schéma à raffiner (inclusion de schéma) tel est le cas de  $S_j$  raffinant  $S_i$  :



$S_j$
$S_i$ ( <i>schema inclusion</i> )
$Att_3 : Type_3$
$Operations$

ou,

2. contenir, parmi les attributs de la partie déclarative, tous les attributs du schéma à raffiner tel est le cas de  $S'_j$  raffinant aussi le même schéma  $S_i$  :

$S'_j$
$Att_1 : Type_1$ ( <i>attribute of <math>S_i</math></i> )
$Att_2 : Type_2$ ( <i>attribute of <math>S_i</math></i> )
$Att_3 : Type_3$
$Operations$

Concernant le raffinement comportemental, il est assuré en ajoutant des formules temporelles représentant quelques décisions d'implémentation. Selon [JP03], une spécification temporelle  $Tpr_2$  (en termes de formules temporelles) est un raffinement d'une autre  $Tpr_1$  (noté par  $Tpr_2 \vdash Tpr_1$ ), si toutes les formules de  $Tpr_1$  peuvent être dérivées à partir des formules de  $Tpr_2$  [MP92]; c'est à dire que la validité de  $Tpr_1$  peut être inférée de la validité de  $Tpr_2$  en utilisant des axiomes et des règles telles que présentées dans la section 3.3.2.

Ainsi, si  $S_i$  et  $S_j$  sont des spécifications en  $\mathcal{T}_{temporal} \mathcal{Z}$ ,  $S_j$  est un raffinement de  $S_i$  si et seulement si  $S_i \sqsubseteq S_j$  (limité à la partie données *data*) et les formules temporelles dans  $S_i$  peuvent être dérivées des formules temporelles de  $S_j$  [RKJ05b, RKJ04a].

### Les propriétés de raffinement

Étant donnée une spécification  $Spec_0$  écrite en  $\mathcal{T}_{temporal} \mathcal{Z}$ , la relation de raffinement entre deux spécifications  $Spec_0$  et  $Spec_1$  est garantie par la satisfaction du théorème suivant :

Soit  $Attr_0$  et  $Attr_1$  les ensembles d'attributs, respectivement, de  $Spec_0$  et  $Spec_1$ , et  $Tpr_0$  et  $Tpr_1$  les formules temporelles, respectivement, de  $Spec_0$  et  $Spec_1$  :

**theorem RefRelation**

$$Attr_0 \subseteq Attr_1 \wedge Tpr_1 \vdash Tpr_0 \Rightarrow Spec_0 \sqsubseteq Spec_1$$

Pour cette relation de raffinement, nous pouvons définir deux propriétés permettant l'achèvement du processus de raffinement qui sera décrit dans le chapitre suivant (le chapitre 4). Ces propriétés sont présentées comme suit :

- **La transitivité** : Étant donné  $Spec_0 \sqsubseteq Spec_1$  et  $Spec_1 \sqsubseteq Spec_2$ , la propriété de transitivité permet la déduction de la relation de raffinement entre  $Spec_0$  et  $Spec_2$  :  $Spec_0 \sqsubseteq Spec_2$ . Cette propriété est assurée par la satisfaction du théorème suivant :

Soit  $Spec_0$ ,  $Spec_1$  et  $Spec_2$  trois spécifications,

**theorem Transitivity**

$$Spec_0 \sqsubseteq Spec_1 \wedge Spec_1 \sqsubseteq Spec_2 \Rightarrow Spec_0 \sqsubseteq Spec_2$$

- **La composition** : Cette propriété exprime que : si un système  $S$  peut être décomposé en  $n$  sous systèmes  $S = s_1 \oplus \dots \oplus s_n$  et chaque sous système  $s_i$  peut être raffiné en  $s'_i$  (noté  $s_i \sqsubseteq s'_i$ ), alors la composition des raffinements des différents sous systèmes  $S' = s'_1 \oplus \dots \oplus s'_n$  constitue un raffinement du système initial (noté  $S \sqsubseteq S'$ ) :

Soit  $S$  un système qui peut être décomposé en  $n$  sous systèmes,

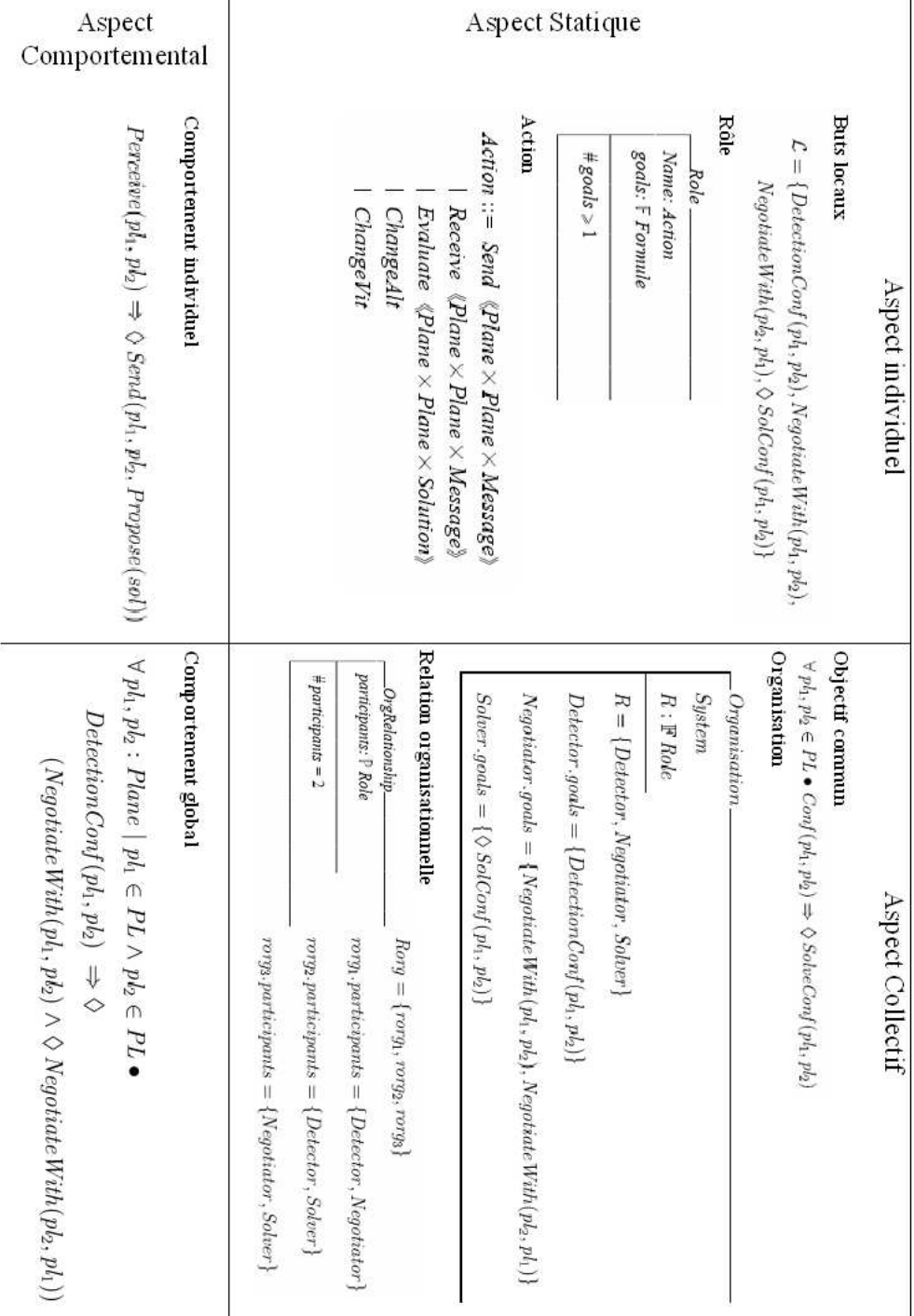
**theorem Compositionality**

$$S = s_1 \oplus \dots \oplus s_n \wedge s_1 \sqsubseteq s'_1 \wedge \dots \wedge s_n \sqsubseteq s'_n \wedge S' = s'_1 \oplus \dots \oplus s'_n \Rightarrow S \sqsubseteq S'$$

### 3.4.4 Le pouvoir expressif de $\mathcal{T}_{temporal}\mathcal{Z}$

L'intégration de Z et de LTL dans  $\mathcal{T}_{temporal}\mathcal{Z}$  accorde à ce dernier la capacité d'exprimer aussi bien les aspects statiques (grâce à Z) que les aspects comportementaux (grâce à LTL) qui caractérisent un SMA.

Dans cette section, nous proposons de montrer cette capacité à travers certains exemples décrivant les principaux concepts dans le contexte d'une application à

FIG. 3.3 – La description des principaux concepts des SMA avec  $\mathcal{T}_{temporal} \mathcal{Z}$

base d'agents se rapportant aussi bien à l'aspect individuel que collectif et mettant en évidence aussi bien à l'aspect statique que comportemental. Ces exemples sont décrits dans la figure 3.3. L'aspect individuel est exprimé à travers des notions tels que buts locaux, rôle et actions exprimant l'aspect statique d'un agent, et la notion de comportement individuel décrivant l'aspect comportemental. Concernant l'aspect collectif,  $\mathcal{T}_{temporal}\mathcal{Z}$  permet aussi de représenter les concepts de l'aspect statique tels que objectif commun, organisation et relation organisationnelle ainsi que les concepts liés à l'aspect comportemental à savoir les comportements globaux.

En se basant sur le langage  $\mathcal{T}_{temporal}\mathcal{Z}$  ainsi que la relation de raffinement décrite ci-dessus, nous développons une méthode de conception formelle de SMA qui accentue la vérification formelle des étapes de raffinement.

### 3.5 Conclusion

Dans ce chapitre, nous avons proposé un langage de spécification qui entre dans le cadre du multi-formalisme par la proposition de l'intégration de la logique temporelle linéaire dans la notation  $\mathcal{Z}$ . Ainsi, nos spécifications peuvent arriver à couvrir les aspects statiques et comportementaux des SMA.

Notre choix est basé sur le fait que la notation  $\mathcal{Z}$  permet de présenter et de spécifier formellement l'aspect statique qui décrit l'état du système et de l'agent, alors que la logique temporelle intervient à la spécification des aspects comportementaux sur l'axe de temps permettant d'exprimer l'évolution de l'état du système au cours du temps. L'intégration des opérateurs temporels dans les schémas  $\mathcal{Z}$  est exprimée par la définition de la syntaxe et de la sémantique de ces opérateurs.

Étant donnée cette intégration, la relation de raffinement consiste, donc, à la combinaison de *raffinement de données* qui concerne la notation  $\mathcal{Z}$  et la *dérivation* de formules concernant la logique temporelle. Ainsi, nous pouvons définir tout un processus de raffinement qui se base sur cette relation en vue d'arriver à concevoir le comportement des agents dans un niveau proche de l'implémentation. Ce processus sera décrit dans le chapitre suivant.



## Chapitre 4

# *FormAAD* : une approche formelle de développement d'applications à base d'agents

La spécification formelle est une phase primordiale dans toute démarche de conception et d'implémentation de systèmes critiques. En effet, cette phase élimine toute ambiguïté et imprécision dans la description du système et permet de raisonner rigoureusement sur ses propriétés structurelles telles que but local et rôle, et comportementales à savoir les comportements individuels et les comportements globaux. Compte tenu de l'hétérogénéité des composants d'un SMA (entités n'ayant pas les mêmes propriétés structurelles et comportementales), la complexité de ses mécanismes d'interaction et le manque d'un consensus au sujet de ses concepts fondamentaux, la spécification formelle devient de plus en plus pesante, afin de maîtriser cette complexité. Ainsi, il est fondamental de modéliser les concepts de base de toute application multi-agent, d'unifier leur représentation et de définir les relations entre eux.

Les travaux de [LHKJ04] proposent une tentative de modélisation conceptuelle générique de l'interaction utilisant la notation Z. Cette modélisation traite les concepts décrivant une activité de coopération et de négociation. Aussi, ils ont proposé la spécification des propriétés nécessaires pour maintenir une cohérence à l'échelle individuelle (Agent) et à l'échelle collective (Société). Cette spécification a été vérifiée syntaxiquement et sémantiquement par l'outil Z-EVES [MS99].

Les concepts liés à la coordination ont été ajoutés avec ceux de la coopération et de la négociation [JKR02] afin d'améliorer l'action du groupe. Le groupement des concepts liés à la coopération, la négociation et la coordination permettent, ainsi, la définition conceptuelle d'un modèle générique d'interaction. Nous pensons que les modèles de coopération et de négociation [KJ99] [KJ01] constituent un premier pas pour la conception de SMA coopératifs. Dans [JKR02], nous avons validé sémantiquement ces modèles auxquels nous avons associé l'aspect coordination. Cette validation est essentiellement axée sur la description du comportement des agents tout au long de leur activité coopérative.

Les modèles conceptuel et opérationnel présentés dans ces travaux sont considérés comme une première réflexion pour le développement d'une méthode fondée de conception descendante des SMA. Dans le cycle de développement d'une telle méthode, nous distinguons deux étapes majeures. La première consiste à définir un langage de spécification approprié faisant intervenir les concepts pertinents de l'agent et du SMA. Ce langage a été déjà présenté dans le chapitre précédent (chapitre 3). La deuxième étape présente plutôt une démarche définissant un nombre d'étapes et les règles de passage entre elles [KRJ07]. Cette démarche se base sur un processus de raffinement utilisant le langage  $\mathcal{T}_{temporal}\mathcal{Z}$ . Ce processus est incrémental et consiste à suivre une suite d'étapes successives dont chacune raffine la précédente en vue de générer une conception vérifiée conforme à la spécification abstraite de départ. Ainsi, un processus de vérification accompagne celui de raffinement et consiste à valider le passage d'une étape à une autre en se servant d'un outil de support du langage adopté. Cette vérification consiste à prouver la relation de raffinement liant les spécifications de deux étapes successives. Les différentes étapes sont accompagnées par des aides méthodologiques qui guident le processus de conception. Certaines étapes peuvent nécessiter une intervention humaine en vue d'effectuer un choix ou d'apporter une définition pour une action spécifique à l'application. Les étapes de la méthode sont intégrées dans un outil d'aide que nous avons développé en vue d'assister et de guider le concepteur à suivre la succession d'étapes partant d'une spécification initiale jusqu'à une conception qui détaille les comportements des différentes entités participantes dans l'achèvement de l'objectif commun.

Dans ce chapitre nous détaillons cette démarche [KRJ07].

## 4.1 Une méthode formelle de développement des applications à base d'agent

Le langage  $\mathcal{T}_{temporal}\mathcal{Z}$  (voir chapitre 3) est utilisé dans la définition d'une approche de conception fournissant quelques principes qui aident et guident le processus de conception. Dans cette section, certains de ces principes sont clarifiés. Plus précisément, notre approche [RKJ05b] est basée sur deux phases principales : (1) La première est une phase de spécification des besoins dans laquelle nous décrivons, de manière abstraite, les besoins de l'application en terme d'un objectif commun pour les agents. (2) La seconde est une phase de conception dans laquelle des spécifications détaillées sont dérivées en se basant sur une succession de raffinements des comportements collectifs (inter-agents) et individuels (intra-agent). La vérification que les spécifications résultantes satisfont les exigences est considérée comme une tâche essentielle qui est progressivement accomplie pendant les étapes de raffinement.

### 4.1.1 Phase de spécification

Dans cette phase, nous définissons les besoins qui correspondent à un objectif commun à atteindre par les agents du système à concevoir. Dans notre approche, cette étape inclut, également, la spécification initiale de l'environnement dans lequel les agents évoluent et qui, généralement, se compose d'un environnement et d'un ensemble d'entités passives (objets).

#### Spécification des entités

Une entité désigne un objet de l'environnement. Cet objet est caractérisé par un ensemble d'attributs (caractéristiques) statiques et/ou comportementaux. Une entité est initialement passive (pas de comportement); une fois elle est dotée de comportement, elle est dite active (un agent).

Une entité est décrite par un schéma  $Z$  défini en terme d'un ensemble d'attributs

et de formules logiques décrivant sa partie structurelle. Dans le cas d'une entité active, nous enrichissons le schéma  $Z$  avec des formules temporelles spécifiant son comportement. La spécification d'une entité active sera raffinée durant le processus de conception.

Un schéma  $Z$  spécifiant une entité possède la forme suivante :

<i>Entity</i>
<i>atr</i> <sub>1</sub> : <i>Type</i> <sub>1</sub> , <i>atr</i> <sub>2</sub> : <i>Type</i> <sub>2</sub> ... <i>atr</i> <sub><i>m</i></sub> : <i>Type</i> <sub><i>m</i></sub>
<i>Spr</i> <sub>1</sub> , ..., <i>Spr</i> <sub><i>n</i></sub>
<i>Cpr</i> <sub>1</sub> , ..., <i>Cpr</i> <sub><i>k</i></sub>

Où chaque *atr*<sub>*i*</sub> correspond à un attribut, chaque *Spr*<sub>*i*</sub> représente une propriété structurelle et chaque *Cpr*<sub>*i*</sub> représente une propriété comportementale.

### Spécification du système

Cette spécification décrit l'environnement dans lequel les agents seront exécutés. Elle inclut, principalement, les entités passives appartenant à l'espace de travail. En outre, elle peut inclure les entités actives (agents) si ces dernières sont connues dès le début.

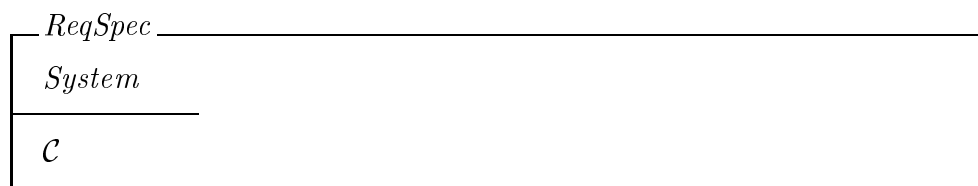
Dans cette spécification, nous présentons des formules reliant les entités passives avec celles actives. Généralement, ceci mène à un schéma  $Z$  de la forme :

<i>System</i>
<i>Objects</i> : $\mathbb{F}$ <i>Entity</i>
<i>Agents</i> : $\mathbb{F}$ <i>Entity</i>
<i>Pr</i> <sub>1</sub> , <i>Pr</i> <sub>2</sub> , ..., <i>Pr</i> <sub><i>i</i></sub>

Où *Objects* est un ensemble d'entités passives, *Agents* est un ensemble d'agents (ceux qui sont connus à l'avance), et chaque *Pr*<sub>*i*</sub> est une formule temporelle.

## Spécification des besoins

Dans notre approche, la spécification des besoins correspond à un ensemble de formules temporelles  $\mathcal{C}$  spécifiant un *objectif commun* à réaliser par un groupe d'agents évoluant dans l'environnement donné. Un objectif commun décrit un état futur désiré du système. Selon l'approche Z, ces spécifications correspondent à une spécialisation de schéma *System*. Ceci est représenté comme suit :



Dans la phase de conception, nous procédons à transformer la spécification du système (*System*) en raffinant la description des agents capables d'achever l'objectif commun selon une stratégie de coopération, une structure organisationnelle et des comportements collectifs et individuels.

### 4.1.2 Phase de conception

L'idée de base de la phase de conception consiste à proposer une séquence de raffinements en utilisant la spécialisation des schémas Z pour le raffinement de données et la dérivation des formules temporelles pour le raffinement du comportement. Les étapes de raffinement sont supportées par un ensemble de règles [RKJ05a]. Les raffinements apparaissent à deux niveaux complémentaires (voir la figure 4.1). Le premier niveau est le niveau collectif où la spécification *System* est enrichie par des propriétés référant, essentiellement, aux aspects collectifs (inter-agent) caractérisant, en particulier, les structures organisationnelle et communicationnelle. Le deuxième niveau concerne plutôt les aspects individuels (intra-agent) en raffinant les spécifications des entités décrites dans la première phase (la section 4.1.1). En outre, il y a la possibilité d'ajouter de nouveaux agents et de les raffiner si nécessaire.

Notre processus de développement proposé est structuré en sept étapes de raffinement comme le montre la figure 4.1. La première étape (désignée par les flèches

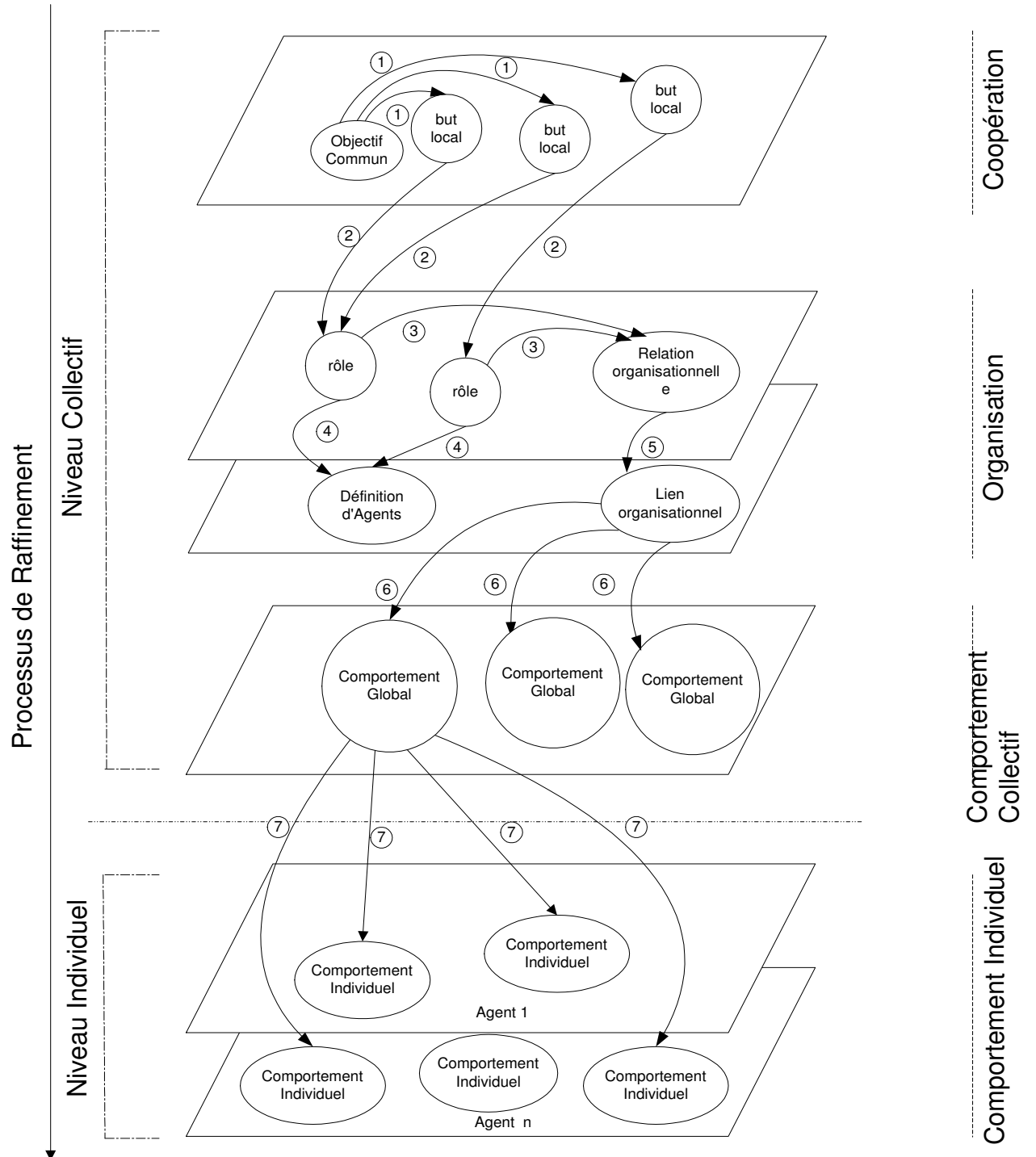


FIG. 4.1 – Le processus de conception basé sur sept étapes de raffinement

numérotées (1)) définit une stratégie de coopération pour achever l'objectif commun. Ceci consiste à décomposer cet objectif en un ensemble de sous-objectifs, appelés buts locaux. La définition d'une structure organisationnelle est assurée en deux étapes (les flèches (2) et (3)). En premier lieu, nous identifions les rôles en groupant les buts locaux; ensuite, nous les relierons par les relations adéquates. Simultanément, nous assignons les rôles aux agents (les flèches numérotées (4)).

Généralement, un agent peut jouer différents rôles et différents agents peuvent avoir le même rôle. Les relations entre les rôles sont, ainsi, translatées, dans le niveau agent, en liens organisationnels (les flèches numérotées (5)) qui représentent les interactions entre les agents du système. En se basant sur ces liens, nous identifions les comportements collectifs nécessaires sous forme de formules temporelles globales (étape (6)). Finalement, il reste à définir les comportements individuels appropriés aux agents (étape (7)). Il est à noter que la vérification que les spécifications résultantes satisfassent les spécifications des besoins est considérée comme une tâche essentielle qui est, progressivement, assurée durant les étapes de raffinement. Toutes ces étapes seront détaillées dans ce qui suit.

## Niveau collectif

La conception des aspects collectifs se focalise sur trois points : la stratégie de coopération, la structure organisationnelle et les comportements collectifs.

- Stratégie de coopération

- *Étape 1 : définition de la stratégie de coopération*

La définition d'une stratégie de coopération consiste à définir un ensemble de buts élémentaires (locaux) qui seront réalisés par les agents. Afin de déterminer ces buts, nous réitérons la réécriture de l'objectif commun  $\mathcal{C}$  jusqu'à atteindre sa forme normale conjonctive. Cette dernière est une conjonction d'un ensemble de clauses  $c_i$ , où chacune est une disjonction des formules temporelles indécomposables. Par conséquent, nous produisons un graphe et/ou présentant la décomposition de l'objectif commun. Sa racine représente l'objectif commun, tandis que, les nœuds feuilles représentent des buts locaux.

Nous dénotons par  $\mathcal{L}$  l'ensemble de ces buts locaux.

<i>Implementation</i> <sub>0</sub>
<i>System</i>
$\mathcal{L} : \mathbb{F} \textit{LocalGoal}$
$c_1, c_2, \dots, c_m$

où

$[\textit{LocalGoal}]$

désigne les buts locaux.

Cette étape de raffinement nécessite la preuve du théorème *CoopStrategy* qui garantit le fait que *Implementation*<sub>0</sub> raffine *ReqSpec*. Ce raffinement provient du fait que nous pouvons, à partir de la conjonction des  $c_i$ , déduire  $\mathcal{C}$  :

**theorem CoopStrategy**

$$c_1, c_2, \dots, c_m \vdash \mathcal{C} \qquad \langle\langle 1 \rangle\rangle$$

Une clause  $c_i$  peut avoir deux formes. La première est un simple prédicat et ainsi, il s'agit d'un élément de  $\mathcal{L}$ . Dans le deuxième cas, la clause  $c_i$  est une disjonction de prédicats simples  $c_{ij} : c_i = \bigvee_{j=1}^k c_{ij}$  ; ainsi, chaque  $c_{ij}$  est un élément de  $\mathcal{L}$ .

A titre d'exemple, soit  $A$  un objectif commun défini comme suit :

$$\textit{build}(\textit{house}) \wedge \diamond (\textit{paint}(\textit{walls}, \textit{house}) \wedge \diamond \textit{furnish}(\textit{house}))$$

L'opérateur  $\diamond$  (inévitablement), précédant la conjonction, ne peut pas être associé aux atomes. Dans ce cas, nous considérons un nouvel objectif commun  $A'$  défini par :

$$\textit{paint}(\textit{walls}, \textit{house}) \wedge \diamond \textit{furnish}(\textit{house})$$

La forme normale conjonctive de  $A'$  est :

$$\textit{paint}(\textit{walls}, \textit{house}) \wedge \diamond (\textit{furnish}(\textit{kitchen}) \vee \textit{furnish}(\textit{room}))$$

Si nous remplaçons  $A'$  par sa forme normale conjonctive, nous obtenons la décomposition suivante de  $A$  :

$$\textit{build}(\textit{house}) \wedge \diamond (\textit{paint}(\textit{walls}, \textit{house}) \wedge \diamond (\textit{furnish}(\textit{kitchen}) \vee \textit{furnish}(\textit{room})))$$

Ainsi, l'ensemble de buts locaux identifiés est :

$$\mathcal{L} = \{build(house), paint(walls, house), furnish(kitchen), furnish(room)\}$$

Cette décomposition est montrée dans la figure 4.2.

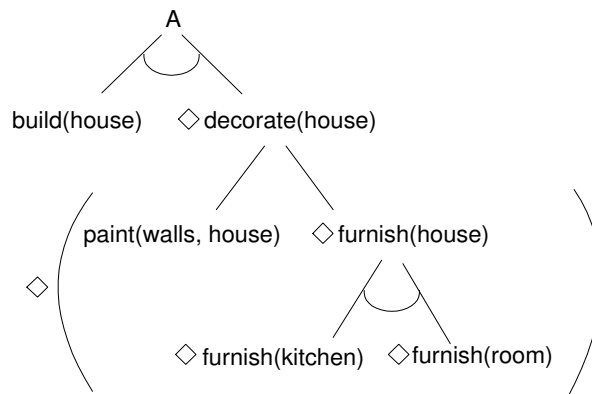


FIG. 4.2 – Le graphe et/ou de l'objectif commun  $A$

Ainsi, nous aurons la spécification suivante :

<i>Implementation<sub>0</sub></i>	_____
<i>System</i>	
$\mathcal{L} : \mathbb{F} LocalGoal$	
	$build(house), paint(walls, house), furnish(kitchen) \vee furnish(room)$
	$\mathcal{L} = \{build(house), paint(walls, house), furnish(kitchen), furnish(room)\}$

#### • Structure organisationnelle

La structure organisationnelle définit implicitement une stratégie de contrôle à respecter par les entités actives. Cette organisation est, généralement, définie en terme de formules temporelles. Dans cette phase, nous dérivons une structure organisationnelle adéquate pour le système à développer en identifiant les rôles nécessaires (étape 2 et étape 3). Ensuite, nous assignons chaque rôle à une ou plusieurs entités actives du système (étape 4 et étape 5).

- *Étape 2 : identification des rôles*

Cette étape consiste à définir les principaux rôles nécessaires pour l'achèvement des buts locaux identifiés dans l'étape précédente. Ici, un rôle (*Role*) est, formellement, représenté par un ensemble de formules temporelles. Ainsi, nous assurons cette étape en groupant les prédicats ayant les mêmes noms.

Formellement, nous définissons un rôle par son nom (*Name*) et un ensemble de buts (*goals*) renfermant ses buts locaux :

<i>Role</i>
$Name : Act$ $goals : \mathbb{F} LocalGoal$
$\#goals \geq 1$ $\forall g : LocalGoal \mid g \in goals \bullet g.action = Name$

Pour un rôle donné de type *Role*, les buts locaux appartenant à l'ensemble *goals* ont tous la même action qui correspond à *Name* du rôle.

Pour illustrer cette étape, nous détaillons l'objectif commun *A* décrit dans l'étape précédente. L'ensemble de buts locaux présente trois noms de prédicats : *build*, *paint* et *furnish*. Ainsi, nous définissons trois rôles : *role*<sub>1</sub>, *role*<sub>2</sub> et *role*<sub>3</sub> appelés respectivement *build*, *paint* et *furnish*.

Ainsi, cette étape aboutit à la spécification raffinée suivante :

<i>Implementation</i> <sub>1</sub>
$System$ $\mathcal{L} : \mathbb{F} LocalGoal$ $R : \mathbb{F} Role$
$\forall b : LocalGoal \mid b \in \mathcal{L} \bullet \exists r : Role \mid r \in R \bullet b \in r.goals$

La partie prédicative indique que tout but local appartenant à l'ensemble  $\mathcal{L}$  doit faire partie de l'ensemble *goals* d'un rôle de l'ensemble *R*.

La preuve à assurer dans cette étape de raffinement ( $Implementation_1 \sqsubseteq Implementation_0$ ) est apportée par le théorème suivant :

**theorem Completeness**

$$Implementation_1 \Rightarrow \bigcup_{r \in R} r.goals = \mathcal{L} \quad \langle\langle 2 \rangle\rangle$$

Ce théorème assure que  $Implementation_1$  garantit que chaque but local appartient à l'ensemble  $goals$  d'un rôle et que l'union des différents ensembles  $goals$  des rôles de  $R$  couvre tous les buts locaux de  $\mathcal{L}$ .

Pour l'exemple de l'étape précédente, nous obtenons le raffinement suivant :

$Implementation_1$
$System$
$\mathcal{L} : \mathbb{F} LocalGoal$
$R : \mathbb{F} Role$
$\exists role1, role2, role3 : R \bullet role1.goals = \{build(house)\} \wedge$ $role2.goals = \{paint(walls, house)\} \wedge$ $role3.goals = \{furnish(kitchen), furnish(room)\}$

- *Étape 3 : définition des relations organisationnelles*

Notre proposition, dans cette étape, est d'établir les relations organisationnelles entre les rôles définis. En pratique, nous essayons d'identifier les arguments communs pour les prédicats décrivant les buts locaux des différents rôles. Un argument commun entre deux rôles indique une relation organisationnelle entre ces rôles. Finalement, nous obtenons un ensemble de relations organisationnelles [ $OrgRelationship$ ] connectant les rôles retenus. Ici, nous faisons référence au modèle conceptuel présenté dans [LHKJ04] où une structure organisationnelle générique  $OrgStructure$  est définie comme un graphe de dépendances entre les rôles.

Cette étape aboutit à un schéma de la forme suivante :

<i>Implementation<sub>2</sub></i>	_____
<i>System</i>	
$R : \mathbb{F} \textit{Role}$	
$Rorg : \mathbb{F} \textit{OrgRelationship}$	
_____	
$\forall r : \textit{Role} \mid r \in R \bullet \exists rorg : \textit{OrgRelationship} \mid rorg \in Rorg \bullet$	
$r \in rorg.\textit{participants}$	

Chaque rôle appartenant à  $R$  doit participer à au moins une relation organisationnelle de l'ensemble  $Rorg$ .

Dans ce contexte, chaque rôle peut être relié à un ou plusieurs autres rôles. Cette propriété est vérifiée en prouvant le théorème *RoleParticipant* :

**theorem RoleParticipant**

$$\textit{Implementation}_2 \Rightarrow \bigcup_{org \in Rorg} org.\textit{participants} = R \quad \langle\langle 3 \rangle\rangle$$

La preuve de ce théorème garantit la validité de cette étape de raffinement.

Considérant l'exemple présenté dans l'étape précédente, nous notons l'existence du paramètre commun *house* pour les buts locaux : *build(house)* et *paint(walls, house)* qui correspondent, respectivement, aux  $role_1$  et  $role_2$ . Ce résultat indique que  $role_1$  et  $role_2$  participent dans une relation organisationnelle.

Ainsi, pour cet exemple, nous aurons :

<i>Implementation<sub>2</sub></i>	_____
<i>System</i>	
$R : \mathbb{F} \textit{Role}$	
$Rorg : \mathbb{F} \textit{OrgRelationship}$	
_____	
$\exists rorg1 : \textit{OrgRelationship} \mid Rorg = \{rorg1\} \bullet$	
$rorg1.\textit{participants} = \{role1, role2\}$	

- *Étape 4 : affectation des rôles*

Étant donné l'ensemble des rôles et l'ensemble des relations entre eux, cette étape consiste à identifier, de manière générique, les agents nécessaires et à leur assigner

les rôles retenus. A ce niveau, nous ne considérons aucun détail d'implémentation sur les agents.

Il est à noter que plusieurs rôles peuvent être assignés à un même agent et un rôle peut être assigné à un ou plusieurs agents.

Afin d'assigner les rôles aux agents, nous avons besoin de spécifier un ordre de précedence entre les buts locaux composants les rôles considérés. Cet ordre dépend des opérateurs temporels utilisés pour décrire les buts locaux.

En pratique, nous développons la forme normale disjonctive de  $\mathcal{C}$ , qui décrit une liste de scénarios différents pour l'achèvement de cet objectif à partir de la forme normale conjonctive déjà trouvée. Chaque scénario est décrit par une séquence d'atomes ( $c_{ij}$ ). En se basant sur cette forme disjonctive, nous établissons un graphe de précedence pour chaque scénario. Un graphe de précedence est un graphe directe acyclique où les noeuds représentent des activités et les arcs représentent des relations de causalité entre elles.

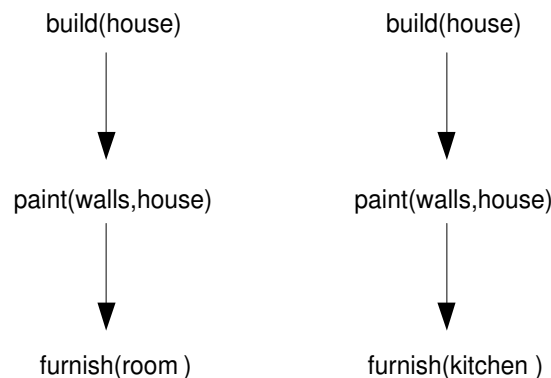


FIG. 4.3 – Les graphes de précedence des deux scénarios

En appliquant ces consignes, l'ordre de précedence entre les buts locaux de l'objectif commun  $A$ , décrits dans les étapes précédentes, est présenté dans la figure 4.3. Il correspond à la forme normale disjonctive suivante qui montre deux scénarios connectés par un *ou* ( $\vee$ ) :

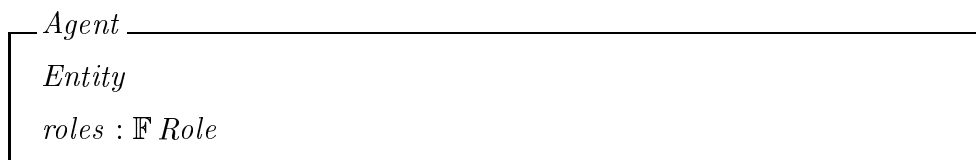
$$(build(house) \wedge \diamond (paint(walls, house) \wedge \diamond furnish(kitchen))) \vee$$

$$(build(house) \wedge \diamond (paint(walls, house) \wedge \diamond furnish(room)))$$

Le but local  $build(house)$  est le premier but à achever. Nous pouvons assigner ce but local à un agent qui va avoir le rôle  $role_1$ . Soit  $ag_1$  cet agent. Le but local  $paint(walls, house)$  commencera après l'achèvement de  $build(house)$ . Ainsi, nous pouvons l'assigner à  $ag_1$  ou à un autre agent (par exemple  $ag_2$ ) qui va avoir le rôle  $role_2$ . De ce fait, nous identifions deux scénarios d'affectation pour ce rôle. Dans ce qui suit, nous allons considérer l'affectation de  $role_2$  à  $ag_2$ . Ainsi, les buts locaux  $furnish(kitchen)$  et  $furnish(room)$  peuvent être achevés simultanément après la terminaison de  $paint(walls, house)$ . Dans ce cas, ils peuvent être affectés à deux agents différents, par exemple  $ag_1$  et  $ag_2$ . Ces deux agents auront le rôle  $role_3$ . Par conséquent, nous obtenons le scénario qui adopte deux agents  $ag_1$  et  $ag_2$  ayant, respectivement, les rôles  $\{role_1, role_3\}$  et  $\{role_2, role_3\}$ .

Il est à noter que nous ne sommes pas concerné, dans ce travail, d'évaluer les différents scénarios d'affectation des rôles aux agents qui peuvent être parfois non optimaux (nombre élevé d'agents, interaction et communication intense). Ce problème d'optimisation peut être résolu en tenant compte des futur étapes de raffinement et ceci en permettant des retours, si nécessaires, pour modifier un scénario choisi.

A ce niveau, nous pouvons définir le schéma *Agent* comme un premier raffinement du schéma *Entity* où nous ajoutons l'attribut *roles* :



Ensuite, nous pouvons raffiner davantage la spécification *System* comme suit :

$Implementation_3$
$System$
$R : \mathbb{F} Role$
$Rorg : \mathbb{F} OrgRelationship$
$Ag : \mathbb{F} Agent$
$Ag = \{ e : Agent \mid e.roles \subseteq R \}$
$\forall r \in Role \bullet \exists e : Agent \mid e \in Ag \bullet r \in e.roles$

Cette spécification raffinée indique que les rôles de chaque agent de  $Ag$  sont inclus dans l'ensemble  $R$ . Aussi, pour chaque rôle de  $R$ , il existe au moins un agent de  $Ag$  qui possède ce rôle.

Afin de vérifier que chaque rôle est assigné à au moins un agent, nous devons prouver le théorème *RoleAssignment* :

**theorem RoleAssignment**

$$Implementation_3 \Rightarrow \bigcup_{a \in Ag} a.roles = R \quad \langle\langle 4 \rangle\rangle$$

Cette preuve garantit que  $Implementation_3$  raffine  $Implementation_2$  :  
 $(Implementation_3 \sqsubseteq Implementation_2)$ .

Étant donné ce raffinement, la spécification de l'exemple sera :

$Implementation_3$
$System$
$R : \mathbb{F} Role$
$Rorg : \mathbb{F} OrgRelationship$
$Ag : \mathbb{F} Agent$
$\exists ag1, ag2 : Agent \mid Ag = \{ ag1, ag2 \} \bullet$ $ag1.roles = \{ role1, role3 \} \wedge ag2.roles = \{ role2, role3 \}$

- *Étape 5 : définition des accointances (liens organisationnels)*

Cette étape est consacrée à l'instanciation des relations d'organisation afin de fixer des liens d'organisation entre les agents selon l'assignation des rôles. Un lien d'organisation entre deux agents correspond à une relation d'organisation entre leurs rôles correspondants. Par conséquent, chaque relation *OrgRelationship* entre deux rôles, évoquée dans le niveau précédent, mène à un lien d'organisation entre les agents ayant ces rôles. Formellement, un lien d'organisation est défini comme suit :

$\textit{OrganisationLink}$ <hr style="border: 0.5px solid black;"/> $E : \mathbb{F} \textit{Agent}$ $Lg : \mathbb{F} \textit{LocalGoal}$ <hr style="border: 0.5px solid black;"/> $\#E \geq 2$
---

Où  $E$  est l'ensemble d'agents participants à ce lien et dont la cardinalité doit être au moins égale à 2 ; et  $Lg$  représente l'ensemble des buts locaux sujet du lien.

Cette étape conduit à la spécification raffinée suivante :

$\textit{Implementation}_4$ <hr style="border: 0.5px solid black;"/> $\textit{System}$ $R : \mathbb{F} \textit{Role}$ $Rorg : \mathbb{F} \textit{OrgRelationship}$ $Ag : \mathbb{F} \textit{Agent}$ $\textit{organizationlinks} : \mathbb{F} \textit{OrganizationLink}$ <hr style="border: 0.5px solid black;"/> $\forall e : \textit{Agent} \bullet \exists Or \in \textit{organizationlinks} \bullet e \in Ag \Leftrightarrow e \in Or.E$
---

La partie prédicative de cette spécification indique que chaque agent appartenant à  $Ag$  doit participer à au moins un lien organisationnel de *organizationlinks*.

La preuve de cette étape de raffinement ( $\textit{Implementation}_4 \sqsubseteq \textit{Implementation}_3$ ) est composée de deux théorèmes (⟨⟨5⟩⟩ et ⟨⟨6⟩⟩) montrant que tout lien organisationnel instancie une relation organisationnelle (le théorème ⟨⟨5⟩⟩) et que toute relation organisationnelle est instanciée par un ou plusieurs liens organisationnels (le théorème

⟨⟨6⟩⟩ :

**theorem Instantiation1**

$$\begin{aligned} Implementation_4 \Rightarrow \forall ol : OrganizationLink \bullet \exists or : OrgRelationship \bullet \\ ol \in organizationlinks \Rightarrow or \in Rorg \end{aligned} \quad \langle\langle 5 \rangle\rangle$$

**theorem Instantiation2**

$$\begin{aligned} Implementation_4 \Rightarrow \forall or : OrgRelationship \bullet \exists O : \mathbb{F} OrganizationLink \bullet \\ or \in Rorg \Rightarrow O \subseteq organizationlinks \end{aligned} \quad \langle\langle 6 \rangle\rangle$$

Ainsi, chaque relation organisationnelle dans  $Rorg$  génère un ensemble de liens organisationnels inclus dans  $organizationlinks$ .

Afin d'illustrer cette étape, nous considérons la relation organisationnelle  $rorg_1$  entre  $role_1$  et  $role_2$ . Cette relation sera instantiée afin de générer un ensemble de liens organisationnels entre les agents. Dans l'étape précédente,  $role_1$  a été affecté à  $ag_1$  pour achever  $build(house)$  et  $role_2$  a été affecté à  $ag_2$  pour achever  $paint(walls, house)$ . Ainsi, nous aurons besoin d'introduire un lien organisationnel  $lorg_1$  qui instantie  $rorg_1$  et reliant ces deux agents. Ce lien sert, par exemple, à ordonner à  $ag_2$  d'achever la construction du  $house$  par  $ag_1$ . Cette information est requise afin de commencer l'action  $paint$ . Nous aurons, ainsi, la spécification suivante :

$Implementation_4$
$System$
$R : \mathbb{F} Role$
$Rorg : \mathbb{F} OrgRelationship$
$Ag : \mathbb{F} Agent$
$organizationlinks : \mathbb{F} OrganizationLink$
$\exists lorg1 : OrganizationLink \mid lorg1 \in organizationlinks \bullet$
$lorg.E = \{ag1, ag2\} \wedge lorg.Lg = \{build(house), paint(walls, house)\}$

• Comportements collectifs

- Étape 6 : définition des comportements collectifs

Le but de cette étape est de définir le comportement collectif des agents selon leurs rôles et leurs liens d'organisation. Ce comportement est exprimé en terme de formules (temporelles) globales où chacune se rapporte à plus qu'un agent. Dans notre approche, ces formules peuvent être déduites en se basant sur les liens d'organisation établis entre les agents. En général, un lien d'organisation mène à une formule globale décrivant une relation de causalité entre les buts locaux inclus dans ce lien. Notons que grâce aux opérateurs temporels, la détermination des propriétés globales, à partir de la relation causale entre les buts et les liens d'organisation, devient une activité relativement simple.

Par exemple, le lien organisationnel  $lorg_1$  entre les agents ( $ag_1$  et  $ag_2$ ), ayant respectivement  $role_1$  et  $role_2$ , va aider à raffiner un ensemble de propriétés globales se référant à  $ag_1$  et  $ag_2$  dans le cadre de l'achèvement de leurs buts locaux. Ces buts locaux sont  $build(house)$  et  $paint(walls, house)$ . Ainsi, tenant compte de la relation de causalité définie entre ces buts, nous pouvons déduire la formule globale suivante qui représente un comportement collectif qui concerne  $ag_1$  et  $ag_2$ . Cette formule atteste que le but  $paint(walls, house)$  ne peut être démarré par  $ag_2$  qu'après avoir terminé le but  $build(house)$  par  $ag_1$  :

$$ag_1.build(house) \Rightarrow \diamond ag_2.paint(walls, house)$$

Cette étape aboutit au schéma suivant (*Implementation<sub>5</sub>*) qui inclut toutes les formules globales déterminées :

<p><i>Implementation<sub>5</sub></i></p> <hr style="border: 0; border-top: 1px solid black; margin: 5px 0;"/> <p><i>System</i></p> <p><math>R : \mathbb{F} \textit{Role}</math></p> <p><math>Rorg : \mathbb{F} \textit{OrgRelationship}</math></p> <p><math>Ag : \mathbb{F} \textit{Agent}</math></p> <p><math>organizationlinks : \mathbb{F} \textit{OrganizationLink}</math></p> <hr style="border: 0; border-top: 1px solid black; margin: 5px 0;"/> <p><math>GlobBehav_1, GlobBehav_2, \dots, GlobBehav_k</math></p>
--

Ces formules doivent prouver que chaque  $GlobBehav_i$  provient d'un lien organisationnel  $ol_i$ .

Pour l'exemple, nous aurons le raffinement suivant :

$Implementation_5$
$System$
$R : \mathbb{F} Role$
$Rorg : \mathbb{F} OrgRelationship$
$Ag : \mathbb{F} Agent$
$organizationlinks : \mathbb{F} OrganizationLink$
$ag_1.build(house) \Rightarrow \diamond ag_2.paint(walls, house)$

En vue de vérifier que  $Implementation_5$  raffine  $Implementation_4$ , le théorème *GlobBehavDefinition* garantit que nous pouvons, toujours, déduire chaque lien organisationnel, à partir d'un sous ensemble de comportements collectifs.

**theorem GlobBehavDefinition**

$$\forall l \in organizationlinks \bullet \\ \exists Gb \subseteq \{GlobBehav_1, GlobBehav_2, \dots, GlobBehav_k\} \bullet Gb \vdash l \quad \langle\langle 7 \rangle\rangle$$

En effet, cette étape contribue à un ensemble de formules temporelles qui décrivent le comportement collectif des agents. En particulier, ces formules décrivent l'activité coopérative des agents selon l'organisation définie dans les étapes précédentes. Il reste à déterminer les comportements individuels assurant le comportement collectif désiré. Ainsi, nous faisons apparaître les activités de communication et de coordination nécessaires.

**Niveau individuel**

L'intérêt principal de cette étape est de décrire le comportement individuel de chaque agent. En pratique, ceci consiste à remplacer, dans la spécification, chaque formule globale par des formules individuelles dont chacune réfère à un seul agent. Ainsi, chaque formule globale  $GlobBehav_i$  sera remplacée par un ensemble  $\{IndBehav_1, IndBehav_2, \dots, IndBehav_n\}$  de formules individuelles.

- *Étape 7 : définition des comportements individuels*

Afin de déterminer les décisions conceptuelles concernant les différents comportements d'un agent, nous commençons par prouver les propriétés globales établies dans l'étape précédente. De cette manière, nous présentons les propriétés additionnelles qui assurent la preuve. Ces nouvelles propriétés mènent à des spécifications raffinées avec les structures et/ou les propriétés additionnelles, auxquelles le même processus de dérivation doit être appliqué de nouveau. Le processus se termine quand les propriétés globales sont prouvées en se basant seulement sur des propriétés individuelles.

Cette étape aboutit à un schéma ayant la même partie déclarative que  $Implementation_5$ ; alors que, dans la partie prédicative, les formules individuelles  $IndBehav_i$  remplacent les formules globales de la spécification précédente.

$Implementation_6$
$System$
$R : \mathbb{F} Role$
$Rorg : \mathbb{F} OrgRelationship$
$Ag : \mathbb{F} Agent$
$organizationlinks : \mathbb{F} OrganizationLink$
<hr style="width: 20%; margin-left: 0;"/> $IndBehav_1, IndBehav_2, \dots, IndBehav_i$

La preuve de ce raffinement est donnée par le théorème  $IndBehavDerivation$  qui montre que nous pouvons déduire les propriétés globales à partir des formules individuelles se trouvant dans  $Implementation_6$ .

**theorem IndBehavDerivation**

$$Implementation_6 \Rightarrow GlobBehav_1, GlobBehav_2, \dots, GlobBehav_k \quad \langle\langle 8 \rangle\rangle$$

Nous notons que la preuve de ce théorème est assurée durant la détermination des propriétés individuelles.

Par exemple, si nous considérons la propriété globale :

$$build(ag_1, house) \Rightarrow \diamond paint(ag_2, walls, house)$$

sa preuve nécessite un mécanisme permettant à  $ag_1$  d'informer  $ag_2$  sur l'achèvement de son action :

$$send(ag_1, ag_2, finish(build(house)))$$

Nous supposons que la communication entre agents est fiable. Ainsi, chaque action  $send$  correspond à une action  $receive$  :

$$send(ag_i, ag_j, inf) \Leftrightarrow receive(ag_j, ag_i, inf) \quad [eq_1]$$

La preuve de la propriété globale sera, alors, comme suit :

$$\begin{aligned} build(ag_1, house) &\Rightarrow \diamond send(ag_1, ag_2, finish(build(house))) && [Fi_1] \\ \Leftrightarrow build(ag_1, house) &\Rightarrow \diamond receive(ag_2, ag_1, finish(build(house))) && [eq_1] \\ \Leftrightarrow build(ag_1, house) &\Rightarrow \diamond paint(ag_2, walls, house) && [Fi_2] \end{aligned}$$

Les formules individuelles ajoutées sont :

$$\begin{aligned} Fi_1 &: build(ag_1, house) \Rightarrow \diamond send(ag_1, ag_2, finish(build(house))) \\ Fi_2 &: receive(ag_2, ag_1, finish(build(house))) \Rightarrow \diamond paint(ag_2, walls, house) \end{aligned}$$

Ainsi, nous obtenons le raffinement suivant :

$Implementation_6$
<i>System</i>
$R : \mathbb{F} Role$
$Rorg : \mathbb{F} OrgRelationship$
$Ag : \mathbb{F} Agent$
$organizationlinks : \mathbb{F} OrganizationLink$
<hr style="width: 80%; margin-left: 0;"/>
$build(ag_1, house) \Rightarrow \diamond send(ag_1, ag_2, finish(build(house)))$
$receive(ag_2, ag_1, finish(build(house))) \Rightarrow \diamond paint(ag_2, walls, house)$

Cette étape ne nécessite pas, seulement, l'insertion de nouvelles formules temporelles, mais aussi, un ensemble d'actions pour lesquelles nous supposons que les agents peuvent assurer. Ainsi, nous enrichissons la spécification raffinée par un ensemble

d'Actions. Par exemple :

$$\begin{aligned} \text{Action} ::= & \text{send}\langle\langle \text{Agent} \times \text{Agent} \times \text{Message} \rangle\rangle \mid \text{build}\langle\langle \text{Agent} \times \text{House} \rangle\rangle \mid \\ & \text{receive}\langle\langle \text{Agent} \times \text{Agent} \times \text{Message} \rangle\rangle \mid \text{paint}\langle\langle \text{Agent} \times \text{Walls} \times \text{House} \rangle\rangle \end{aligned}$$

La phase de conception mène à des spécifications détaillées du système et à des comportements détaillés et des capacités des entités actives. Les spécifications raffinées correspondent au schéma du système (*System*) augmenté par les propriétés supplémentaires aux niveaux collectif et individuel. Ainsi, par la transitivité de la relation de raffinement, nous arrivons à prouver que la spécification détaillée (*Implementation<sub>6</sub>*) raffine les besoins initiaux (*ReqSpec*). En particulier, nous pouvons déduire l'objectif commun ( $\mathcal{C}$ ) à partir des formules individuelles données dans *Implementation<sub>6</sub>*. Le théorème «9» exprime cette déduction :

**theorem VerifSpec**

$$Fi_1, Fi_2, \dots, Fi_m \vdash \mathcal{C} \qquad \langle\langle 9 \rangle\rangle$$

## 4.2 $\mathcal{F}_{or}\mathcal{MAAD}\mathcal{T}ools$ :

La méthode  $\mathcal{F}_{or}\mathcal{MAAD}$  se décompose en deux grandes phases : la phase de spécification et la phase de conception. Ces phases ont fait l'objet d'une intégration dans un outil d'aide à la conception nommé  $\mathcal{F}_{or}\mathcal{MAAD}\mathcal{T}ools$  qui sera sujet de cette section.

### 4.2.1 Environnement de développement

Pour l'implémentation de cet atelier, nous avons intégré les outils et les “packages” développés dans le cadre du projet CZT [XU07]. CZT (Community Z Tools) offre un ensemble d'outils pour éditer, vérifier le typage et animer des spécifications formelles écrites en Z. Son objectif est de permettre l'échange de spécifications Z entre des outils existants (via un format d'échange XML Standard pour Z).

En particulier CZT offre un plugin JEdit qui permet de lui ajouter un éditeur de spécifications Z en LaTeX ou en unicode, un parseur, un vérificateur de type intégré

et un convertisseur entre LaTeX, unicode et XML. ZML (XML markup for the Z) est basé sur le standard ISO pour Z, ce format est spécifié par un schéma XML. Ceci présente la syntaxe des spécifications qui seront écrites en ZML. CZT offre aussi un package Java qui permet de programmer des outils utilisant des spécifications Z. Ainsi, l'atelier est développé en langage Java en se servant de ces packages et en utilisant l'environnement de développement Eclipse.

## 4.2.2 Phase de spécification

Ce module de l'atelier s'appelle *ForMAAD Requirement Specification*. L'interface principale (la figure 4.4) se compose de quatre parties :

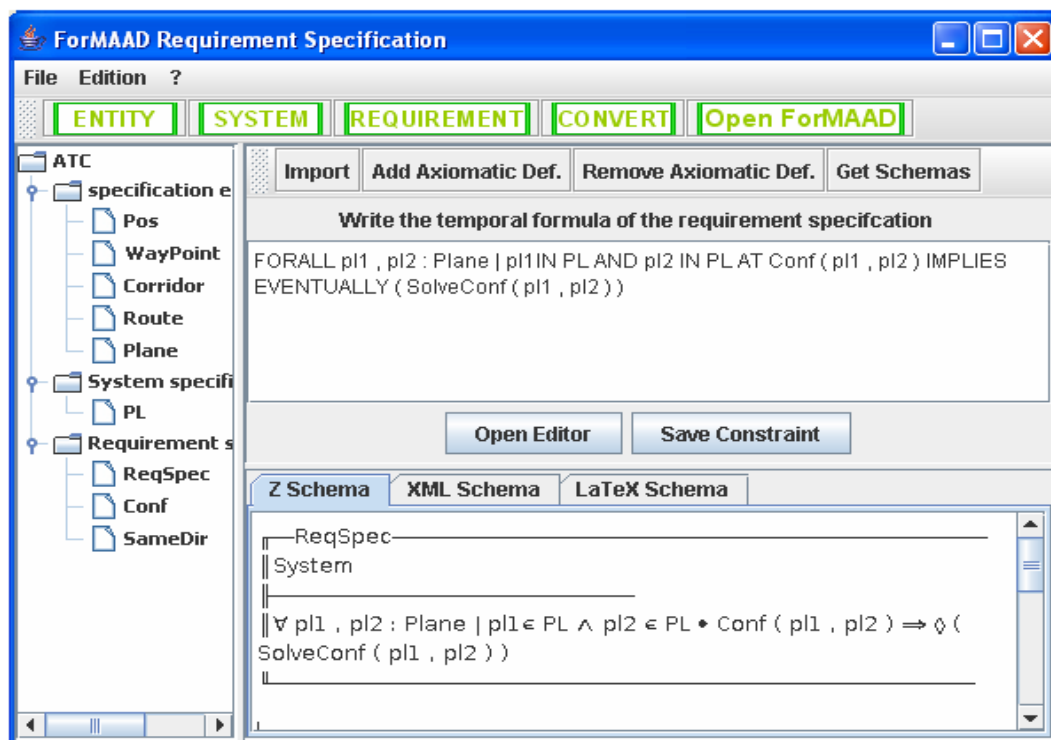


FIG. 4.4 – L'interface principale de la spécification des besoins

1. Un panneau principal : il permet d'effectuer les différentes saisies. Ce panneau permet la gestion des spécifications Z des entités (le panneau *JPEntity*), du système (le panneau *JPSystem*) et des besoins (le panneau *JPGraph*).

2. Une zone de texte : elle est découpée en trois onglets : Z schema, XML schema et LaTeX schema. Chacun d'eux contient la codification des données respectivement en Z, XML et LaTeX.
3. Une zone d'arborescence : elle contient une forme arborescente des entités créées dans la phase de spécification des entités, des objets créés dans la phase de spécification du système ainsi que les fonctions axiomatiques et le schéma *ReqSpec* de la phase de spécification des besoins.
4. Une barre de menus : elle permet de changer les panneaux de l'application, de réaliser la conversion des données dans les différents formats et de lancer le module *F<sub>or</sub>MAAD Tools* qui correspond à la phase suivante (la phase de conception).

Ce module a pour entrée des spécifications Z en format XML. Par contre, il permet de générer trois formats de fichiers différents : le format Z qui est le format indispensable, le format XML en codifiant les différents symboles de Z et le format LaTeX afin d'assurer le lien avec le module suivant qui demande pour entrée des spécifications en LaTeX ainsi que pour assurer l'échange avec l'outil Z-EVES pour la vérification.

### 4.2.3 Phase de conception

Les étapes présentées dans le cadre de la phase de conception de la méthode *F<sub>or</sub>MAAD* sont implémentées et intégrées dans le module *F<sub>or</sub>MAAD Tools* de l'atelier d'aide à la conception qui permet de guider et d'assister l'utilisateur dans le développement des applications à base d'agents.

Ce module prend en entrée une spécification Z sous forme d'un fichier XML ou LaTeX. Il permet, par la suite, de paramétrer chaque étape de raffinement et de générer la spécification résultante. Cette génération peut être automatique (le cas des étapes 2, 3, 5 et 6) ou semi-automatique nécessitant l'intervention de l'utilisateur (le cas des étapes 1, 4 et 7). Les résultats des différentes étapes ainsi que la spécification finale sont stockés en format XML et peuvent être convertis en format LaTeX. Ceci permettra l'utilisation des résultats par d'autres outils ou éditeurs. Cependant, nous

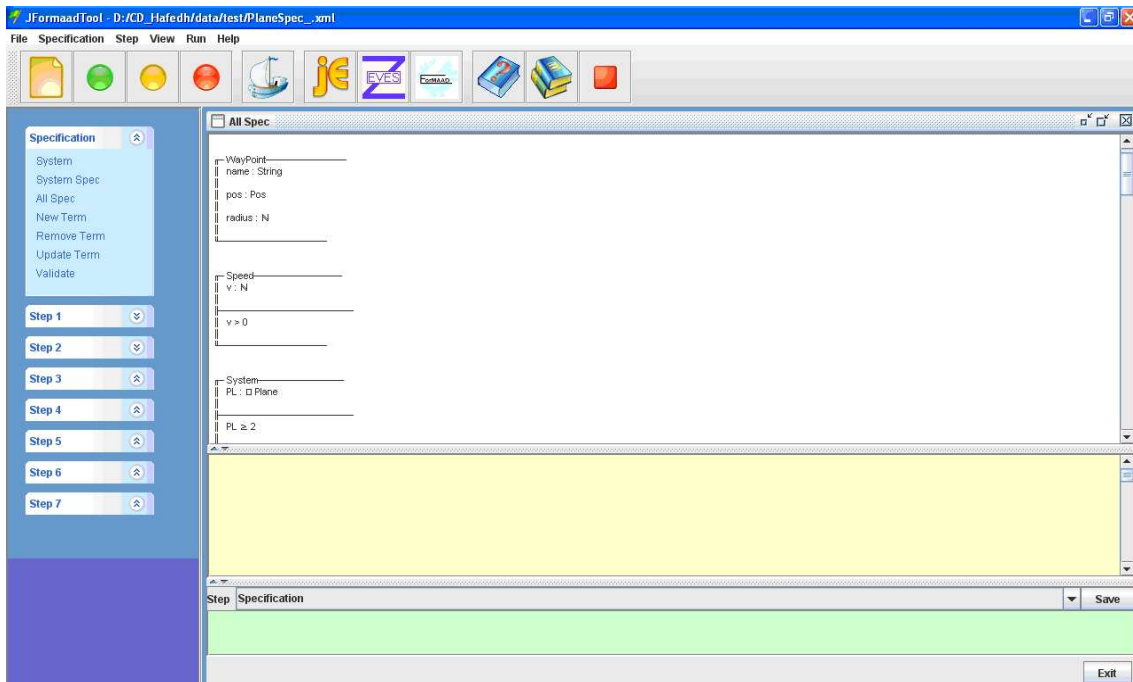


FIG. 4.5 – La première interface de  $\mathcal{F}_{or}MAADTools$

avons choisis d'afficher les spécifications sous forme de schémas Z. L'utilisateur peut enrichir et commenter les différents schémas des différentes étapes.

Pour les étapes nécessitant la génération de certains graphes tels que le graphe de précedence ou le graphe et/ou,  $\mathcal{F}_{or}MAADTools$  garantit automatiquement cette génération (les figures 4.6 et 4.7).

En plus, l'outil garantit la vérification des différents raffinements en échangeant les spécifications utilisant l'outil Z-EVES en vue d'exécuter les preuves nécessaires de théorèmes.

### 4.3 Conclusion

Dans ce chapitre, nous avons présenté notre méthode formelle  $\mathcal{F}_{or}MAAD$  de spécification et vérification des applications à base d'agents. Notre principale contribution, qui distingue  $\mathcal{F}_{or}MAAD$  des autres méthodes formelles telles que RIO et DESIRE, consiste à définir une succession d'étapes se basant sur une relation

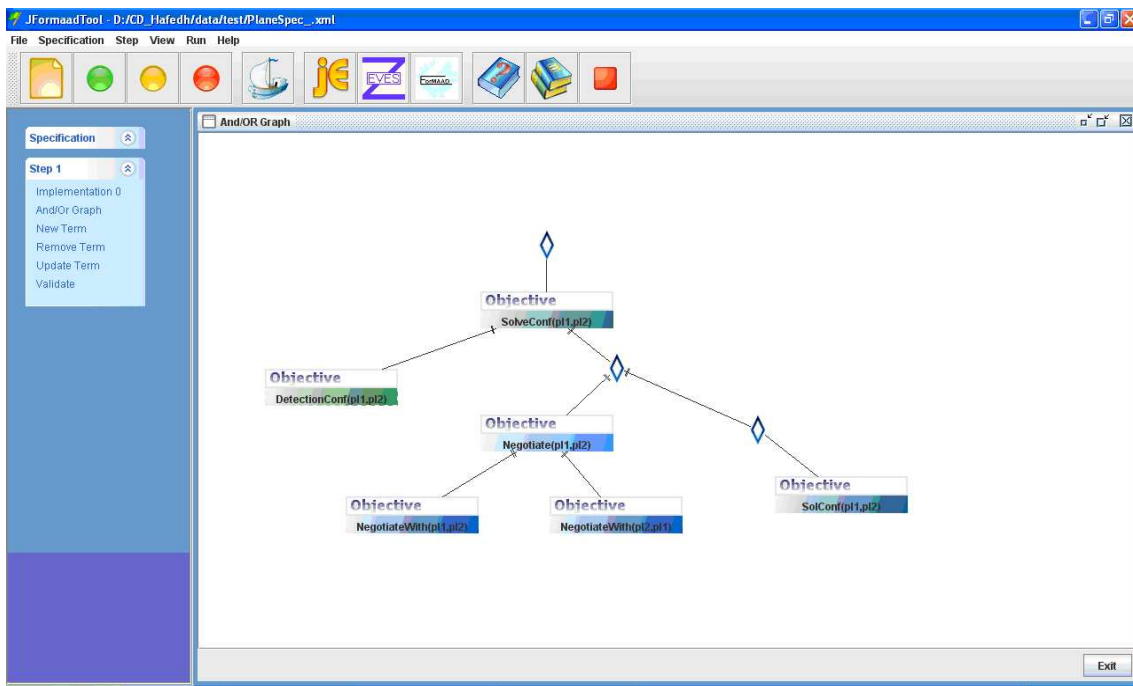


FIG. 4.6 – Un exemple de la génération automatique du graphe et/ou

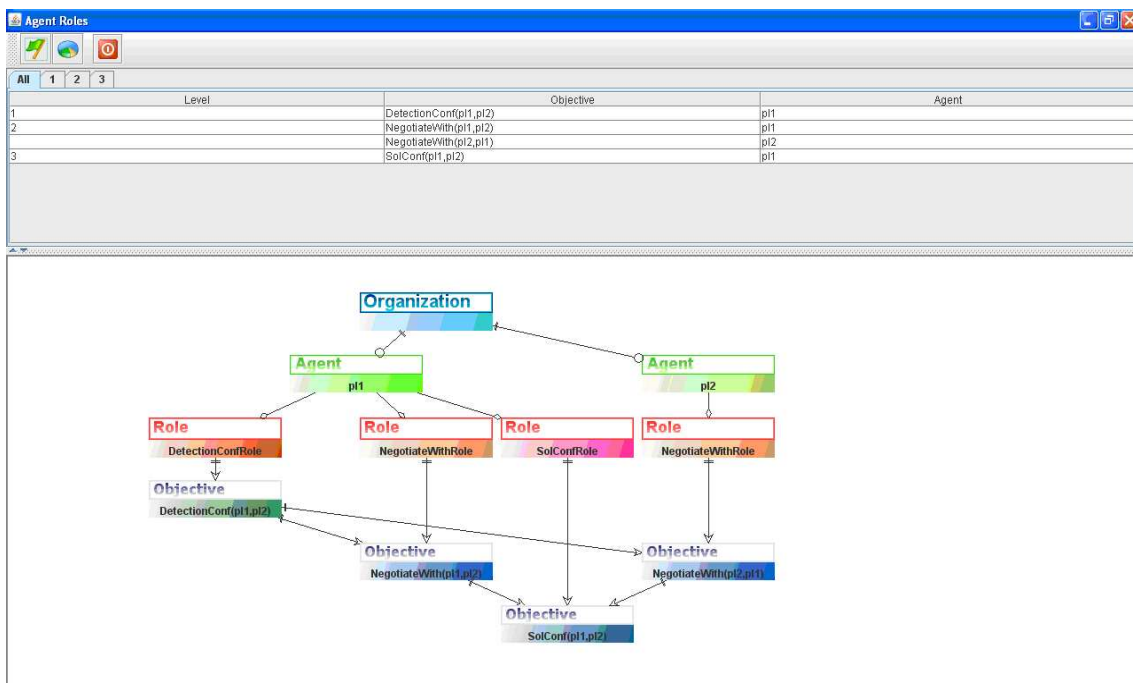


FIG. 4.7 – Un exemple de la proposition d'un scénario d'affectation des rôles

de raffinement associée à un processus de vérification. Ces étapes développent, à partir d'une spécification abstraite des besoins, les comportements individuels des différentes entités à implémenter garantissant la réalisation de l'objectif commun spécifié initialement.

La phase de spécification et les sept étapes de la phase de conception présentées dans le cadre de la méthode de conception  $\mathcal{F}_{or}\mathcal{MAAD}$  sont implémentées et intégrées dans un atelier d'aide à la conception qui permet de guider et d'assister l'utilisateur dans les différents raffinements. Cet atelier prend en entrée une description initiale des entités, du système et de l'état à atteindre. Le premier module de l'atelier, qui correspond à la phase de spécification, génère automatiquement une spécification sous forme de schémas  $Z$  de la description initiale. Ces schémas représentent l'entrée du deuxième module de l'atelier, qui correspond à la phase de conception. Ce module permet de paramétrer chaque étape de raffinement et de générer, de manière partiellement automatique, la spécification résultante de cette étape. Chaque étape est enchaînée par l'étape suivante selon la relation de raffinement sujet d'une preuve garantie en faisant recours à l'outil  $Z$ -EVES.

Le chapitre suivant consiste à valider la méthode proposée ainsi que l'atelier développé en procédant à spécifier et concevoir trois études de cas de divers domaines d'application des SMA et montrant divers aspects et diverses formes d'interaction entre les agents.



# Chapitre 5

## Études de cas

En vue de valider la méthode  $\mathcal{F}_{or}\mathcal{MAAD}$  présentée dans le chapitre précédent, trois études de cas ont été élaborées. Dans une première illustration, nous avons choisi d'aborder un cas très utilisé en SMA à savoir le problème de jeu de poursuite (proie/prédateurs) où nous avons insisté sur la description de l'aspect coopération entre les prédateurs en vue de capturer la proie.

Contrairement à la première étude de cas, la deuxième est plus concrète et plus proche de la réalité à savoir le contrôle du trafic aérien. Cette illustration est présentée dans la section 5.2 où nous avons choisi de décrire les étapes qui mettent en évidence le processus de négociation caractérisant cette étude de cas.

Finalement, nous avons pensé à présenter une autre étude de cas plus complexe et faisant intervenir un nombre d'agents plus important à savoir la gestion coopérative de pannes dans un réseau (la section 5.3) qui, en plus de l'aspect coopération, met en évidence les aspects organisation et coordination d'action permettant aux agents gestionnaires d'assurer la réparation effective de toute panne.

### 5.1 Le jeu de poursuite

Une première illustration de notre méthode  $\mathcal{F}_{or}\mathcal{MAAD}$  est la spécification d'une solution multi-agent pour le problème de poursuite [Bou92]. Cette application inclut une proie et quatre prédateurs (la figure 5.1). La proie se déplace, aléatoirement, dans

une grille sans aucune perception de l'environnement. Les prédateurs coopèrent pour capturer la proie utilisant leur perception et leur capacité de communication.

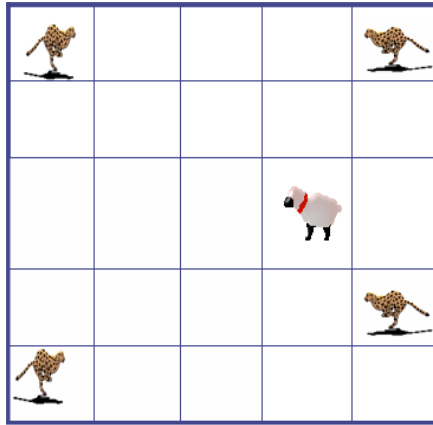


FIG. 5.1 – Le jeu de poursuite

Initialement, chaque prédateur se déplace indépendamment des autres. Dès qu'un prédateur perçoit la proie, il la suit jusqu'à sa capture du côté le plus proche. Ce prédateur, jouant le rôle de *superviseur*, va régulièrement informer les autres prédateurs sur la position courante de la proie. A partir de ce moment, chaque prédateur essaye de capturer la proie du côté approprié en se basant, essentiellement, sur l'échange d'information.

### 5.1.1 Phase de spécification

Une case de la grille sur la quelle se déplacent la proie et les prédateurs se caractérise par son abscisse et son ordonnée.

<i>Case</i>
$abs : \mathbb{N}$
$ord : \mathbb{N}$
$abs \neq 0$
$ord \neq 0$

Le concept de base est l'entité "*Entity*" caractérisée par la case qu'elle occupe qui représente sa *position* dans la grille. Une entité est capable de se déplacer aléatoirement sur la grille en effectuant, à chaque instant, au plus un pas d'un côté (gauche, droite, haut ou bas) mais pas sur la diagonale.

<i>Entity</i>
<i>pos</i> : <i>Case</i>
$x, y : \mathbb{N} \mid x = pos.abs \wedge y = pos.ord \bullet \exists i, j \in \{0, 1, -1\} \bullet$ $(i = 0 \vee j = 0) \wedge \circ (pos.abs = x + i \wedge pos.ord = y + j)$

Une proie *Prey* est une simple entité ; alors qu'un prédateur *Predator* est doté, aussi, d'une capacité de perception.

<i>Predator</i>
<i>Entity</i>
<i>Perception</i> : $\mathbb{P} \textit{Entity}$
$Perception = \{e : \textit{Entity} \mid (pos.abs - 2 \leq e.pos.abs \leq pos.abs + 2) \wedge$ $(pos.ord - 2 \leq e.pos.ord \leq pos.ord + 2) \}$

La partie prédicative insiste sur le fait que l'attribut *Perception* regroupe les entités dont les positions se trouvent dans le champ de perception qui se propage sur deux cases autour du prédateur.

Ainsi, un environnement (*System*) renferme une grille ayant des dimensions fixes, une proie *prey* de type *Entity* et quatre prédateurs de type *Predator*. Dans cet environnement, certaines conditions doivent être vérifiées telles que le fait que la position de toute entité (proie ou prédateur) ne doit pas dépasser les limites de la grille. Aussi, nous ne pouvons jamais avoir deux entités qui se positionnent dans une même case de la grille.

<i>System</i>
$X : \mathbb{N}$
$Y : \mathbb{N}$
$prey : Entity$
$pr_1, pr_2, pr_3, pr_4 : Predator$
$X > 3 \wedge Y > 3$
$\forall e \in \{prey, pr_1, pr_2, pr_3, pr_4\} \bullet e.pos.abs \leq X \wedge e.pos.ord \leq Y$
$\forall e_1, e_2 : Entity \bullet e_1.pos = e_2.pos \Rightarrow e_1 = e_2$

Le principal objectif de la spécification du problème de poursuite est de vérifier que les quatre prédateurs arrivent, sûrement dans le futur, à capturer la proie des quatre côtés. Ceci est donné par le schéma suivant :

<i>ReqSpec</i>
<i>System</i>
$\diamond \square Captured(pre, \{pr_1, pr_2, pr_3, pr_4\})$

### 5.1.2 Phase de conception

Vu que le jeu de proie/prédateur se caractérise par son aspect de coopération, nous avons choisi de mettre en relief la description de cet aspect et ceci en présentant particulièrement l'étape 1. Une description détaillée de cette étude de cas est présentée dans [RKJ04a].

#### Niveau collectif

- Stratégie de coopération :
- *Étape 1 : définition d'une stratégie de coopération*

Comme présenté dans la section 4.1.2, nous commençons cette étape par construire le graphe et/ou correspondant à l'objectif commun du jeu (la figure 5.2). Cet objectif a été décomposé en buts locaux et mis sous la forme normale conjonctive.

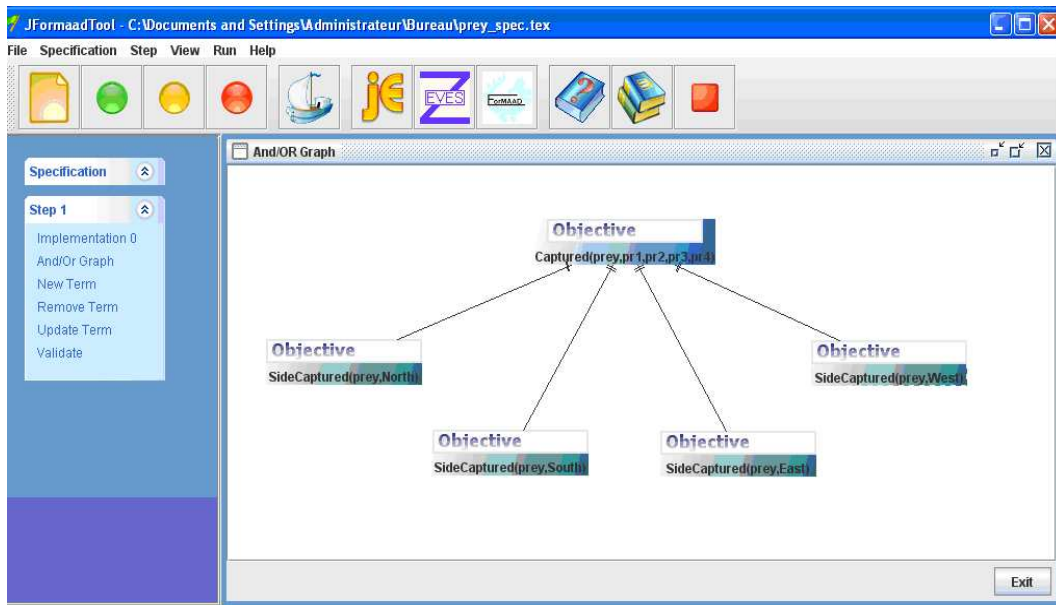


FIG. 5.2 – Le graphe et/ou généré par  $\mathcal{F}_{or}MAAD\ Tools$

Selon ce graphe, la décomposition de l'objectif commun en buts locaux permet le raffinement suivant :

$Implementation_0$
$System$
$Perceive(pre) \wedge$ $\diamond SideCaptured(pre, North) \wedge \diamond SideCaptured(pre, South) \wedge$ $\diamond SideCaptured(pre, East) \wedge \diamond SideCaptured(pre, West)$

Ce raffinement indique que pour assurer l'achèvement de l'objectif commun (capture de la proie par les prédateurs), certains buts locaux doivent être achevés tels que la perception de la proie et sa capture de chaque côté.

Ainsi, l'ensemble de buts locaux est :

$$\mathcal{L} = \{Perceive(pre), SideCaptured(pre, North), SideCaptured(pre, South), SideCaptured(pre, East), SideCaptured(pre, West)\}$$

Étant donnés ces buts locaux, l'achèvement de l'objectif commun consiste à répartir ces buts aux agents nécessaires et de garantir la réalisation de chacun.

### Niveau individuel

L'application des différentes étapes de la méthode  $\mathcal{F}_{or}\mathcal{MAAD}$  a abouti à la spécification  $Implementation_6$  qui décrit l'aspect structurel et l'aspect comportemental du système :

$$\begin{array}{l}
 \text{Implementation}_6 \\
 \hline
 \text{Implementation}_5 \\
 \hline
 \forall pr \in \{pr_2, pr_3, pr_4\} \bullet \exists s : Side \bullet \\
 \quad receive(pr_1, pr, Pos\ pr.state.pos) \Rightarrow \diamond send(pr_1, pr, Assign(pre y, s)) \\
 \\
 \forall pr_1 : Predator \bullet \forall prey : Entity \bullet \exists side : Side \bullet \\
 \quad receive(pr_1, Assign(pre y, side)) \Rightarrow \circ updateGoal(side) \wedge \\
 \quad \circ updateDest(pospre y, side) \\
 \\
 \forall x : Box \mid x = state.pos \bullet \exists s : Side \bullet updateGoal(s) \Rightarrow \\
 \quad \diamond Reduce(Dist(x, dest), Dist(state.pos, dest)) \wedge \\
 \quad \diamond Reduce(Dist(x, dest), 0)
 \end{array}$$

Cette spécification finale exprime le comportement détaillé de chaque prédateur qui consiste à recevoir le côté de capture de la proie, mettre à jour la destination et chercher à réduire la distance en vue d'atteindre cette destination.

La vérification de la satisfaction de l'objectif, à savoir la capture de la proie par les quatre prédateurs, nécessite la preuve d'une séquence de théorèmes dans le cadre des étapes de la méthode  $\mathcal{F}_{or}\mathcal{MAAD}$ . La preuve de ces théorèmes a été accomplie à l'aide de l'outil Z-EVES [RKJ04a, RKJ05b].

Ainsi, par transitivité nous arrivons à prouver le théorème suivant :

#### theorem VerifSpec

$$Implementation_6 \Rightarrow \diamond \square Captured(pre y, (pr_1, pr_2, pr_3, pr_4))$$

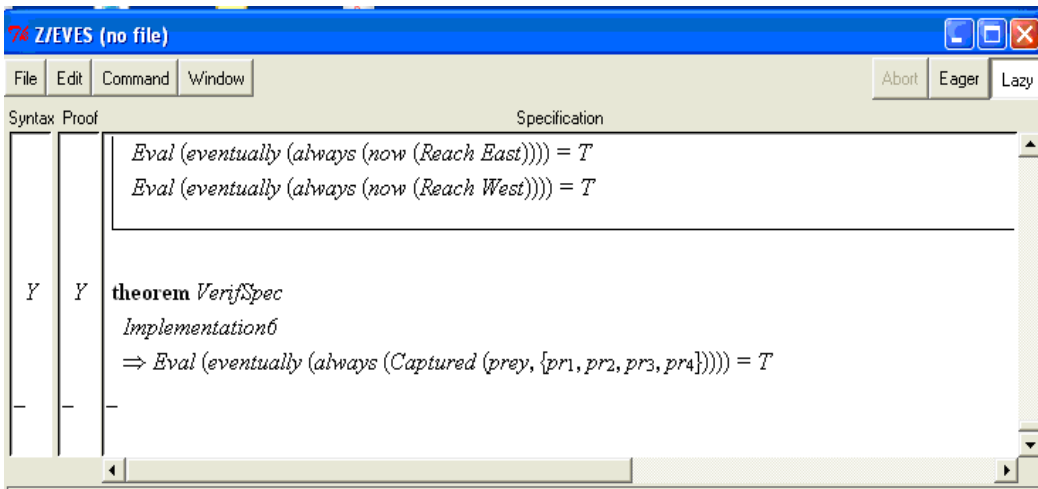


FIG. 5.3 – La preuve par réduction du théorème VerifSpec avec Z-EVES

## 5.2 Le contrôle du trafic aérien

La détection et la résolution de conflit occupent une place prépondérante dans le contrôle aérien et plus précisément dans la gestion du trafic aérien. Devant les énormes efforts qui ont été fournis pour la conception des systèmes de résolution de conflits (conflict avoidance systems [Kum99]), peu d'efforts ont été investis dans l'analyse des problèmes d'interaction de plusieurs avions [BP00, DFB02]. Bien que l'occurrence de telle situation est très faible, le problème engendré reste un des plus difficiles à résoudre. En effet, la résolution de conflit entre deux avions fait apparaître, souvent, un nouveau conflit avec un troisième avion.

Dans la littérature, plusieurs projets ont été développés afin d'assurer la gestion du trafic aérien. Parmi ces projets, certains travaux se sont intéressés à améliorer le contrôle centralisé existant [FT93]; d'autres proposent des solutions décentralisées basées sur différentes approches telles que les méthodes neuronales utilisant les algorithmes génériques [DAM00] ou les méthodes séquentielles [AGD04]. D'autres travaux sont basés sur le concept d'agent. Parmi ces travaux, nous citons ceux de

Hexmoor et Heng [HH00] basés sur le concept d'autonomie. Wangermann et Stengel [WS99] proposent une approche basée sur la coordination distribuée. Ces études ont permis de déduire que la nature d'une telle application impose une vérification formelle.

Une conception selon  $\mathcal{F}_{or}\mathcal{MAAD}$  a pu apporter une solution fiable au problème de la gestion du contrôle de trafic aérien [RKKJ06].

### 5.2.1 Phase de spécification

Dans le contrôle du trafic aérien, le concept principal est la *route* aérienne. Dans l'espace aérien, les avions doivent suivre des itinéraires bien établis désignés par des routes. Un autre concept est le secteur de contrôle (*sector*) qui correspond à une partie de l'espace aérien. Un secteur peut avoir des points de rencontre (*waypoints*) où les routes aériennes se croisent.

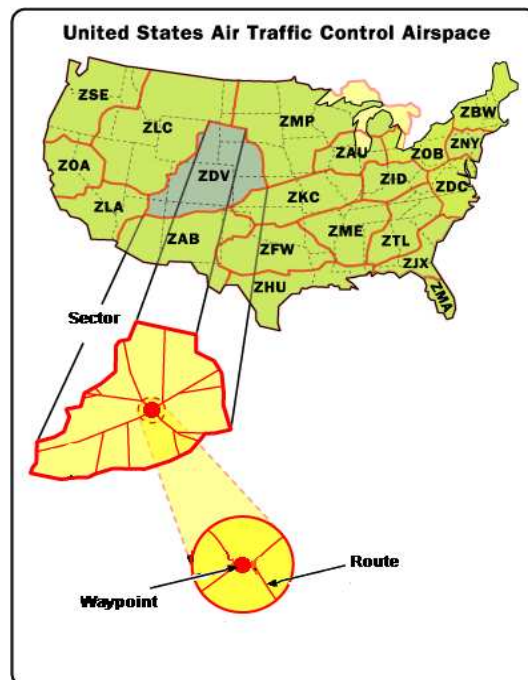


FIG. 5.4 – Le plan de routage

Nous commençons cette phase par définir l'environnement d'exécution qui correspond, dans ce cas, à l'espace aérien *sectors* (la figure 5.4). Chaque *sector* se compose

de plusieurs *routes* liées par des points appelés *waypoints*. Entre deux *waypoints*, la *route* se compose d'une séquence de couloirs parallèles (*corridors*) qui sont décrits par leur niveau de vol (*altitude*), d'un point de départ et d'un point d'arrivée. Un *corridor* peut se croiser avec un autre ayant la même *altitude*.

### Spécification des entités (avions)

Un avion (*plane*) est spécifié par un secteur (*sector*), une position (*pos*), une vitesse (*speed*) et une altitude (*alt*).

Ainsi, nous présentons une spécification initiale de l'avion :

<i>Plane</i>
<i>sector</i> : seq <i>WayPoints</i>
<i>pos</i> : <i>Pos</i>
<i>speed</i> : $\mathbb{N}$
<i>alt</i> : $\mathbb{N}$
<i>Corr</i> : <i>Corridor</i>
<i>route</i> : <i>Route</i>
<i>way</i> : seq <i>WayPoint</i>
<i>Perception</i> : $\mathbb{P}$ <i>Pos</i>
<i>PlSect</i> : $\mathbb{P}$ <i>Pos</i>
<hr/>
$alt = Corr.alt$
$\exists i : \mathbb{N} \mid 1 \leq i < \#route.SeqCorr \bullet route.SeqCorr\ i = Corr$
$\exists i : \mathbb{N} \mid 0 < i < \#way \bullet way\ i = Corr.wp_1 \wedge way\ (i + 1) = Corr.wp_2$
$Corr.wp_1.pos.x \leq pos.x \leq Corr.wp_2.pos.x$
$Corr.wp_1.pos.y \leq pos.y \leq Corr.wp_2.pos.y$
$Perception = \{p : Pos \mid pos.x \leq p.x \leq pos.x + PerMinMax$ $\wedge pos.y \leq p.y \leq pos.y + PerMinMax\}$

La partie prédicative de cette spécification indique que la position d'un avion se

trouve dans un couloir et entre deux *waypoints* adjacents. En outre, un avion est capable de percevoir tout autre avion se trouvant dans un voisinage qui s'étend sur un diamètre *PerMinMax*.

### Spécification du système

Après avoir spécifié les entités de base, nous définissons initialement un système *System* par un ensemble d'avions appelé *PL*. Cet ensemble renferme au moins deux avions. Nous rappelons que cette spécification sera enrichie, durant le processus de conception, par des détails référant aux niveaux collectif et individuel.

<i>System</i>
$PL : \mathbb{F} \textit{Plane}$
$\#PL \geq 2$

### Spécification du besoin

Cette spécification contient la description de l'objectif commun. Ce dernier consiste à résoudre toute situation conflictuelle entre deux avions. Si deux avions ont la même altitude et appartiennent à deux couloirs différents et se dirigent vers le même *waypoint*, alors ces avions se trouvent dans une situation conflictuelle. En plus, un avion peut participer à un ou plusieurs conflits. Ainsi, l'objectif commun est présenté dans la partie prédicative de la spécification suivante :

<i>ReqSpec</i>
<i>System</i>
$\forall pl_1, pl_2 \in PL \bullet \textit{Conf}(pl_1, pl_2) \Rightarrow \diamond \textit{SolveConf}(pl_1, pl_2)$

Cet objectif consiste à une éventuelle résolution de tout conflit existant entre deux avions  $pl_1$  et  $pl_2$ .

## 5.2.2 Phase de conception

Cette phase consiste à suivre les sept étapes de conception présentées dans la section 4.1.2. Notre spécification de départ est le schéma *System*. Nous le raffinons en ajoutant des détails d'implémentation selon les règles proposées dans chaque étape. La solution obtenue doit satisfaire la partie prédictive de *ReqSpec*.

### Niveau collectif

Nous considérons la conception des aspects collectifs à partir de la définition de la stratégie de coopération. Ainsi, l'étape 1 aboutit à la définition de l'ensemble de buts locaux suivants :

$$\mathcal{L} = \{DetectionConf(pl_1, pl_2), NegotiateWith(pl_1, pl_2), \\ NegotiateWith(pl_2, pl_1), SolConf(pl_1, pl_2)\}$$

Puis, dans l'étape 2, nous présentons la description de la structure d'organisation à travers la définition des rôles. Ainsi, trois rôles sont nécessaires *Detector*, *Negotiator* et *Solver* correspondant, respectivement, aux prédicats présentés dans l'ensemble  $\mathcal{L}$ . Ces rôles montrent trois relations organisationnelles (étape 3) *rorg<sub>1</sub>*, *rorg<sub>2</sub>* et *rorg<sub>3</sub>* qui les relient deux à deux. Par la suite, les rôles sont affectés aux deux avions (étape 4). Plusieurs scénarios d'affectation de rôles peuvent être définis. Dans le reste de cette section, nous allons retenir un scénario qui consiste à affecter *Detector*, *Negotiator* et *Solver* au premier avion  $pl_1$ ; et seulement *Negotiator* est affecté au deuxième avion  $pl_2$ .

L'étape 5 consiste à instancier les relations organisationnelles définies dans l'étape 3. Chaque relation faisant intervenir deux rôles doit être instanciée par au moins un lien organisationnel entre les deux avions ayant ces rôles. Ainsi, *rorg<sub>1</sub>*, *rorg<sub>2</sub>* et *rorg<sub>3</sub>* sont, respectivement, instanciées par  $ol_1$ ,  $ol_2$  et  $ol_3$ .

Les cinq étapes décrites brièvement ci-dessus ont été détaillées davantage dans [RKKJ06]. L'étape 6 consiste à décrire l'interaction entre les avions en termes de comportements collectifs .

Vu qu'une telle application se caractérise par le processus de négociation à travers des échanges de messages pour se mettre d'accord sur la solution convenable, nous proposons de présenter en détail uniquement les étapes 6 et 7. Une présentation détaillée de cette étude de cas est faite dans [KRJ07].

- Comportement collectif

- *Étape 6 : définition du comportement collectif*

A partir des liens présentés dans l'étape 5, nous déduisons les relations de causalité entre les buts locaux.

Ainsi, nous présentons *Implementation<sub>5</sub>* groupant les relations entre buts locaux. Chacun représente un comportement global :

<p style="margin: 0;"><i>Implementation<sub>5</sub></i></p> <hr style="border: 0; border-top: 1px solid black; margin: 5px 0;"/> <p style="margin: 0;"><i>System</i></p> <p style="margin: 0;"><i>R</i> : <math>\mathbb{F}</math> <i>Role</i></p> <p style="margin: 0;"><i>Rorg</i> : <math>\mathbb{F}</math> <i>OrgRelationship</i></p> <p style="margin: 0;"><i>organizationlinks</i> : <math>\mathbb{F}</math> <i>OrganizationLink</i></p> <hr style="border: 0; border-top: 1px solid black; margin: 5px 0;"/> <p style="margin: 0;"><math>\forall pl_1, pl_2 : Plane \mid pl_1 \in PL \wedge pl_2 \in PL \bullet</math></p> <p style="margin: 0; padding-left: 20px;"><i>DetectionConf</i>(<math>pl_1, pl_2</math>) <math>\Rightarrow \diamond</math></p> <p style="margin: 0; padding-left: 40px;"><math>(NegotiateWith(pl_1, pl_2) \wedge \diamond NegotiateWith(pl_2, pl_1))</math></p> <p style="margin: 0;"><math>\forall pl_1, pl_2 : Plane \mid pl_1 \in PL \wedge pl_2 \in PL \bullet</math></p> <p style="margin: 0; padding-left: 20px;"><math>(NegotiateWith(pl_1, pl_2) \wedge NegotiateWith(pl_2, pl_1))</math></p> <p style="margin: 0; padding-left: 40px;"><math>\Rightarrow \diamond SolConf(pl_1, pl_2)</math></p> <p style="margin: 0;"><math>\forall pl_1, pl_2 : Plane \mid pl_1 \in PL \wedge pl_2 \in PL \bullet</math></p> <p style="margin: 0; padding-left: 20px;"><i>DetectionConf</i>(<math>pl_1, pl_2</math>) <math>\Rightarrow \diamond SolConf(pl_1, pl_2)</math></p>
---

Le processus de négociation permet de prendre une décision sur la manière de résoudre le conflit.

Dans ce processus, une proposition est une *solution* pour la résolution du conflit qui peut être un changement d'altitude ou de vitesse :

$$Solution ::= ChangSpeed\langle\langle Speed \rangle\rangle \mid ChangAlt\langle\langle Pos \rangle\rangle$$

Dans le processus de négociation, un *Message* peut être soit une *proposition* d'une solution, une *counter proposition*, un *reject* ou une *acceptance* :

$$\begin{aligned} \text{Message} ::= & \text{Propose}\langle\langle \text{Solution} \rangle\rangle \mid \text{CounterPropose}\langle\langle \text{Solution} \rangle\rangle \\ & \mid \text{Accept}\langle\langle \text{Solution} \rangle\rangle \mid \text{Reject}\langle\langle \text{Solution} \rangle\rangle \end{aligned}$$

Chaque avion peut proposer une solution (*Propose*) pour résoudre le conflit (changer l'altitude ou changer la vitesse). Toute proposition est évaluée par l'autre. Ce dernier peut soit accepter la proposition et envoyer une acceptation (*Accept*), ou la rejeter (*Reject*). Dans le deuxième cas, l'avion rejetant la proposition doit proposer une contre proposition (*CounterPropose*) qui sera à son tour évaluée. Ces détails sont décrits dans la fonction axiomatique *NegotiateWith*. Cette dernière consiste à décrire le processus de négociation entre deux avions :

*NegotiateWith* : *Plane* × *Plane*

$$\forall pl_1, pl_2 : \text{Plane} \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \exists sol : \text{Solution} \bullet \\ \text{NegotiateWith}(pl_1, pl_2) \Rightarrow \text{Send}(pl_1, pl_2, \text{Propose}(sol))$$

$$\forall pl_1, pl_2 : \text{Plane} \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \forall sol : \text{Solution} \bullet \\ \text{Send}(pl_1, pl_2, \text{Propose}(sol)) \Rightarrow \diamond \text{Evaluate}(pl_2, pl_1, sol)$$

$$\forall pl_1, pl_2 : \text{Plane} \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \forall sol : \text{Solution} \bullet \\ \text{Evaluate}(pl_2, pl_1, sol) \Rightarrow \diamond \text{Send}(pl_2, pl_1, \text{Accept}(sol)) \vee \\ \diamond \text{Send}(pl_2, pl_1, \text{Reject}(sol))$$

$$\forall pl_1, pl_2 : \text{Plane} \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \forall sol : \text{Solution} \bullet \\ \exists sol_2 : \text{Solution} \bullet \text{Send}(pl_2, pl_1, \text{Reject}(sol)) \Rightarrow \\ \diamond \text{Send}(pl_2, pl_1, \text{CounterPropose}(sol_2))$$

$$\forall pl_1, pl_2 : \text{Plane} \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \forall sol : \text{Solution} \bullet \\ \text{Send}(pl_2, pl_1, \text{Accept}(sol)) \Rightarrow \diamond \text{sconf}(pl_1, pl_2)$$

$$\forall pl_1, pl_2 : \text{Plane} \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \forall sol : \text{Solution} \bullet \\ \text{Send}(pl_2, pl_1, \text{CounterPropose}(sol)) \Rightarrow \diamond \text{Evaluate}(pl_1, pl_2, sol)$$

## Niveau individuel

Une fois le niveau collectif est spécifié, la déduction de la description du niveau individuel sera facile par une décomposition de chaque propriété collective jusqu'à trouver les propriétés individuelles.

### - Étape 7 : définition du comportement individuel

Un comportement individuel se présente par une formule temporelle où les prédicats concernent un seul avion.

Dans cette étape, nous décrivons les comportements individuels des avions en prouvant les formules présentant les comportements collectifs. Cette preuve fait référence au processus de négociation. Pour ce faire, nous définissons une relation d'équivalence entre les actions *Send* et *Receive* précisant que chaque information émise par un avion est reçue par le destinataire :

$$S/REq : Send(pl_i, pl_j, inf) \Leftrightarrow Receive(pl_j, pl_i, inf)$$

Dans ce qui suit, nous présentons, à titre d'exemple, la preuve de l'une des formules globales décrites dans *Implementation*<sub>5</sub> ; soit :

$$DetectionConf(pl_1, pl_2) \Rightarrow \diamond SolConf(pl_1, pl_2)$$

$$\begin{aligned} & DetectionConf(pl_1, pl_2) \Rightarrow \diamond Send(pl_1, pl_2, Propose(sol)) && [IndBeh_1] \\ \Leftrightarrow & DetectionConf(pl_1, pl_2) \Rightarrow \diamond Receive(pl_2, pl_1, Propose(sol)) && [S/REq] \\ \Leftrightarrow & DetectionConf(pl_1, pl_2) \Rightarrow \diamond Evaluate(pl_2, pl_1, sol) && [IndBeh_2] \\ \Leftrightarrow & DetectionConf(pl_1, pl_2) \Rightarrow \diamond Send(pl_2, pl_1, Accept(sol)) && [IndBeh_3] \\ \Leftrightarrow & DetectionConf(pl_1, pl_2) \Rightarrow \diamond Receive(pl_1, pl_2, Accept(sol)) && [S/REq] \\ \Leftrightarrow & DetectionConf(pl_1, pl_2) \Rightarrow \diamond SolConf(pl_1, pl_2) && [IndBeh_4] \end{aligned}$$

Les formules individuelles *IndBeh*<sub>1</sub>, *IndBeh*<sub>2</sub>, *IndBeh*<sub>3</sub> et *IndBeh*<sub>4</sub> ainsi que d'autres nécessaires pour prouver la liste des formules globales sont exprimées dans la partie prédictive de la spécification finale *Implementation*<sub>6</sub> dont nous présentons une partie :

<i>Implementation<sub>6</sub></i>	
<i>System</i>	
$R : \mathbb{F} \text{ Role}$	
$Rorg : \mathbb{F} \text{ OrgRelationship}$	
$organizationlinks : \mathbb{F} \text{ OrganizationLink}$	
$\forall pl_1, pl_2 : \text{Plane} \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \exists sol : \text{Solution} \bullet$	$Perceive(pl_1, pl_2) \Rightarrow \diamond Send(pl_1, pl_2, Propose(sol))$
$\forall pl_1, pl_2 : \text{Plane} \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \exists sol : \text{Solution} \bullet$	$Receive(pl_1, pl_2, Propose(sol)) \Rightarrow$ $\diamond Evaluate(pl_1, pl_2, sol)$

Toutes les obligations de preuve ont été prouvées pour cette étude de cas à l'aide de l'outil Z-EVES [RKKJ06]. Par transitivité, le théorème suivant est, ainsi, prouvé (la figure 5.5) :

**theorem VerifSpec**

$$Implementation_6 \Rightarrow (Conf(pl_1, pl_2) \Rightarrow \diamond SolveConf(pl_1, pl_2))$$

*VerifSpec* garantit que la solution proposée satisfait les besoins initiaux.

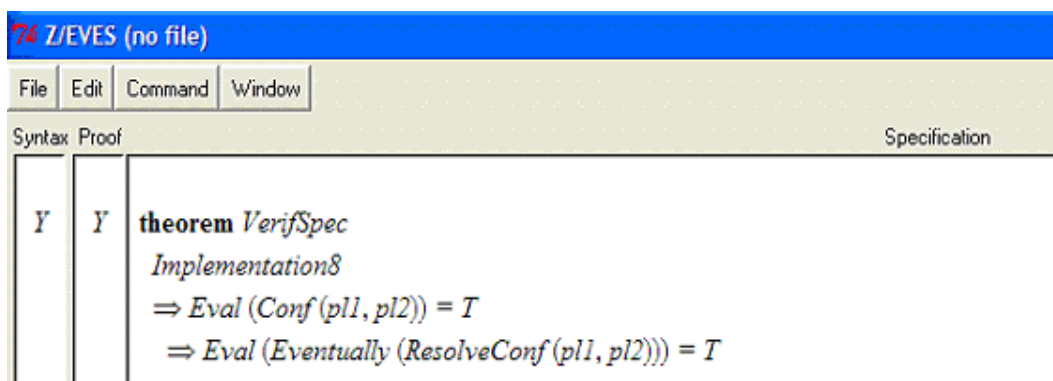


FIG. 5.5 – La preuve par réduction du théorème VerifSpec

### 5.3 La gestion coopérative de pannes dans un réseau

Contrairement aux études de cas déjà présentées, la gestion coopérative de pannes dans un réseau est une application complexe qui nécessite un nombre d'agents plus important. Aussi, elle se base essentiellement sur l'aspect coordination d'action à côté des autres aspects déjà abordés dans les études de cas précédentes à savoir la coopération et la négociation.

En outre, dans les deux premières illustrations, le nombre d'agents participants dans l'achèvement de l'objectif commun est connu d'avance (quatre prédateurs pour le problème de poursuite et deux avions pour le contrôle de trafic aérien), tandis que, pour la gestion de pannes dans un réseau, ce nombre (combien de gestionnaires vont participer pour la réparation de cette panne) n'est pas préalablement connu.

Une autre caractéristique de cette troisième étude de cas, qui la distingue des deux premières, est que nous n'avons aucune idée initiale sur la description des gestionnaires de pannes et toutes leurs propriétés statiques et comportementales seront définies durant le processus de raffinement incrémental. Par contre pour les avions et les prédateurs, nous débutons le processus de conception avec certaines propriétés statiques sur ces deux types d'entités.

Toutes ces motivations nous ont incité à choisir cette application qui représente l'un des domaines récents adoptant des techniques intelligentes pour repérer la panne et décider sur le plan de réparation adéquat en se basant sur des politiques de réparation.

Ainsi, nous désirons concevoir un SMA pour la gestion coopérative de pannes dans un réseau (Cooperative Network-Fault Management CNFM [GCS96]). Les agents interagissent afin de gérer chaque défaut dans le réseau. L'objectif de ce système est de réparer chaque défaut qui se présente. Cette réparation se fait selon un processus de plusieurs étapes.

Ce processus de gestion débute par la détection du défaut (un dysfonctionnement). Ainsi, au moins une entité (un gestionnaire) détecte un événement de panne

et il devient le gestionnaire *Access*. L'événement détecté sera sujet d'un message émis à au moins un autre gestionnaire. Ainsi l'entité *Access* doit informer les autres entités de la présente situation en diffusant l'information correspondante.

Une entité (*Recognition*) recevant cette information peut se charger de la collection d'information portant sur le présent défaut (date de détection, position, contexte, ...). A partir de ces informations et en se basant sur un historique des cas de pannes, cette entité procède à ségréguer les informations requises sur différents cas de pannes. Ces cas seront envoyés aux autres entités.

Une entité (*Diagnostic*) reçoit ces cas de pannes possibles et procède à produire des hypothèses sur les causes de ces pannes afin de décider sur la cause effective de chacune d'entre elles. Ces causes doivent être confirmées en utilisant le service d'une autre entité (*Operation*). Finalement, une seule cause est retenue (celle qui a été confirmée) en vue d'être résolue.

Travaillant comme serveur pour les entités *Recognition* et *Diagnostic*, le principe de l'entité *Operation* est de déterminer les opérations demandées dans les différentes phases du processus de gestion et d'ordonner leurs exécutions à travers les services de l'entité *Access*. Ainsi, l'entité *Operation*, une parmi ceux présentes dans le système, propose des services aux autres gestionnaires durant le processus de réparation. La contribution de cette entité consiste à sélectionner la stratégie de réparation à adopter et générer des plans d'actions nécessaires. Durant le processus de gestion de pannes, cette entité est chargée de la génération de plusieurs types de plans. Les types de plans les plus importants sont :

- Plan de preuve : quand l'entité *Diagnostic* décide de valider une liste d'hypothèses, elle demande à l'entité *Operation* de confirmer l'une de ces hypothèses et par la suite une seule cause sera retenue. Il est nécessaire d'effectuer une preuve pour chaque hypothèse.
- Plans de réparation : une fois le *Diagnostic* informe l'entité *Recognition* sur la cause valide du défaut, l'entité *Operation* sera chargée de générer un plan de réparation afin de corriger le dysfonctionnement. Ainsi, elle doit trouver une solution possible et les opérations qui en découlent.

- Plan de validation : ce plan nécessite des informations sur le réseau afin de vérifier si la solution est atteinte.

Une ou plusieurs entités *Reparation* exécutent les plans de réparation déjà générés par l'entité *Operation* et aboutissent aux résultats attendus éliminant toute panne dans le réseau.

### 5.3.1 Phase de spécification

Nous débutons cette phase par la définition de l'environnement d'exécution qui correspond, dans ce cas, au système CNFM. Ce système est composé de plusieurs entités coopérant pour gérer tout défaut dans le réseau.

#### Spécification des entités

Dans cette étude de cas, le processus de conception commence sans aucune information sur les entités participantes. Ainsi, une entité *Entity* est spécifiée par ses comportements dans le processus coopératif. Les comportements seront spécifiés durant le processus de raffinement.

<i>Entity</i> <i>name : String</i>
---------------------------------------

#### Spécification du système

Initialement, nous définissons un système (*System*) comme un ensemble d'entités appelées *Managers*. Cet ensemble contient au moins deux entités. Aussi, un système est, initialement, caractérisé par un historique des cas de pannes rencontrés (*cases*) ainsi que leurs causes (*causes*) et les différentes stratégies de réparation convenables (*polices*). Nous rappelons que cette spécification du système sera enrichie, durant le processus de conception, par les détails relatifs aux deux niveaux collectif et individuel.

<i>System</i>
<i>Managers</i> : $\mathbb{F}$ <i>Entity</i>
<i>cases</i> : $\mathbb{F}$ <i>Cases</i>
<i>causes</i> : $\mathbb{F}$ <i>Causes</i>
<i>policies</i> : $\mathbb{F}$ <i>Policy</i>
$\#Managers \geq 2$

### Spécification du besoin

La spécification du besoin contient la description de l'objectif commun. Ce dernier, dans le contexte de ce cas, consiste à résoudre chaque situation potentielle de panne dans un réseau : s'il existe un signe de détection de panne, les entités responsables coopèrent pour éliminer les causes de cette panne et retrouver la situation normale. Ainsi, l'objectif commun est présenté dans la partie prédicative de la spécification des besoins suivantes :

<i>ReqSpec</i>
<i>System</i>
$\forall manager \in Managers \bullet \forall events : Events \bullet$ $DetectFailure(manager, events) \Rightarrow \diamond ResolveFailure(Managers, events)$

Le prédicat *DetectFailure* décrit une situation de défaut qui se manifeste par le fait qu'une entité détecte un événement indiquant une situation de dysfonctionnement. Dans ce qui suit, nous n'allons pas nous intéresser aux types d'événements et comment les détecter mais, plutôt, il suffit de signaler la présence d'une situation de dysfonctionnement. Étant donné cette situation, le système va converger vers une situation normale une fois le défaut est réparé.

Le prédicat *ResolveFailure* décrit la réparation de défaut assurée par un ensemble de gestionnaires.

### 5.3.2 Phase de conception

Dans cette section, nous procédons à décrire, pour cette étude de cas, les étapes de spécification présentées lors de la présentation de la méthode  $\mathcal{F}_{or}MAAD$ . La première spécification est celle du système (le schéma *System*). Ce schéma sera raffiné par l'ajout des différents détails d'implémentation selon les règles de chaque étape. La spécification finale doit satisfaire la partie prédictive de *ReqSpec*.

#### Niveau collectif

Dans ce niveau, nous considérons la description des aspects collectifs partant de la définition de la stratégie de coopération. Ensuite, nous décrivons la structure organisationnelle présentant les interactions en termes de comportements globaux.

• Stratégie de coopération :

- *Étape 1 : définition de la stratégie de coopération*

Comme il est déjà présenté, nous commençons cette étape par construire le graphe et/ou correspondant à l'objectif commun défini (la figure 5.6).

Ce graphe est basé sur un ensemble de fonctions axiomatiques permettant la décomposition des formules temporelles décrivant l'objectif commun :

$$\begin{array}{|l} \hline \textit{ResolveFailure} : \textit{Managers} \times \textit{Events} \\ \hline \textit{ResolveFailure}(\textit{Managers}, \textit{events}) \Leftrightarrow \textit{FindCases}(\textit{events}, \textit{cases}) \wedge \\ \quad \diamond (\textit{FindCause}(\textit{cases}, \textit{cause}) \wedge \diamond \textit{Repair}(\textit{cause})) \end{array}$$

$$\begin{array}{|l} \hline \textit{FindCases} : \textit{Managers} \times \textit{Events} \times \textit{Cases} \\ \hline \textit{FindCases}(\textit{events}, \textit{cases}) \Leftrightarrow \textit{CollecteInf}(\textit{events}) \wedge \\ \quad \diamond (\textit{Segregate}(\textit{events}, \textit{cases})) \end{array}$$

$$\begin{array}{|l} \hline \textit{FindCause} : \textit{Managers} \times \textit{Cases} \times \textit{Causes} \\ \hline \textit{FindCause}(\textit{cases}, \textit{causes}) \Leftrightarrow \textit{ProposeHypo}(\textit{cases}, \textit{causes}) \wedge \\ \quad \diamond (\textit{Confirm}(\textit{cause})) \end{array}$$

$$\text{Repair} : \text{Managers} \times \text{Causes}$$

$$\text{Repair}(\text{cause}) \Leftrightarrow \text{GeneratePlan}(\text{cause}, \text{plan}) \wedge \\ \diamond (\text{ExecutePlan}(\text{plan}))$$

Où les prédicats *CollecteInf*, *Segregate*, *ProposeHypo*, *Confirm*, *GeneratePlan* et *ExecutePlan* sont des actions élémentaires détaillant, respectivement, la collecte d'informations à propos de la panne, la ségrégation de ces informations en cas de panne, la proposition des hypothèses sur les causes de la panne, la confirmation d'une cause effective, la génération du plan adéquat pour la réparation ainsi que l'exécution effective de ce plan.

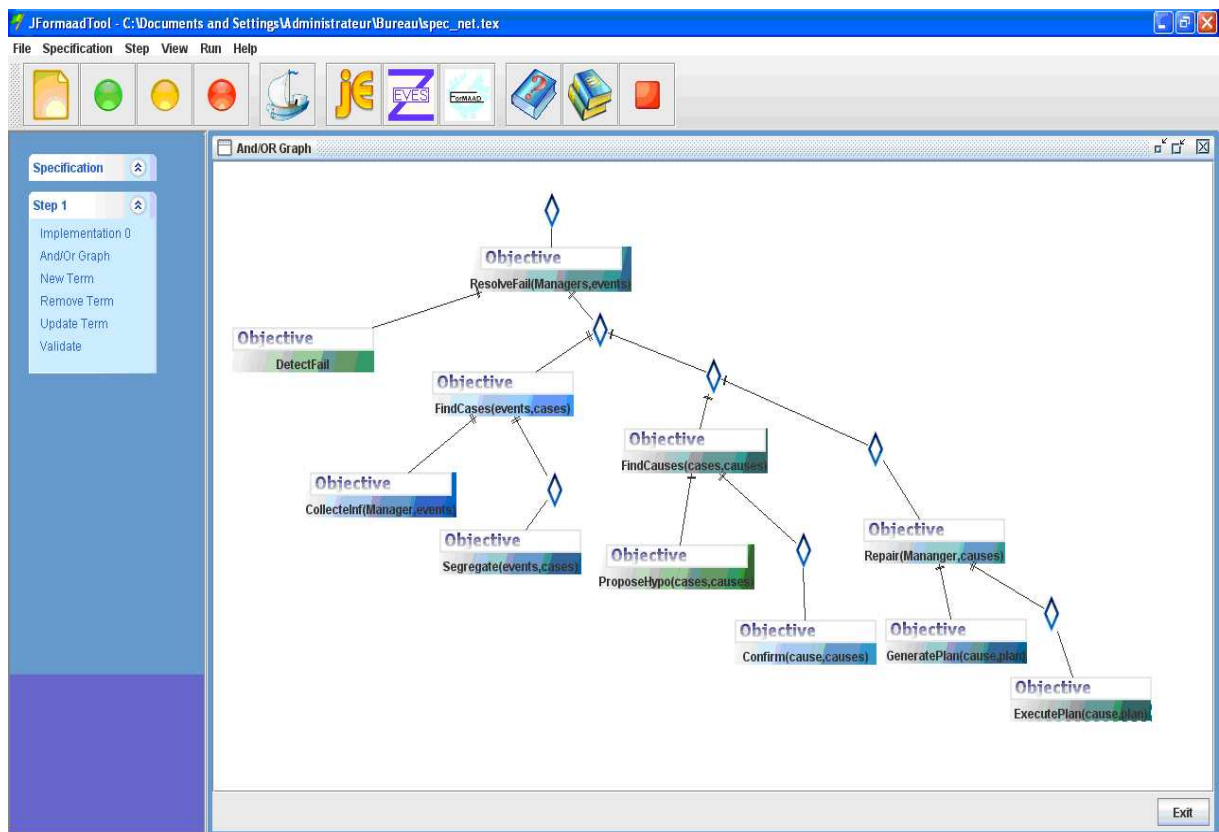


FIG. 5.6 – Le graphe et/ou de l'application CNFM généré par l'outil  $\mathcal{F}_{or}MAAD$  Tools

Selon le graphe et/ou (la figure 5.6), la décomposition de l'objectif commun en buts locaux aboutit au raffinement suivant :

<i>Implementation</i> <sub>0</sub>
<i>System</i>
$\forall events : Events \bullet \exists cases : Cases \bullet \exists cause : Causes \bullet$ $\exists plan : seq\ RepairAct \bullet DetectFailure(events) \Rightarrow$ $\diamond ((CollecteInf(events) \wedge \diamond (Segregate(events, cases))) \wedge$ $\diamond ((ProposeHypo(cases, causes) \wedge \diamond (Confirm(cause))) \wedge$ $\diamond (GeneratePlan(cause, plan) \wedge \diamond (ExecutePlan(plan))))$

Ainsi, l'ensemble de buts locaux est :

$$\mathcal{L} = \{DetectFailure(events), CollecteInf(events), \\ Segregate(events, cases), ProposeHypo(cases, cause), \\ Confirm(cause), GeneratePlan(cause, plan), \\ ExecutePlan(plan)\}$$

▪ Structure organisationnelle :

Après avoir défini la liste des buts locaux, nous procédons, dans les quatre étapes qui suivent respectivement, à (1) identifier les rôles nécessaires, (2) trouver les relations organisationnelles entre ces rôles, (3) assigner les rôles définis à un ensemble d'agents et, finalement, (4) déduire l'ensemble de liens organisationnels.

- *Étape 2 : identification des rôles*

Partant de l'ensemble  $\mathcal{L}$  de buts locaux, nous dénotons sept noms de prédicats à savoir *DetectFailure*, *CollecteInf*, *Segregate*, *ProposeHypo*, *Confirm*, *GeneratePlan* et *ExecutePlan*. Ainsi, sept rôles apparaissent à savoir *Access*, *Recognition*, *Segregation*, *Diagnostic*, *Confirmation*, *Operation* et *Rreparation* correspondant, respectivement, aux prédicats présentés.

*Implementation*<sub>1</sub> raffinant *Implementation*<sub>0</sub> et décrivant ces rôles se présente comme suit :

<i>Implementation<sub>1</sub></i>
<i>System</i>
$R : \mathbb{F} \text{ Role}$
$Access.goals = \{DetectFailure(events)\} \wedge$
$Recognition.goals = \{CollecteInf(events)\} \wedge$
$Segregation.goals = \{Segregate(events, cases)\} \wedge$
$Diagnostic.goals = \{ProposeHypo(cases, causes)\} \wedge$
$Confirmation.goals = \{Confirm(cause)\} \wedge$
$Operation.goals = \{GeneratePlan(cause, plan)\} \wedge$
$Reparation.goals = \{ExecutePlan(plan)\}$

- *Étape 3 : définition des relations organisationnelles*

Le raffinement résultant introduit le concept *Rorg* :

<i>Implementation<sub>2</sub></i>
<i>System</i>
$R : \mathbb{F} \text{ Role}$
$Rorg : \mathbb{F} \text{ OrgRelationship}$
$Rorg = \{rorg_1, rorg_2, rorg_3, rorg_4, rorg_5, rorg_6, rorg_7\}$
$rorg_1.participants = \{Access, Recognition\}$
$rorg_2.participants = \{Recognition, Segregation\}$
$rorg_3.participants = \{Access, Segregation\}$
$rorg_4.participants = \{Segregation, Diagnostic\}$
$rorg_5.participants = \{Diagnostic, Confirmation\}$
$rorg_6.participants = \{Confirmation, Operation\}$
$rorg_7.participants = \{Operation, Reparation\}$

Étant donné un ensemble de buts locaux associé à chaque rôle, dans cette étape, nous mettons en évidence les paramètres communs entre les buts locaux de chaque

deux rôles distincts. A titre d'exemple, une première relation organisationnelle  $rorg_1$  concerne *Access* et *Recognition* dû à la présence du paramètre commun *events*. D'une manière similaire, nous définissons les autres relations à savoir  $rorg_2$  entre *Recognition* et *Segregation*,  $rorg_3$  concernant *Access* et *Segregation*,  $rorg_4$  entre *Segregation* et *Diagnostic*,  $rorg_5$  entre *Diagnostic* et *Confirmation*,  $rorg_6$  entre *Confirmation* et *Operation* et, finalement  $rorg_7$  entre *Operation* et *Reparation*.

- *Étape 4 : affectation des rôles*

L'affectation des rôles aux agents nécessite la spécification de l'ordre de précedence entre les différents buts locaux. Cet ordre est basé sur les opérateurs temporels utilisés pour décrire les buts locaux. De ce fait, nous développons la forme normale

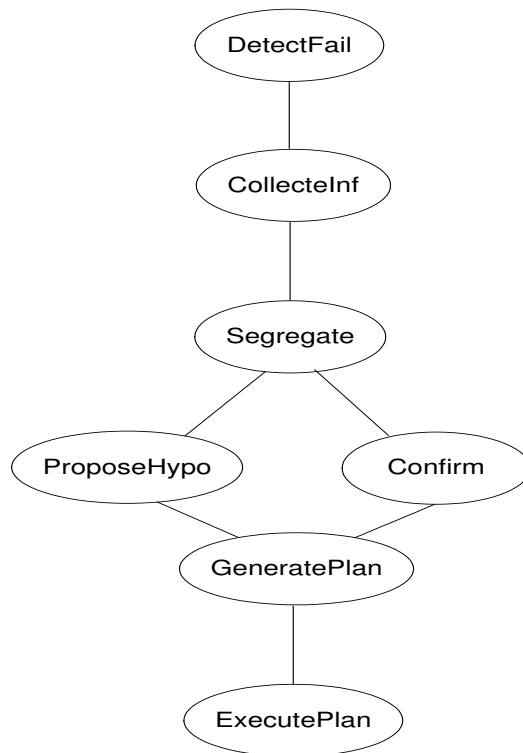


FIG. 5.7 – Le graphe de précedence des buts locaux de CNFM

disjonctive de l'objectif commun à partir de la forme conjonctive.

Cette forme normale disjonctive décrit une liste de scénarios exclusifs pour l'exécution de l'objectif commun. Dans cette étude de cas, un seul scénario d'exécution est défini (la figure 5.7).

*DetectFailure* est le premier but local qui doit être exécuté par un agent : c'est le premier qui détecte un événement exprimant un défaut. Cet agent fait partie d'un ensemble de gestionnaires *Managers* et il aura le rôle *Access*. Le deuxième but local à achever est *CollecteInf*. Ensuite, vient le but local *Segregate*. Une fois les cas de panne sont trouvés, l'opération suivante consiste à chercher les causes potentielles de cette panne et de mettre en évidence la cause effective et ceci à travers les deux buts locaux qui peuvent être exécutés en partie en parallèle à savoir *ProposeHypo* et *Confirm*. L'étape suivante consiste à générer un plan de réparation *GeneratePlan* en se basant sur une politique de réparation relative à la cause signalée. Finalement, le dernier but local à achever est *ExecutePlan* qui représente la réparation effective de la panne.

Ainsi, différents scénarios d'affectation de rôles sont possibles grâce à la domination de l'aspect séquentiel de ce processus de gestion de pannes. Dans cette section, nous allons adopter un scénario qui consiste à assigner *Access*, *Recognition*, *Diagnostic*, *GeneratePlan* et *ExecutePlan* à, respectivement, cinq agents *manager<sub>1</sub>*, *manager<sub>2</sub>*, *manager<sub>3</sub>*, *manager<sub>4</sub>* et *manager<sub>5</sub>*. Alors que le rôle *Segregation* sera de même assigné à *manager<sub>2</sub>* et le rôle *Confirmation* sera affecté au *manager<sub>4</sub>*.

*Implementation<sub>3</sub>*

*System*

$R : \mathbb{F} \text{ Role}$

$R_{org} : \mathbb{F} \text{ OrgRelationship}$

$manager_1.roles = \{Access\} \wedge$

$manager_2.roles = \{Recognition, Segregation\} \wedge$

$manager_3.roles = \{Diagnostic\} \wedge$

$manager_4.roles = \{Operation, Confirmation\} \wedge$

$manager_5.roles = \{Reparation\}$

- *Étape 5 : définition des accointances (liens organisationnels)*

Nous rappelons que cette étape consiste à instancier les relations organisationnelles définies dans l'étape 3. Chaque relation organisationnelle connectant deux rôles doit être instanciée par au moins un lien organisationnel entre les gestionnaires ayant ces

rôles. Ainsi,  $rorg_1, rorg_2, rorg_3, rorg_4, rorg_5, rorg_6$  et  $rorg_7$  sont respectivement instanciées par  $ol_1, ol_2, ol_3, ol_4, ol_5, ol_6$  et  $ol_7$ .

Par exemple,  $rorg_1$  est une relation entre *Access* et *Recognition*. Un lien organisationnel correspondant  $ol_1$  doit joindre  $manager_1$  et  $manager_2$ . De même, nous pouvons retrouver les participants dans  $ol_2, ol_3, ol_4, ol_5, ol_6$  et  $ol_7$ .

<i>Implementation</i> <sub>4</sub>
<p><i>System</i></p> <p><math>R : \mathbb{F} \text{ Role}</math></p> <p><math>Rorg : \mathbb{F} \text{ OrgRelationship}</math></p> <p><math>organizationlinks : \mathbb{F} \text{ OrganizationLink}</math></p>
<p><math>ol_1.E = \{manager_1, manager_2\} \wedge ol_3.E = \{manager_1, manager_2\} \wedge</math></p> <p><math>ol_4.E = \{manager_2, manager_3\} \wedge ol_5.E = \{manager_3, manager_4\} \wedge</math></p> <p><math>ol_7.E = \{manager_4, manager_5\}</math></p>

- Comportement collectif

- *Étape 6 : définition du comportement collectif*

A partir des liens organisationnels présentés dans l'étape précédente et les rôles correspondants, nous déduisons les relations de causalité connectant les buts locaux identifiés.

Ainsi, pour  $ol_1$  connectant  $manager_1$  (*Access*) et  $manager_2$  (*Recognition*), nous dénotons une première relation de causalité entre les buts locaux des rôles correspondants.

De ce fait, nous présentons un schéma raffiné *Implementation*<sub>5</sub> groupant la liste des relations entre les buts locaux. Chacune de ces relations représente un comportement global :

---

*Implementation<sub>5</sub>*

*System*

*plan* : seq *RepairAct*

*R* :  $\mathbb{F}$  *Role*

*Rorg* :  $\mathbb{F}$  *OrgRelationship*

*organizationlinks* :  $\mathbb{F}$  *OrganizationLink*

---

*DetectFailure*(*manager*<sub>1</sub>, *events*)  $\Rightarrow \diamond$  *CollecteInf*(*manager*<sub>2</sub>, *events*)

*CollecteInf*(*manager*<sub>2</sub>, *events*)  $\Rightarrow \diamond$  *Segregate*(*manager*<sub>2</sub>, *events*, *cases*)

*Segregate*(*manager*<sub>2</sub>, *events*, *cases*)  $\Rightarrow \diamond$  *ProposeHypo*(*manager*<sub>3</sub>, *cases*, *causes*)

*ProposeHypo*(*manager*<sub>3</sub>, *cases*, *causes*)  $\Rightarrow \diamond$  *Confirm*(*manager*<sub>4</sub>, *cause*)

*Confirm*(*manager*<sub>4</sub>, *cause*)  $\Rightarrow \diamond$  *GeneratePlan*(*manager*<sub>4</sub>, *cause*, *plan*)

*GeneratePlan*(*manager*<sub>4</sub>, *cause*, *plan*)  $\Rightarrow \diamond$  *ExecutePlan*(*manager*<sub>5</sub>, *plan*)

---

## Niveau individuel

Une fois les aspects collectifs sont spécifiés, la déduction de la description du niveau individuel devient facile par la décomposition répétitive de chaque propriété globale jusqu'à trouver des propriétés individuelles.

### - Étape 7 : définition du comportement individuel

Dans cette étape, nous décrivons les comportements individuels des gestionnaires en prouvant les comportements globaux se référant au processus de gestion de pannes. Un comportement individuel est présenté par une formule temporelle où les prédicats pertinents concernent un seul gestionnaire.

En vue de déduire ces comportements individuels à partir des comportements globaux, nous définissons une relation d'équivalence entre *send* et *receive* assurant que chaque information émise par un gestionnaire est reçue par son destinataire :

$$S/REq : send(manager_i, manager_j, msg) \Leftrightarrow receive(manager_j, manager_i, msg)$$

Dans ce qui suit, nous illustrons à titre d'exemple, comment prouver un des six formules globales décrites dans *Implementation<sub>5</sub>* ; soit :

$$DetectFailure(manager_1, events) \Rightarrow \diamond CollecteInf(manager_2, events)$$

En vue de prouver cette relation de causalité, nous devons ajouter deux formules temporelles indiquant que si un agent détecte un défaut, il informe un autre agent gestionnaire (*IndBeh<sub>1</sub>*) et que si un agent reçoit cette notification, il se met à collecter les informations portant sur la présente panne (*IndBeh<sub>2</sub>*) :

$$\begin{aligned} & DetectFailure(manager_1, events) \Rightarrow \\ & \quad \diamond send(manager_1, manager_2, Fail(events)) \quad [IndBeh_1] \\ \Leftrightarrow & DetectFailure(manager_1, events) \Rightarrow \\ & \quad \diamond receive(manager_2, manager_1, Fail(events)) \quad [S/REq] \\ \Leftrightarrow & DetectFailure(manager_1, events) \Rightarrow \\ & \quad \diamond CollecteInf(manager_2, events) \quad [IndBeh_2] \end{aligned}$$

Ainsi, la liste des formules décrivant les comportements individuels nécessaires pour ce comportement global est :

$$\begin{aligned} IndBeh_1 : & DetectFailure(manager_1, events) \Rightarrow \\ & \quad \diamond send(manager_1, manager_2, Fail(events)) \end{aligned}$$

$$\begin{aligned} IndBeh_2 : & receive(manager_2, manager_1, Fail(events)) \Rightarrow \\ & \quad \diamond CollecteInf(manager_2, events) \end{aligned}$$

Nous pouvons déduire la liste exhaustive des comportements individuels des différents gestionnaires présentée dans le dernier raffinement dont nous présentons une partie :

---

*Implementation<sub>6</sub>*

*System*

*plan* : seq *RepairAct*

*R* :  $\mathbb{F}$  *Role*

*Rorg* :  $\mathbb{F}$  *OrgRelationship*

*organizationlinks* :  $\mathbb{F}$  *OrganizationLink*

---

*DetectFailure*(*manager*<sub>1</sub>, *events*)  $\Rightarrow$

$\diamond$  *send*(*manager*<sub>1</sub>, *manager*<sub>2</sub>, *Fail*(*events*))

*receive*(*manager*<sub>2</sub>, *manager*<sub>1</sub>, *Fail*(*events*))  $\Rightarrow$

$\diamond$  *CollecteInf*(*manager*<sub>2</sub>, *events*)

*CollecteInf*(*manager*<sub>2</sub>, *events*)  $\Rightarrow$   $\diamond$  *Segregate*(*manager*<sub>2</sub>, *events*, *cases*)

*Segregate*(*manager*<sub>2</sub>, *events*, *cases*)  $\Rightarrow$

$\diamond$  *send*(*manager*<sub>2</sub>, *manager*<sub>3</sub>, *FailCases*(*events*, *cases*))

---

*Message* ::= *Fail* $\langle\langle$ *Events* $\rangle\rangle$  | *FailCases* $\langle\langle$ *Events*  $\times$  *Cases* $\rangle\rangle$   
 | *Hypothese* $\langle\langle$ *Cases*  $\times$  *Causes* $\rangle\rangle$  | *FailCause* $\langle\langle$ *Cases*  $\times$  *Cause* $\rangle\rangle$   
 | *Repair* $\langle\langle$ *Cause* $\rangle\rangle$

Le théorème *VerifSpec* garantit que la solution proposée satisfait le besoin initial. Ce théorème est, simplement, déduit par la transitivité de la relation de raffinement.

**theorem VerifSpec**

*Implementation<sub>6</sub>*  $\Rightarrow$  (*DetectFailure*(*manager*<sub>1</sub>, *events*)  $\Rightarrow$

$\diamond$  *ResolveFailure*(*Managers*, *events*))

Comme le montre la figure 5.8, ce théorème est prouvé par l'outil Z-EVES.

Toutes les preuves (théorèmes) présentés dans la section 4.1.2 ont été réalisées pour cette étude de cas utilisant l'outil Z-EVES.

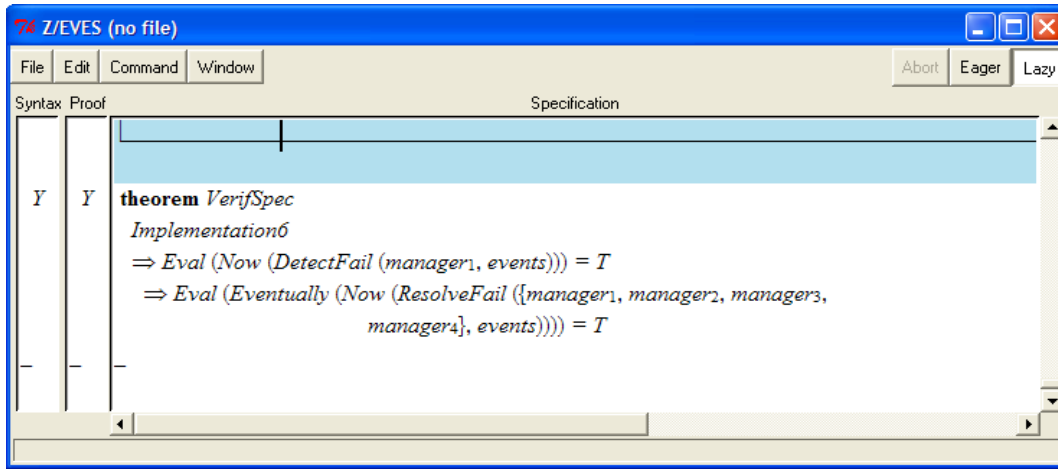


FIG. 5.8 – La preuve par réduction du théorème VerifSpec de CNFM

## 5.4 Conclusion

Ce chapitre présente une illustration de la méthode proposée par trois études de cas de divers domaines à savoir le problème de jeu de poursuite, le contrôle du trafic aérien et la gestion de pannes dans un réseau. Le premier cas est celui du jeu de poursuite (proie/prédateurs). Il s'agit d'un cas simple caractérisé essentiellement par l'aspect coopération. Par contre, les deux autres cas sont plus proches de la réalité et assez complexes abordant d'autres aspects à savoir la négociation dans le contrôle de trafic aérien et la coordination d'action ainsi que l'organisation pour la gestion de pannes dans un réseau.

Ainsi, nous avons essayé de s'assurer de la validité et de la couverture de notre méthode  $\mathcal{F}_{or}MAAD$  en l'utilisant pour concevoir certaines applications de domaines et de degrés de complexités variés tout en se servant de l'outil  $\mathcal{F}_{or}MAADTools$ . Tout résultat a été sujet d'une preuve de théorème intégrée dans la méthode et assurée en faisant recours à l'outil Z-EVES.

Suite à ces études de cas, nous sommes arrivés à vérifier les différentes spécifications résultantes et à s'assurer de leur convergence vers la satisfaction des besoins initiaux. Par ailleurs, nous avons noté le besoin d'adopter des

spécifications réutilisables de différents protocoles d'interaction qui seront spécifiées avec  $\mathcal{F}_{or}MAAD$  et intégrées dans  $\mathcal{F}_{or}MAADTools$  tels que les protocoles de négociation, de coopération, ... afin de les utiliser pour concevoir des applications si nécessaire.



# Conclusion générale

Dans cette thèse, nous avons proposé une méthode formelle pour concevoir des applications à base d'agents ( $\mathcal{F}_{or}\mathcal{MAAD}$ ). Notre contribution consiste à fournir une démarche claire permettant de couvrir les différents aspects des SMA et accompagnée d'un ensemble d'aides méthodologiques qui guident le processus de conception. Nous avons, principalement, cherché à assurer la vérification de la conception obtenue par rapport à la spécification des besoins et ceci à travers des obligations de preuves qui accompagnent chaque étape du processus de conception.

La proposition du multi-formalisme  $\mathcal{T}_{emporal}\mathcal{Z}$  qui intègre aux niveaux syntaxique et sémantique les opérateurs de la logique temporelle dans les schémas  $Z$ , nous a permis de se servir des outils supportant  $Z$ , comme  $Z$ -EVES [MS99], pour la vérification de la syntaxe et des types, aussi bien que le raisonnement au sujet de l'exactitude des étapes de raffinement en s'assurant de la satisfaction de certaines propriétés.

Les réflexions à propos du jeu de poursuite et du contrôle de trafic aérien ont permis une illustration de la méthode proposée. En effet, pour le jeu de poursuite, nous avons conçu une solution à base d'agents coopératifs, à savoir les prédateurs, qui contribuent à la capture de la proie des quatre côtés. Alors que, pour le contrôle aérien, nous avons conçu une solution décentralisée à base d'agents pour la gestion des conflits dans le cadre du trafic aérien. La solution spécifie un avion comme un agent autonome capable de détecter des conflits potentiels. La résolution efficace d'un conflit est le résultat d'un processus de négociation entre les avions.

Suite à ces deux illustrations, nous avons noté le besoin de valider la méthode avec des applications industrielles. Ainsi, nous avons proposé, dans ce sens, de développer

une conception à base d'agents pour le problème de la gestion coopérative des pannes dans les réseaux informatiques. Une telle application exige un nombre significatif d'agents montrant des comportements plus compliqués et une interaction plus forte entre les agents (les gestionnaires).

Bien que la vérification des étapes de raffinement soit totalement soutenue à l'aide de l'outil Z-EVES, nous sommes convaincus qu'il est nécessaire de mettre en œuvre un atelier qui guident l'utilisateur tout au long de ces étapes et qui peut l'aider à prendre les décisions de conception appropriées. Ainsi, nous avons développé un tel atelier, baptisé  $\mathcal{F}_{or}MAADTools$ , qui se compose d'un ensemble d'interfaces utilisateurs couplées à Z-EVES. Nous notons que les étapes 1, 2, 3 et 5 (décrites dans la section 4.1.2) peuvent être exécutées d'une manière complètement automatique ; alors que les étapes 4, 6 et 7 sont interactives et exigent l'intervention du concepteur. Nous insistons sur l'avantage que les preuves sont entièrement soutenues par Z-EVES.

Finalement, il est nécessaire de se situer par rapport aux autres travaux similaires afin de mettre en évidence l'originalité de notre travail. En fait, deux travaux sont à évoquer vu leur complétude et leur ressemblance à notre contribution à savoir RIO de Hilaire et al. [HKGM00] et DESIRE de Brazier et al [BJT98].

RIO est une méthode qui permet la spécification formelle des systèmes multi-agents fondée sur la phase d'analyse et de conception. Elle traite les niveaux collectif et individuel. De plus, elle se base sur une approche de multi-formalisme qui est développée à partir des résultats de la composition de l'object-Z et statechart. Étant donnée cette approche d'intégration de formalismes, l'approche de vérification proposée n'est pas directe mais elle est basée sur la traduction de spécifications en systèmes de transitions ce qui nécessite la définition des règles de transformation. Cette méthode présente aussi une autre limite. Il s'agit du fait qu'elle ne présente pas de démarche claire, notamment pour le passage d'une étape à une autre.

Concernant DESIRE, elle est une méthode issue de l'ingénierie de connaissances basée sur un modèle traitant la connaissance, l'interaction, la coordination des tâches et les possibilités de raisonnement dans les SMA. DESIRE propose deux modèles qui

devraient être spécifiés par le concepteur du système à savoir le modèle intra-agent et le modèle inter-agent. Aussi, une méthode de vérification compositionnelle est décrite et appliquée utilisant le système de raisonnement de la logique temporelle. Mais, le processus de développement présente les différentes étapes sans aucune guide sur le passage d'une étape à une autre.

Contrairement à ces deux travaux, notre méthode  $\mathcal{F}_{or}\mathcal{MAAD}$  propose un processus de conception incrémental accompagné d'un ensemble de guides méthodologiques assurant le passage d'une étape à une autre. Grâce au choix du langage de spécification  $Z$  et à l'intégration aussi bien syntaxique que sémantique de la logique temporelle dans  $Z$ , le processus de conception a pu être couplé avec un environnement qui supporte la vérification et le raisonnement, à savoir  $Z$ -EVES [MS99]. Aussi, ce processus est accompagné par un outil d'aide supportant la méthode proposée.

Notre contribution a porté essentiellement sur une spécification abstraite détaillée en  $\mathcal{T}_{emporal}\mathcal{Z}$  sans la génération de code correspondant. Une continuation de notre processus de développement proposé se fait par la transformation de la spécification de la conception, présentée en  $\mathcal{T}_{emporal}\mathcal{Z}$ , vers une spécification opérationnelle dans un langage d'algèbre de processus. Le langage CSP-Z [MS01] qui intègre, comme son nom l'indique, l'algèbre de processus CSP [Hoa85] et la notation  $Z$  [Spi92] semble être un candidat approprié. En effet, dans notre équipe, quelques travaux [KK07] ont porté sur l'utilisation du langage CSP-Z couplé avec son *model checker* SPIN [Hol97] employant (1) Promela pour construire les modèles de vérification ; et (2) la logique temporelle linéaire [MP92] pour la formulation des propriétés désirées.

Ainsi, une première perspective consiste à associer ces différents travaux en vue de définir toute une démarche de développement assez complète allant jusqu'à l'implémentation. Il s'agit d'automatiser les règles de passage de  $\mathcal{T}_{emporal}\mathcal{Z}$  vers CSP-Z et de raffiner cette spécification en vue de cibler une implémentation.

Partant de  $\mathcal{F}_{or}\mathcal{MAAD}$  et étant donné son aspect purement formel, une deuxième perspective consiste à décrire ces différentes étapes en se servant de l'approche orientée modèle. Ainsi, une couche semi-formelle sera définie présentant un ensemble de diagrammes UML (l'extension AML [CTCG05] de UML) décrivant les aspects

statiques et comportementaux des SMA. Dans la perspective de préserver les détails apportés par chaque étape de  $\mathcal{F}_{or}\mathcal{MAAD}$  ainsi que la sémantique des différentes formules, un ensemble de règles est à définir. Ces règles permettent de garantir qu'à partir des diagrammes UML élaborés, nous pouvons retrouver les schémas de  $\mathcal{F}_{or}\mathcal{MAAD}$  et ceci pour chacune de ses étapes de conception. Ainsi, nous profitons des avantages de  $\mathcal{F}_{or}\mathcal{MAAD}$  se rapportant à la vérification de la satisfaction de besoins tout en bénéficiant de la simplicité et la compréhensibilité des diagrammes UML.

# Bibliographie

- [AASD99] E. M. Atkins, T. F. Abdelzaher, K. G. Shin, and E. H. Durfee. Planning and resource allocation for hard real-time, fault tolerant plan execution. In *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, 1999.
- [AGD04] N. Archambaul, G. Granger, and N. Durand. Heuristiques d'ordonnement pour une résolution embarquée de conflits aériens par une méthode séquentielle. In *International Conference on computer sciences - Research, Innovation and Vision for the Future RIVF*, pages 1–6, 2004.
- [AGJ<sup>+</sup>94] J. A. Alty, D. Griffiths, N. R. Jennings, E. H. Mamdani, A. Struthers, and M. E. Wiegand. ADEPT : Advanced decision environment for process tasks : Overview and architecture. In *Proceedings of the BCS Expert Systems Conference (Applications Track, ISIP Theme)*, pages 359–371, 1994.
- [BAL05] A. Brandao, P. Alencar, and C. J. P. Lucena. AgentZ : Extending Object-Z for multi-agent systems specification. In P. Bresciani, P. Giorgini, B. Henderson-Sellers, G. Low, and M. Winikoff, editors, *Agent-Oriented Information Systems II, 6th International Bi-Conference Workshop*, volume 3508 of *Lecture Notes in Computer Science*, pages 125–139. Springer, 2005.
- [BDM97] B. Burmeister, J. Doormann, and G. Matylis. Agent-oriented traffic simulation. *Special Issue : Multi-Agent Systems Simulation*, 14(2) :79–86, 1997.

- [BG98] R. Bussow and W. Grieskamp. The Z of ZETA. The ZETA System Documentation, Technische Universitat Berlin, 1998.
- [BGPP03] C. Bernon, M. P. Gleizes, S. Peyruqueou, and G. Picard. Adelfe : A methodology for adaptive multi-agent systems engineering. volume 2577 of *Lecture Notes in Computer Science*, page 156–169, 2003.
- [BJT98] M.T. Brazier, M. Jonker, and J. Treur. Principles of compositional multi-agent system development. In *The Proceedings of the IFIP Conference IT-KNOWS98*, 1998.
- [BL00] M. Barbuceanu and W. Lo. Integrating individual, organizational and market level reasoning for agent coordination. In *European Conference on Artificial Intelligence*, pages 343–347, 2000.
- [BM96] M. Bernard and B. Mario. A scenario-based design method and an environment for the development of multiagent systems. In D. Lukose and C. Zhang, editors, *First Australian workshop on Distributed Artificial Intelligence*, LNAI Lecture Notes in Artificial Intelligence volume 1087, pages 216–231. Springer-Verlag, 1996.
- [BM04] B. Bauer and J. P. Müller. Using UML in the context of agent-oriented software engineering : State of the art. In *Agent Oriented Software Engineering, AOSE 2003*, volume 2935 of *Lecture Notes in Computer Science*, pages 1–24. Springer-Verlag, 2004.
- [BMO01] B. Bauer, J. P. Müller, and J. Odell. Agent UML : A formalism for specifying multiagent software systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(3) :207–230, 2001.
- [Bou92] T. Bouron. *Structures de Communication et d’Organisation pour la Coopération dans un Univers Multi-Agents*. PhD thesis, 1992.
- [Bow87] J. Bowen. Proceedings of Z users meeting. Technical report, Wellington Square, Oxford, 1987.
- [BP00] A. Bicchi and L. Pallottino. On optimal cooperative conflict resolution for air traffic management systems. *IEEE Transaction on Intelligent Transportation Systems*, 1(4) :221–231, December 2000.

- [BPR01] F. Bellifemine, A. Poggi, and G. Rimassa. JADE : a FIPA compliant agent development environment. In *Proceedings of the fifth international conference on Autonomous agents*. ACM Press, 2001.
- [Bru98] P. Brun. *XTL : une logique temporelle pour la spécification formelle des systèmes interactifs*. PhD thesis, 1998.
- [CCDT01] M. Coté, B. Chaib-Draa, and N. Troudi. NetSA : Une architecture multiagent réutilisable pour les environnements riches en information. *Information-Interaction-Intelligence*, (2), 2001.
- [CCG<sup>+</sup>01] G. Caire, W. Coulier, F. J. Garijo, J. Gomez, J. Pavon, F. Leal, P. Chainho, P. E. Kearney, J. Stark, R. Evans, and P. Massonet. Agent oriented analysis using message/uml. In *AOSE*, pages 119–135, 2001.
- [CG05] C. Iglesias and M. Garijo. The agent-oriented methodology MAS-CommonKADS. In *Agent Oriented Methodologies*, pages 46–78, 2005.
- [CN99] J. Collis and D. Ndumu. The ZEUS technical manual. External documentation, 1999.
- [COO] Systèmes multi-agents coopératifs. [www.irit.fr/SMAC](http://www.irit.fr/SMAC).
- [Cot99] M. Coté. Une architecture multiagent et son application aux systèmes financiers. Rapport de mastère, Université Laval, 1999.
- [CSC04] M. Cossentino, L. Sabatucci, and A. Chella. Patterns reuse in the PASSI methodology. In *Engineering Societies in the Agents World IV, 4th International Workshop*, volume 3071 of *Lecture Notes in Computer Science*, pages 294–310. Springer, 2004.
- [CT98] M. Coté and N. Troudi. Une architecture multiagent pour la recherche sur internet. *L'expertise Informatique*, (3), 1998.
- [CTCG05] R. Cervenka, I. Trencansky, M. Calisti, and D. Greenwood. AML : Agent modeling language toward industry-grade agent-based modeling. In *Agent Oriented Software Engineering, AOSE 2004 Berlin Heidelberg*, volume 3382 of *Lecture Notes in Computer Science*, page 3146. Springer-Verlag, 2005.

- [DAM00] N. Durand, J. Alliot, and F. Medioni. Neural Nets trained by genetic algorithms for collision avoidance. *Applied Artificial Intelligence*, 13(3), 2000.
- [DB01] J. Derrick and E. Boiten. *Refinement in Z and Object-Z : Foundations and Advanced Applications*. Springer Verlag, May 2001.
- [DE94] Davies and P. Edwards. Agent-K : An integration of AOP and KQML. *King's College Technical Report AUCS/TR9406*, 1994.
- [DES] D'Écision, systèmes intelligents et recherche opérationnelle. <http://www-desir.lip6.fr/>.
- [DFB02] D. Dugail, E. Feron, and K. Bilimoria. Stability of intersecting aircraft flows using heading change maneuvers for conflict avoidance. In *American Control Conference*, 2002.
- [DFWN<sup>+</sup>01] L. Cingiser DiPippo, V. Fay-Wolfe, L. Nair, E. Hodys, and O. Uvarov. A real-time multi-agent system architecture for e-commerce applications. In *Proceedings of the 5th International Symposium on Autonomous Decentralized Systems*, pages 357 – 364, 2001.
- [DG89] R. Duke and G. Smith. Temporal Logic and Z Specifications. *Australian Computer Journal*, 21(2) :62–66, 1989.
- [DHK01] R. Depke, R. Heckel, and J.M. Kuster. Roles in agent oriented modeling. *International Journal of Software Engineering and Knowledge Engineering*, 11(3) :281–302, 2001.
- [Dig03] V. Dignum. *A Model for Organizational Interaction, based on Agents, founded in Logic*. PhD thesis, 2003.
- [DL90] S. Decker and R. Lesser. A scenario for cooperative distributed problem solving. In *Proceedings of the Tenth International Workshop on Distributed AI*, Texas, 1990.
- [dL97] M. d'Inverno and M. Luck. Development and application of a formal agent framework. *ICFEM97 : First IEEE International Conference on Formal Engineering Methods*, 1997.

- [DW01] S. Deloach and M. Wood. Analysis and design using MaSE and AgentTool. In *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference MAICS 2001*, Miami University, Oxford, Ohio, 2001.
- [FCDM<sup>+</sup>99] K. Fisher, B. Chaib-Draa, J. Müller, M. Pischel, and C. Gerber. A simulation approach based on negotiation and cooperation between agents : a case study. In *IEEE Transactions on systems, man and cybernetics*, 1999.
- [Fer95] J. Ferber. *Les Systèmes multi-agents : Vers une intelligence collective*. intereditions, 1995.
- [Fer00] J. Ferber. MadKit : A generic multi-agent platform. In *Proceedings of the Fourth International Conference on Autonomous Agents*. ACM Press, 2000.
- [Fis94] M. Fisher. A survey of concurrent MetateM – The language and its applications. In *Proceedings of the 1st International Conference on Temporal Logic*, Lecture Notes in Computer Science, pages 480–505. Springer-Verlag, 1994.
- [FT93] B. M. X. Fron and J. Tumelin. Arc 2000 : Automatic radar. Technical report, Eurocontrol, 1993.
- [FT98] B. M. X. Fron and J. Tumelin. ZTC : A Type Checker for Z Notation. Users guide (version 2.03), DePaul University, 1998.
- [GCS96] M. Garijo, A. Cancer, and J. Sanchez. A multi-agent system for cooperative network-fault management. In *Proceedings of the First International Conference and Exhibition on the practical applications of intelligent agents and multi-agent technology*, pages 279– 294, London, 1996.
- [GF00] O. Gutknecht and J. Ferber. Un modèle d’analyse, de construction et d’exécution de systèmes multi-agents. In J.P. Barthes, editor, *Actes des Journées Francophones en Intelligence Artificielle Distribuée et Systèmes Multi-Agents*. Hermès, 2000.

- [Gha92] M. Ghallab. Past and future chronicles for supervision and planning. In J. P. Haton, editor, *Proceedings of the 12th International Avignon Conference*, pages 216–231, 1992.
- [GJ95] P. Gray and C. Johnson. Requirements for the next generation of user interface specification languages. *The design, specification and verification of interactive systems*, 1995.
- [Gla97] N. Glaser. The CoMoMAS methodology and environment for multi-agent system development. *Lecture Notes in Computer Science*, 1286(1), 1997.
- [GMP02] F. Giunchiglia, J. Mylopoulos, and A. Perini. The Tropos development methodology : Processes, models and diagrams. In *Proceedings of the 2002 Autonomous Agent and Multi-Agent Systems*, Bologna, Italy, 2002.
- [GP03] M. P. Gleizes and G. Picard. OpenTool, outil pour la réalisation de systèmes multi-agents adaptatifs dans le cadre de la méthode ADELFE. *Technique et Science Informatiques*, 22(4) :249–253, 2003.
- [GS97] A.J. Galloway and W.J. Stoddart. An operational semantics for ZCCS. In *First IEEE International Conference on Formal Engineering Methods*, pages 272 – 282, 1997.
- [HH00] H. Hexmoor and T. Heng. Air traffic control and Alert Agent. In *Proceedings of the International Conference on Autonomous Agents*, pages 237–238, 2000.
- [HKG00] V. Hilaire, A. Koukam, P. Gruer, and J. Muller. Formal specification and prototyping of multi-agent systems. *ESAW : Engineering Societies in the Agents World*, pages 114–127, 2000.
- [Hoa85] C.A. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.
- [Hol97] G. J. Holzman. The model checker SPIN. *IEEE Transactions on Software Engineering*, 5(23) :279–295, 1997.

- [INR] Institut national de recherche en informatique et en automatique / centre de recherche INRIA Paris-Rocquencourt. <http://www-c.inria.fr>.
- [JCd02] I. Jarras and B. Chaib-draa. *Aperçu sur les systèmes multiagents*. Série Scientifique, 2002.
- [JKR02] M. Jmaiel, A. Hadj Kacem, and A. Regayeg. An operational semantics dedicated to the coordination of cooperating agents. In *Proceedings IEEE International Conference on Systems, Man and Cybernetics*, Hammamet, Tunisia, 2002.
- [JP03] M. Jmaiel and P. Pepper. Development of communication protocols using algebraic and temporal specifications. *Computer Networks Journal*, 42 :737–764, 2003.
- [JSW98] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *International Journal of Autonomous Agents and Multi-Agent Systems*, 1(1) :7–38, 1998.
- [Kep02] J. O. Kephart. Software agents and the route to the information economy. In *Proceedings of the National Academy of Sciences*, volume 99, pages 7207–7213, 2002.
- [KG96] D. Kinny and M. Georgeff. Modelling and design of multi-agent systems. In *Intelligent Agents III : Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96)*, Budapest, Hungary, 1996. Springer-Verlag : Heidelberg, Germany.
- [KHCM02] S. Kumar, M. J. Huber, P. R. Cohen, and D. R. McGee. Toward a formalism for conversation protocols using joint intention theory. *Computational Intelligence Journal (Special Issue on Agent Communication Language)*, (2) :174–228, 2002.
- [Kin02] D. Kinny. Vip : A visual programming language for plan execution systems. In *AAMAS'02, Lecture Notes in Computer Science*, Bologna, Italy, 2002.
- [KJ99] A. Hadj Kacem and M. Jmaiel. Towards a formal definition of the cooperation among multi-agent systems. In *Proceedings of the first IFAC*

*Workshop on Multi-Agent-Systems in production*, Vienna, Austria, December, 2-4 1999.

- [KJ01] A. Hadj Kacem and M. Jmaiel. A formal negotiation model for cooperating agents. In *Proceedings of the AAAI Workshop on Negotiation Methods for Autonomous Cooperative Systems*, North Falmouth, Massachusetts, USA, November 2001.
- [KK07] A. Hadj Kacem and N. Hadj Kacem. From formal specification to model checking of MAS using CSP-Z and SPIN. *International Journal of Computing and Information Sciences - (IJCIS)*, 5(1) :35–44, April 2007. ISSN 1708-0460.
- [Klu99] M. Klusch. *Intelligent Information Agents : Agent-Based Information Discovery and Management on the Internet*. Springer, 1999.
- [Klu01] M. Klusch. Information agent technology for the internet : A survey. *Data and Knowledge Engineering*, 36(3) :337–372, March 2001.
- [KP95] K. Kouiss and H. Pierreval. Systèmes multi-agents : Directions actuelles pour les systèmes de productions. In *Congrès International de Génie Industriel de Montréal : La productivité dans un monde sans frontière*, II, pages 2029–2039, 1995.
- [KRJ07] A. Hadj Kacem, A. Regayeg, and M. Jmaiel. ForMAAD : A formal method for agent-based application design. *Journal of Web Intelligence and Agent Systems*, 5(4) :216–334, 2007.
- [KSW96] R. Kolyang, T. Santen, and B. Wolff. A structure preserving encoding of Z in Isabelle-Hol. In J. von Wright, J. Grundy, and J. Harrison, editors, *9th International Conference on Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science 1125, pages 283–298. Springer Verlag, 1996.
- [Kum99] K. Kumar. Immunized adaptive critic for an autonomous aircraft control application. *Artificial Immune Systems and their Applications*, 1999.

- [Lam94] L. Lamport. LaTeX, a document preparation system, user's guide and reference manual 2nd edition. Technical report, Addison-Wesley Publishing Company, 1994.
- [LdI01] M. Luck and M. d'Inverno. A conceptual framework for agent definition and development. *The Computer Journal*, 44(1) :1–20, 2001.
- [LHKJ04] M. Loulou, A. Hadj-Kacem, and M. Jmaiel. Formalization of cooperation in MAS : Towards a generic conceptual model. In *The IX Ibero-American Conference on Artificial Intelligence (IBERAMIA 2004)*, volume 3315, pages 43–52, Puebla, Mexico, November 2004. Springer-Verlag.
- [MBLA96] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. *The Swarm Simulation System, A Toolkit for Building Multi-Agent Simulations*. 1996.
- [MC94] B. Moulin and L. Cloutier. Collaborative work based on multiagent architectures : a methodological perspective. In *Neural Networks and Distributed Artificial Intelligence*, Prentice Hall, pages 261–296, 1994.
- [MM05] T. Miller and P. McBurney. Multi-agent system specification using TCOZ. In *German Conference on Multi-Agent System Technologies (MATES'05)*, 2005.
- [MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.
- [MS99] I. Meisels and M. Saaltink. The Z/EVES 2.0 reference manual. Technical Report TR-99-5493-03e, ORA, Canada, 1999.
- [MS01] A. Mota and A. Sampaio. Model-checking CSP-Z : Strategy, tool support and industrial application. *Science of Computer Programming*, (40) :59–96, 2001.
- [MSH02] H. Mazouzi, A. Fallah Seghrouchni, and S. Haddad. Agent communication languages : Open protocol design for complex interactions in multi-agent systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems : part 2*. ACM Press, 2002.

- [NDD04] D. M. Nguyen, A. Drogoul, and V. Duong. Conception d'un simulateur multi-agents pour la gestion du trafic aérien. In *Recherche Informatique Vietnam et Francophonie*, page 3340, 2004.
- [NFK<sup>+</sup>00] M. Nodine, J. Fowler, T. Ksiezzyk, B. Perry, M. Taylor, and A. Unruh. Active information gathering in InfoSleuth. *International Journal of Cooperative Information Systems*, 1 :3–28, 2000.
- [NPW04] T. Nipkow, L. C. Paulson, and M. Wenzel. *A Proof Assistant for Higher-Order Logic*. Springer-Verlag, 2004.
- [PAV06] J. PAVÓN. INGENIAS : Développement dirigé par modèles des systèmes multi-agents. Dossier d'habilitation à diriger des recherches, Université Pierre et Marie Curie, 2006.
- [PGSF05] J. Pavón, J. J. Gómez-Sanz, and R. Fuentes. The INGENIAS methodology and tools. In *Agent-Oriented Methodologies*, page 236276. Idea Group Publishing, 2005.
- [PH97] M. Perez and G. Herrera. Metrics of methodology : FUSION case. In *Proceedings of IEEE international conference on Systems, Man and Cybernetics*, volume 4, pages 3844–3848, 1997.
- [PW02] L. Padgham and M. Winikoff. Prometheus : A methodology for developing intelligent agent. In *Proceedings of the third International Workshop on Agent Oriented Software Engineering AAMAS*, Bologna, Italy, 2002.
- [RD00] P.M. Ricordel and Y. Demazeau. From analysis to deployment : A multi-agent platform survey. In *Engineering Societies in the Agents World, First International Workshop*, pages 93–105. Lecture Notes in Artificial Intelligence, Springer- Verlag, 2000.
- [RG98] A. S. Rao and M. Georgeff. Decision procedures of BDI logics. *Logic and Computation*, pages 293–344, 1998.
- [RHW<sup>+</sup>89] B. H. Roth, M. Hewett, R. Washington, R. Hewett, and A. Seiver. Distributing intelligence within an individual. *Distributed Artificial Intelligence*, pages 385–412, 1989.

- [RKJ04a] A. Regayeg, A. Hadj Kacem, and M. Jmaiel. Specification and design of multi-agent applications using Temporal Z. In M. Barley and N. K. Kasabov, editors, *Intelligent Agents and Multi-Agent Systems, 7th Pacific Rim International Workshop on Multi-Agents, PRIMA 2004, Auckland, New Zealand, August 8-13, 2004, Revised Selected Papers*, volume 3371 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2004.
- [RKJ04b] A. Regayeg, A. Hadj Kacem, and M. Jmaiel. Specification and verification of multi-agent applications using Temporal Z. In *2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2004), 20-24 September 2004, Beijing, China*, pages 260–266. IEEE Computer Society, 2004.
- [RKJ05a] A. Regayeg, A. Hadj Kacem, and M. Jmaiel. Towards a formal methodology for designing multi-agent applications. In *Multiagent System Technologies : Third German Conference, MATES 2005, Koblenz, Germany, September 11-13, 2005.*, volume 3550 of *Lecture Notes in Computer Science*, pages 153 – 164. Springer-Verlag, 2005.
- [RKJ05b] A. Regayeg, A. Hadj Kacem, and M. Jmaiel. Towards a formal methodology for developing multi-agent applications using Temporal Z. In *The 3rd ACS/IEEE International Conference on Computer Systems and Applications (AICCSA '05)*, Cairo, Egypt, January 2005.
- [RKKJ06] A. Regayeg, S. Kallel, A. Hadj Kacem, and M. Jmaiel. ForMAAD Method : An Experimental Design for Air Traffic Control. *International Transactions on Systems Science and Applications*, 1(4) :327–334, September 2006.
- [RN03] S. Russell and P. Norvig. *Artificial Intelligence - A Modern Approach - The Intelligent Agent Book*. 2003.
- [San00] D. Sannella. Algebraic specification and program development by stepwise refinement. In *Proceedings of the 9th International Workshop on*

- Logic-based Program Synthesis and Transformation, LOPSTR'99*, volume 1817 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2000.
- [Ser00] G. D. Serugendo. A formal development and validation methodology applied to agent-based systems. In *Agents Workshop on Infrastructure for Multi-Agent Systems*, pages 214–225, 2000.
- [SKL99] A. Sheth, V. Kashyap, and T. Lima. *Semantic Information Brokering : How Can a Multi-Agent Approach Help ?*, volume 1652 of *Book Series Lecture Notes in Computer Science*. Subject Collection Computer Science, 1999.
- [SLL02] S. Shapiro, Y. Lesperance, and H. J. Levesque. The cognitive agents specification language and verification environment for multiagent systems. In *International Conference on Autonomous Agents archive Proceedings of the first international joint conference on Autonomous agents and multiagent systems : part 1*, pages 19 – 26, Bologna, Italy, 2002.
- [SN00] W. Shen and D.H. Norrie. Using mobile agents to implement flexible network management strategies. *The Computer Communications*, 23 :708–719, 2000.
- [Spi92] M. Spivey. *The Z Notation (second edition)*. Prentice Hall International, 1992.
- [Spi00] M. Spivey. *The Fuzz Manual, 2nd Edition*. 2000.
- [SR98] P. H. Starke and S. Roch. INA : Integrated Net Analyzer, Version 2.1. Manuel d'utilisation, Université de Berlin, 1998.
- [SSG78] F. Schmidt, N. S. Sridharan, and L. Goodson. The plan recognition problem : An intersection of psychology and artificial intelligence. *Artificial Intelligence*, pages 45–83, 1978.
- [SW94] G. Schreiber and B. J. Wielinga. CML : The CommonKADS conceptual modeling language. In *Proceedings of the 8th European Knowledge Acquisition Workshop*, LNAI, pages 1–25, 1994.

- [SWdH94] G. Schreiber, B. Wielinga, and R. de Hoog. CommonKADS : A comprehensive methodology for KBS development. *IEEE Expert*, pages 28–36, 1994.
- [SY93] Shoham and Yoav. Agent-oriented programming. *Artificial Intelligence*, 1993.
- [SY94] Shoham and Yoav. Agent oriented programming : an overview of the framework and a summary of recent research. *Knowledge Representation and Reasoning Under Uncertainty*, 1994.
- [Sys01] R. Systems. An integrated toolkit for constructing intelligent software agents, AgentBuilder, user’s guide. documentation externe, 2001.
- [TB96] R. Teigen and M. Barbuceanu. The supply chain demonstrator. Report and user guide, Enterprise Integration Laboratory, University of Toronto, 1996.
- [TPS02] J. Thomas, A. Pearce, and L. Sterling. Assembling agent oriented software engineering methodologies from features. In *Proceedings of the 2002 Autonomous Agent and Multi-Agent Systems*, Bologna, Italy, 2002.
- [Tra00] E. Tranvouez. *Modélisation Multi-Agents de systèmes d’information distribués et coopératifs : une application à l’ordonnancement*. PhD thesis, 2000.
- [Tro99] N. Troudi. Un système multiagent pour les environnements riches en information. Rapport de mastère, Université Laval, 1999.
- [UMD] UMDL Technologies. Architecture : Agents and Ontologies. <http://www.si.umich.edu/UMDL/architecture.html>.
- [Wer89] E. Werner. Cooperating agents : A unified theory of communication and social structure. *Distributed Artificial Intelligence*, II :3–36, 1989.
- [WJK99] M. Wooldridge, N. R. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. In O. Etzioni, J. P. Müller, and J. M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents’99)*, pages 69–76, Seattle, WA, USA, 1999. ACM Press.

- [WJK00] M. Wooldridge, N. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents*, 3, 2000.
- [Woo00] M. J. Wooldridge. Reasoning about rational agents. *Intelligent robotics and autonomous agents series*, 2000.
- [WS99] J. P. Wangermann and R. F. Stengel. Optimization and coordination of multiagent systems. using principled negotiation. *Journal of guidance, control, and dynamics*, 22 :4350, 1999.
- [XS01] H. Xu and S. M. Shatz. Formal methods in agent-oriented design and analysis. *Intelligent Agent Software Engineering*, pages 1–5, 2001.
- [XU07] C. Xu and M. Utting. Reference manual for the CZT Eclipse Interface. Technical report, 2007.
- [ZO04] F. Zambonelli and A. Omicini. Challenges and research directions in agent-oriented software engineering. *Autonomous Agents and Multi-Agent Systems*, 9(3) :253–283, 2004.
- [ZS96] D. Zeng and K. Sycara. Coordination of multiple intelligent software agents. *International Journal of Cooperative Information Systems*, (5) :181–212, 1996.