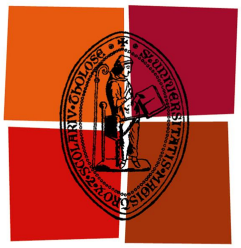

N° d'ordre :

THÈSE

présentée devant

l'université Paul Sabatier, Toulouse III

U.F.R. MATHÉMATIQUES, INFORMATIQUE ET GESTION

pour obtenir le titre de

DOCTEUR DE L'UNIVERSITÉ TOULOUSE

Mention INFORMATIQUE

par

SYLVAIN ROUGEMAILLE

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

Ingénierie des systèmes multi-agents adaptatifs dirigée par les modèles

---

Commission d'examen :

JURY

Juan PAVÒN Mestras	*Hdr, Universidad Complutense Madrid*	(rapporteur)
Philippe MATHIEU	*Professeur, université de Lille*	(rapporteur)
Marie-Pierre GLEIZES	*Professeur, université de Toulouse III*	(directeur de thèse)
Frédéric MIGEON	*Maître de conférences, université de Toulouse III*	(encadrant)
Christine MAUREL	*Maître de conférences, université de Toulouse III*	(co-encadrante)
Gauthier PICARD	*Maître assistant, école des mines de St Etienne*	(examinateur)

Sylvain Rougemaille

INGÉNIERIE DES SYSTÈMES MULTI-AGENTS ADAPTATIFS DIRIGÉE PAR LES MODÈLES.

Directeur de thèse :
Marie-Pierre Gleizes, Professeur
Université Paul Sabatier

Co-encadrement :
Frédéric Migeon, Maître de conférences
Université Paul Sabatier

Résumé

La complexité des systèmes informatiques actuels et en devenir implique la mise en œuvre d'approches nouvelles pour en faciliter le développement. La diffusion de plus en plus large de l'informatique et son usage de plus en plus nomade induisent des besoins importants en termes de réactivité et de tolérance vis-à-vis des modifications du contexte d'exécution. Dans ce cadre, nous proposons des moyens conceptuels, logiciels et méthodologiques pour faire face au problème de la conception de ces systèmes dits complexes. Nous nous appuyons pour ce faire sur la notion de système multi-agent adaptatif ainsi que sur le principe d'agent flexible. Ces deux paradigmes offrent en effet la capacité de maîtriser l'adaptation aux environnements dynamiques aussi bien au niveau du système qu'à celui des agents qui le composent. Le travail présenté ici établit un processus de conception faisant cohabiter ces deux notions grâce à l'Ingénierie Dirigée par les Modèles (IDM). Il s'appuie sur la définition de langages de modélisation dédiés et sur le traitement automatique des modèles qu'ils permettent de définir. Notre solution s'appuie sur une succession de transformations de modèles et de générations de code. Cette proposition s'inscrit en outre, dans une démarche méthodologique en enrichissant le processus de développement d'ADELFE (Atelier de DEveloppement de Logiciels à Fonctionnalité Emergente).

Institut de Recherche en Informatique de Toulouse - UMR 5505
Université Paul Sabatier, 118 route de Narbonne, 31062 TOULOUSE cedex 9

Sylvain Rougemaille

Model Driven Engineering for Adaptive Multi-Agent Systems.

Director :
Marie-Pierre Gleizes, Professor
Université Paul Sabatier

Co-supervisor :
Frédéric Migeon, Associate Professor
Université Paul Sabatier

Abstract

The complexity of current and future computer systems implies the implementation of new approaches to ease their development. The widening spread of those systems and their use that become more and more nomadic induce significant needs in terms of robustness and flexibility. A challenging issue is the definition of systems that are able to modify their behavior according to changes in their execution context. Within this specific framework we propose conceptual and methodological means as well as tools to address the problems raised by the design of complex systems. Our approach is based on the concepts of adaptive multi-agent systems and the principle of flexible agent. These two paradigms offer the ability to control adaptation to dynamic environments both at the system and the agent level. The work presented here establishes a design process that make these two concepts coexist thanks to the principles promoted by the Model Driven Engineering (MDE). It is based on the definition of domain specific modeling languages and automatic model processing. We propose a solution a set of model transformations and automatic code productions. This proposal integrates a methodological aspect as it provides an extension to the ADELFE development process.

Institut de Recherche en Informatique de Toulouse - UMR 5505
Université Paul Sabatier, 118 route de Narbonne, 31062 TOULOUSE cedex 9

Je dédie cette thèse
à Malory.

JE tiens à remercier l'équipe SMAC pour son accueil, sa disponibilité, ses conseils, l'enthousiasme et la motivation communicatifs qui la caractérisent.

Je souhaite exprimer toute ma reconnaissance à Monsieur Juan Pavón Mestras et Monsieur Philippe Mathieu pour l'honneur qu'ils me font en acceptant d'être les rapporteurs de ce mémoire ainsi que pour leurs remarques et conseils qui ont permis d'en améliorer la rédaction

Je tiens à remercier Monsieur Gauthier Picard pour avoir accepté de participer en tant qu'examinateur à mon jury de thèse et pour son travail qui fut une réelle inspiration pour mes contributions.

Mes remerciements vont, en particulier, à Marie-Pierre Gleizes pour m'avoir intégré à l'ensemble des travaux de l'équipe, m'ayant ainsi permis de m'épanouir tant du point de vue scientifique que personnel au sein de l'équipe SMAC.

Je souhaite également apporter mes remerciements à Pierre Glize qui m'a énormément soutenu pour mettre au point ce document et plus largement inspiré pour l'ensemble de mes travaux grâce aux nombreuses réunions et entretiens que nous avons pu avoir à ce sujet.

Je voudrais exprimer la plus profonde des gratitudes à l'égard de mes encadrants, Frédéric Migeon et Christine Maurel, qui me soutiennent et me motivent depuis plusieurs années déjà. Je souhaite leur faire part du bonheur que j'ai eu à travailler avec eux : aussi bien dans le cadre des enseignements que de la recherche. Au delà de ces remerciements, je voudrais leur exprimer plus largement mon amitié et l'estime sincère que je leur porte pour tout ce qu'ils ont pu faire pour moi.

Je remercie également Jean-Paul Arcangeli pour les discussions scientifiques, bien évidemment extrêmement relevées, que nous avons pu avoir et anecdotiquement pour m'avoir supporté dans son bureau au cours de cette dernière année.

Je souhaite remercier mes compagnons d'infortune Guillaume et Vincent pour leur humour plus que douteux, leurs conseils toujours avisés, leur aide et leur amitié.

Un merci tout particulier à celle qui partage ma vie, Malory, qui m'a toujours compris et soutenu pour que je puisse arriver jusqu'ici. Je ne lui dirai jamais assez : merci !

Un grand merci à tous mes amis qui ont toujours eu confiance en moi, pour tout les moments que nous avons pu partager et partagerons encore. Ils m'ont permis, sans doute à leur insu, d'aller de l'avant et d'atteindre mes objectifs.

Enfin je voudrais remercier mes parents et mes sœurs sans qui je n'aurais pas pu mener à bien mon parcours universitaire. Plus personnellement, je tiens à les remercier de leur affection, de leur soutien et de l'aide qu'ils ont su m'apporter.

Cet exercice de style s'avère plus ardu que je ne l'aurais pensé... Je souhaiterais n'omettre personne mais cela me semble difficile. Aussi, je remercie par anticipation tous ceux qui liront ces lignes en ayant le sentiment d'avoir été oublié dans la liste précédente.

Table des matières

Introduction générale

Contexte de travail

Les domaines d'application des systèmes informatiques actuels sont de plus en plus vastes et complexes. Il leur est demandé de gérer de plus en plus d'interactivité, de réactivité, de mobilité dans leurs utilisations les plus répandues (téléphonie mobile, informatique nomade, etc.). Les domaines de l'ingénierie des systèmes embarqués voire critiques, des sciences sociales, des sciences physiques ou encore de l'économie adoptent de plus en plus les modèles numériques pour la simulation et le prototypage. Là encore, la fonction dévolue aux systèmes informatiques atteint un degré de complexité de plus en plus important. Il est donc nécessaire de proposer des solutions conceptuelles ou méthodologiques, pour faire face aux difficultés croissantes du développement logiciel.

Un des problèmes majeurs que soulèvent ces nouveaux systèmes, réside dans la prise en compte et l'adaptation à des contraintes dynamiques. Suivant le domaine d'application, ces contraintes peuvent provenir de l'utilisateur, de l'infrastructure, de l'environnement ou du problème à résoudre. Quelle que soit la source de ces contraintes, l'important est de pouvoir exprimer des systèmes capables de s'y conformer dynamiquement, de s'y adapter par eux-mêmes.

La notion de système multi-agent est initialement apparue dans le cadre de l'intelligence artificielle. Les caractéristiques de ce type de systèmes leur confère une grande souplesse et une réactivité vis-à-vis des modifications de leur environnement. Ceci explique le succès qu'ils remportent depuis plusieurs années, notamment comme support de simulation ou de résolution de problèmes. Toutefois, leurs capacités de communication et de raisonnement ne sont pas nécessairement mises au service de l'auto-adaptation.

L'approche par Système Multi-Agent Adaptatif (AMAS, Adaptive Multi-Agent System), proposée par notre équipe, permet de répondre à cette problématique en se basant sur les principes d'auto-organisation et d'émergence. L'idée maîtresse est de tirer profit de la coopération entre agents pour faire émerger une solution adaptée à l'environnement dans lequel le système s'exécute. Nous souhaitons grâce à cette approche apporter des solutions pratiques à la complexité croissante des systèmes informatiques.

Problématiques

Comme nous venons de le voir les systèmes informatiques et artificiels requièrent, de plus en plus une flexibilité et une réactivité face aux dynamiques multiples de leur environnement qui les rendent particulièrement complexes. L'objet de ce travail est de proposer une solution au problème de la conception de ces systèmes complexes évoluant dans des environnements dynamiques. Nous souhaitons permettre à des utilisateurs, non spécialistes de l'émergence ou de l'auto-adaptation, de concevoir des systèmes offrant de telles capacités sans que cela ne constitue un surcoût de développement.
Nous disposons pour ceci d'un paradigme, celui des SMA Adaptatifs, sur lequel nous reposer pour satisfaire aux besoins d'auto-adaptation des systèmes. Nous voulons offrir des concepts et des outils permettant de faciliter la conception et le développement de ces systèmes dans un cadre méthodologique précis, celui du processus d'ADELFE (Atelier de DEveloppement de Logiciel à Fonctionnalité Emergente). La principale difficulté à laquelle il faut faire face lors de l'étude des systèmes complexes est qu'il n'est pas possible de déterminer les constituants du système et leurs interactions à partir de l'observation du comportement du système global (par exemple les fluctuations des marchés financiers). L'idée majeure proposée par l'approche AMAS est de focaliser l'attention sur le niveau micro (les agents constituant le système) en respectant des principes de coopération. Ces critères de coopération permettent aux agents de s'auto-organiser et de garantir le comportement adéquat du système au niveau global. Nous voulons porter cette vision à un plus haut niveau d'abstraction, c'est-à-dire, au moment de la conception. Cela pose tout de même le problème de la prise en charge de ces principes, de leur formalisation, afin de les intégrer au processus de développement. Processus qui doit, de surcroît, mener à une implémentation du système telle que la coopération des agents soit garantie. Il s'agit par conséquent, de fournir les moyens conceptuels, méthodologiques, techniques pour permettre de faciliter la conception et l'implantation des systèmes auto-adaptatifs.

Contributions

Les travaux présentés dans ce document sont situés à la frontière de plusieurs domaines : le génie logiciel et notamment l'approche dirigée par les modèles, les SMA adaptatifs, les architectures logicielles à composants et le principe d'agent flexible. L'une de mes premières contributions a été d'intégrer les capacités issues de ces différents domaines pour proposer une approche visant à simplifier le développement des systèmes complexes. L'ensemble de mes contributions s'inscrivent sur plusieurs plans :

– *Sur le plan des SMA adaptatifs :*
  J'ai décrit une spécification semi-formelle (grâce à un méta-modèle) des AMAS, en m'attachant particulièrement à préciser l'agent et son comportement. Ceci m'a permis de définir un langage de modélisation spécifique (AMAS-ML) dédié à la conception

détaillée des agents coopératifs. Ce dernier offre au concepteur un moyen de modéliser leur comportement coopératif.
Je propose, en outre, une séparation des préoccupations dans l'implantation des agents, en dissociant les aspects opératoires des aspects comportementaux. Les aspects opératoires des agents ont eux aussi été formalisés au sein d'un méta-modèle et peuvent être modélisés grâce à un langage spécifique (μADL). Il garantit le respect d'un style architectural permettant l'adaptation dynamique des mécanismes opératoires de l'agent (agent flexible).

– *Sur le plan méthodologique :*
J'ai proposé une extension de la méthodologie ADELFE en y intégrant l'utilisation d'UML 2.0 et en en décrivant le processus à l'aide du nouveau standard de l'OMG SPEM 2.0. Auparavant ADELFE ne permettait pas de guider l'utilisateur dans le développement final du système. Pour palier cette limitation, j'ai mis en place une approche dirigée par les modèles initiés par la définition de méta-modèles et de langages de modélisation spécifiques. L'objet de cette démarche est d'automatiser un maximum de tâches par transformation de modèles. Cette chaîne de transformations facilite le travail des concepteurs et des développeurs en leur donnant les moyens de se concentrer sur les aspects spécifiques de leurs tâches, comme la description du comportement coopératif et en masquant les aspects techniques de leur mise en œuvre. Ceci m'a permis de décrire et d'ajouter à ADELFE une phase d'implémentation centrée sur les modèles plutôt que sur le code.

– *Sur le plan du support applicatif :*
J'ai exprimé les principes des agents flexibles sous la forme d'un méta-modèle et implémenté un éditeur graphique permettant de décrire des modèles d'agent s'y conformant (μADL). Cet éditeur a été intégré à un outil de génération d'API agent appelé MAY (Make Agent Yourself) qui permet de construire une bibliothèque Java, à partir d'une spécification des mécanismes opératoires d'agents en μADL. Cet outil de génération fait partie de la nouvelle phase d'implantation d'ADELFE. J'ai implanté une transformation de modèles permettant d'extraire depuis le modèle spécifique AMAS-ML les caractéristiques opératoires des agents coopératifs afin d'en dériver automatiquement une API (Application Programing Interface) spécifique.

Organisation du mémoire

Ce document est organisé en quatre parties ; la première est constituée de trois chapitres qui présentent la problématique et les objectifs de notre étude. Le premier chapitre décrit les domaines applicatifs dans lesquels nous souhaitons inscrire nos solutions. Il caractérise la notion de système complexe, les propriétés singulières de ce domaine et place la notion d'adaptation, plus particulièrement d'auto-adaptation, comme une caractéristique essen-

tielle. Le chapitre suivant décrit, quant à lui, le domaine des Systèmes Multi-Agents (SMA) comme une première approche pour faire face à cette complexité. Le troisième et dernier chapitre de cette première partie présente nos objectifs ainsi que les questions auxquelles nous souhaitons apporter des réponses.

La deuxième partie constitue une présentation de l'état de l'art des domaines dans lesquels nous inscrivons nos travaux et est, elle aussi, divisée en trois chapitres. Le premier chapitre concerne le génie logiciel, l'Ingénierie Dirigée par les Modèles et ses caractéristiques. Nous considérons cette approche comme incontournable dans l'optique de faciliter la conception et le développement des systèmes complexes. Le second chapitre propose un tour d'horizon des travaux menés dans le domaine de l'ingénierie des systèmes multi-agents et particulièrement ceux s'appuyant sur une démarche orientée modèle. Dans le troisième chapitre, nous présentons plus particulièrement la théorie des AMAS ainsi que le principe d'agent flexible. Ces deux approches constituent un moyen de maîtriser l'auto-adaptation du système et de ses agents. Du reste, ce chapitre détaille la méthodologie ADELFE dédiée aux AMAS qui sert de cadre méthodologique à l'ensemble de nos travaux.

Nous présentons par la suite, dans la troisième partie, notre contribution au développement des systèmes complexes. Cette partie présente dans un premier chapitre les aspects méthodologiques de nos travaux et les améliorations apportées au processus de développement d'ADELFE. Notre objectif d'apporter plus de facilité et de précision dans la conception et l'implantation des AMAS nous a conduit à spécifier de nouveaux méta-modèles. Le chapitre suivant décrit notre contribution en termes de langages de modélisation (utilisation et définition). Ces langages et leur intégration à ADELFE nous ont conduits à décrire un ensemble de transformations qui constituent le cœur de notre démarche orientée modèle. Ces ponts technologiques, transformations et générations de code, sont détaillés dans le chapitre suivant.

Afin de valider notre approche (modélisation et transformation), nous avons développé deux applications. Ces expérimentations et les conclusions que nous avons pu en tirer constituent l'objet de la quatrième et dernière partie. Nous y présentons en premier lieu, la définition d'un environnement de simulation destiné à l'étude du comportement de fourragement d'une colonie de fourmis. En second lieu, nous décrivons le développement d'une application de résolution de contraintes distribuées, dans le cadre spécifique de la gestion d'une chaîne de production. Ces deux exemples nous ont permis de constater un certain nombre d'apports et de limitations que nous évoquons pour conclure cette partie.

Au terme de ce document, nous tirons des conclusions d'ordre plus général et proposons un ensemble de perspectives potentielles à ces travaux.

Première partie

Contexte et objectifs

Introduction

L'objet de cette partie est de préciser le contexte général de notre travail et de caractériser les objectifs que nous poursuivons.

Nous présentons le domaine des systèmes informatiques complexes et les caractéristiques spécifiques que nous identifions. Ces caractéristiques vont nous permettre d'établir les besoins à prendre en charge pour faciliter leur conception et leur développement. L'un des besoins principaux que nous exhibons est la nécessité d'adaptation aux contraintes d'un environnement dynamique.

Nous étudions l'approche des systèmes multi-agents qui offrent une solution partielle à ce besoin. Notre attention se porte particulièrement sur les principes d'émergence et d'auto-organisation qui peuvent être mis à profit pour satisfaire les besoins des systèmes complexes en apportant une réponse théorique à la problématique de l'auto-adaptation.

Nous souhaitons offrir des moyens spécifiques et précis pour formaliser et modéliser cette notion d'auto-organisation coopérative afin de faciliter la conception et l'implantation de systèmes complexes. Le dernier chapitre précise cet objectif et décrit les problèmes auxquels nous devons apporter une réponse pour l'atteindre.

1 Les systèmes complexes

IL est un fait que l'on ne peut nier, les systèmes informatiques prennent de plus en plus de place dans nos vies, s'intègrent de plus en plus aux objets du quotidien, nous promettent de plus en plus d'interaction, où que nous soyons et que nous le désirions ou non. Cette diversification des supports, cette omniprésence des systèmes informatiques tend à complexifier grandement l'ensemble de leur cycle de vie (de la conception à la maintenance). Au-delà de cette constatation des plus évidentes, ce chapitre cherche à exprimer plus clairement ce que nous avons considéré comme des systèmes informatiques complexes (leurs caractéristiques) et où doivent se porter nos efforts afin que le travail que nous menons facilite le développement ou plus largement l'ingénierie de ces systèmes.

1.1 Caractéristiques

Abbott présente l'exemple d'un système Newtonien pour illustrer la notion de système complexe (Abbott, 2007). Poincaré trouva que trois corps obéissant à la gravitation universelle de Newton ont, sous certaines conditions, une trajectoire qui dépend fortement de la condition initiale. Ainsi, on ne pourra jamais déterminer avec exactitude le destin de ces corps, car la moindre perturbation dans ses mesures entraînerait irrémédiablement une forte différence de trajectoire. En suivant ces considérations, un système complexe soumis à des interactions non linéaires entre ses parties aura un comportement parfois imprévisible alors que les règles qui régissent ses constituants sont connues. Ainsi, un système peut être considéré comme complexe si son comportement ou son état, n'est pas prévisible a priori alors que les règles qui régissent ses constituants sont connues. C'est la multiplicité de ces règles et leurs interactions qui sont responsables de la complexité ou de la non-linéarité des observations au niveau du système. C'est-à-dire, que l'on ne peut pas réaliser une trace analytique entre les évènements du niveau macro et les interactions du niveau micro (irréductibilité). L'étude des systèmes complexes est donc par essence multi-niveau, un niveau *système* ou macro et un niveau *interne* ou micro. En outre, elle est indéniablement pluridisciplinaire. Le règne de la complexité s'étend des mathématiques à la biologie en passant par les sciences sociales ou plus récemment à l'informatique. Ces systèmes peuvent être abordés par bien des angles mais pour ce qui concerne nos travaux, nous nous intéressons aux systèmes complexes informatiques et à leur conception.

La notion de système complexe est intimement liée à la notion de phénomènes émergents. (Holland, 1998) donne un exemple intuitif de cette notion en considérant le jeu d'échec. La

complexité observable du jeu d'échec : le nombre de combinaisons ou de parties possibles, l'apparition de nouvelles stratégies envisageables pour gagner l'une d'entre elles n'est finalement issue que d'une douzaine de règles. Depuis son origine jusqu'à nos jours, le jeu d'échec continue de faire naître (émerger) pour le joueur de nouvelles manières de battre son adversaire. Pourtant, elles ne sont pas directement identifiables par la simple étude des règles qui contraignent une ou même l'ensemble des pièces. Dans un système, la nouveauté et l'apparition de régularités a priori impossibles à imputer à tel ou tel élément le constituant, caractérise l'émergence et identifie, par là même, ce système comme « complexe ».

Afin d'identifier plus précisément les systèmes auxquels nous portons une attention toute particulière, nous allons caractériser les propriétés que nous considérons comme révélatrices de la complexité d'un système. Bien entendu, il ne s'agit pas d'une liste exhaustive mais davantage des points que nous essayons de prendre en charge dans ce travail et dans l'ensemble des travaux de notre équipe :

- *la multiplicité des échelles* : on distingue le niveau du système de celui de ses constituants (dichotomie macro, micro),
- *l'ouverture* : on considère ici l'apparition ou la disparition possible d'éléments dans le système,
- *la dynamicité* : c'est-à-dire le changement fréquent de comportement du système et de ses constituants,
- *la réactivité* : il s'agit de l'aptitude d'un système à réagir au monde qui l'entoure, à s'organiser afin de prendre en compte de nouvelles contraintes,
- *l'hétérogénéité* : un système doit prendre en compte des éléments de diverses natures et en garantir les interactions,
- *la décentralisation* : on considère le fait de répartir le contrôle, l'activité du système au sein de nombreux éléments.

Un système peut donc être considéré comme complexe dès lors qu'il vérifie ces conditions (pas nécessairement l'intégralité). Ces caractéristiques peuvent provenir pour un système informatique, du problème que l'on cherche à résoudre ou du domaine d'application. Par exemple, dans le cadre de l'informatique ambiante ou ubiquitaire, les systèmes sont caractérisés par la *décentralisation* de leur activité au sein de dispositifs *hétérogènes* pouvant apparaître ou disparaître du fait de leur mobilité (*ouverture*) ce qui nécessite de leur part une grande *réactivité* face à ces aléas. Si l'on considère la modélisation et la simulation de phénomènes physiques ou sociaux, elles font face à la complexité issue du système qu'elles cherchent à abstraire : *dynamicité*, *multiplicité des échelles*, *ouvertures* , etc.

A partir de cette grille de lecture, nous pouvons nous pencher davantage sur les types de systèmes ou les domaines qui remplissent ces différentes caractéristiques ; le paragraphe qui suit en fait l'objet.

1.2 Systèmes informatiques complexes

Comme nous l'avons vu, les systèmes dit complexes se trouvent à la frontière de nombreuses disciplines et peuvent être abordés sous différents angles. Quoi qu'il en soit, dans le cadre qui nous intéresse plus particulièrement, on peut distinguer de nombreux systèmes répondant totalement ou partiellement aux critères de caractérisation que nous avons essayé de dégager dans le paragraphe précédent. Ces différents types ou domaines que nous soulignons ici constituent des champs d'investigation privilégiés pour nos travaux.

1.2.1 Informatique ubiquitaire, intelligence ambiante

La notion d'informatique ubiquitaire, pervasive ou enfouie est originellement présentée par les chercheurs du Xerox Palo Alto Research Center dans (Weiser, 1999). Ils la caractérisent comme étant l'avenir des systèmes informatiques. Les propriétés principales de ces systèmes sont leur omniprésence ainsi que leur « dimension tacite »rendues possibles par les progrès techniques et la miniaturisation. En d'autres termes, ces systèmes disparaissent à notre vue et nous en bénéficions sans même en avoir conscience. Weiser insiste sur le fait que « le réel pouvoir de ces concepts ne provient d'aucun d'entre eux - il émerge de leur interaction... » , le tout est plus que la somme des parties. Cette constatation, à la base du courant de pensée systémique, constitue un présupposé majeur à l'étude des systèmes complexes.

Cette notion d'ubiquité des systèmes a donné naissance à un concept originellement issu du domaine industriel : l'intelligence ambiante (AmI). Elle cherche à tirer profit de cette ubiquité en y ajoutant, au centre des préoccupations, la dimension utilisateur. Son objectif est justement la mise en pratique de cette omniprésence invisible pour assister, divertir ou surveiller l'utilisateur humain dans son activité quotidienne. (Aarts, Emile, 2005) présente l'AmI comme un domaine où l'innovation doit être importante dans les années à venir, considérant un partenariat entre les différentes parties du monde académique et industriel. Outre cette vision économique du concept, dans sa présentation, (Aarts, Emile, 2006) explicite les caractéristiques principales ou souhaitées d'un système ambiant :

- *embarqué* : constitué de nombreux équipements dispersés dans l'environnement,
- *conscient de son environnement* : il connaît sa situation dans son environnement,
- *personnalisé* : il peut être adapté à des besoins ou un individu particuliers,
- *adaptatif* : il peut changer de réponse envers son utilisateur ou son environnement,
- *possède une capacité d'anticipation* : il doit anticiper sur les besoins de l'utilisateur sans requérir d'action de sa part.

L'intelligence ambiante constitue un cadre que l'on peut considérer comme particulièrement complexe. En effet, il est soumis à de fortes contraintes liées notamment à la *dynamicité* de l'environnement (prise en compte du facteur humain), à la nécessité d'y apporter une réponse adaptée (personnalisation, adaptation dynamique au contexte) ainsi qu'à l'hétérogénéité des supports d'exécution (un téléphone mobile, un téléviseur, etc.).

1.2.2 Simulation

La réalisation de modèles est un processus inhérent à la compréhension du monde qui nous entoure. Abstraire un élément du monde réel pour tenter d'en comprendre ou d'en décrire le fonctionnement est une activité fréquente dans le domaine des sciences ou de l'ingénierie et ce depuis leurs origines (cf. Marc Vitruve Pollion, Leonardo di ser Piero da Vinci, etc.). Au-delà de la réalisation d'abstractions de systèmes ou de phénomènes complexes, la simulation permet d'animer celles-ci afin de définir et d'expérimenter des hypothèses sur leur fonctionnement. (Shannon, 1998) présente la simulation comme *« un des outils les plus puissants à la disposition des responsables de la conception et du fonctionnement de processus et de systèmes complexes. »*. Il s'agit effectivement d'un outil au service des scientifiques, des décideurs pour tenter d'appréhender la complexité de phénomènes économiques, biologiques ou sociologiques. L'avènement de l'informatique et la puissance calculatoire qu'elle fournit permet de concevoir des abstractions numériques exécutant ou simulant le comportement de systèmes complexes. (Shannon, 1975) définit la simulation comme *« la modélisation informatisée d'un système réel permettant, au travers d'expérimentations conduites sur ce modèle, d'en comprendre le comportement ou bien d'évaluer différentes stratégies de fonctionnement »*. En effet, la simulation peut avoir comme finalité la compréhension d'un système, son prototypage dans le cadre de l'ingénierie automobile ou encore aéronautique.

La modélisation est une des premières tâches à réaliser lorsque l'on mène une activité de simulation. Celle-ci a pour finalité la réduction de la complexité des systèmes qu'elle étudie en omettant les aspects non pertinents. Cependant, la puissance calculatoire des machines actuelles permet d'imaginer des simulations de plus en plus complètes et complexes. Il n'est pas rare de combiner plusieurs modèles mathématiques ou numériques, mettant en jeu un grand nombre d'éléments en interaction et de paramètres au sein d'une même simulation. La puissance nécessaire à ce type de simulation implique l'utilisation de machines extrêmement coûteuses ou celle de nouvelles approches. Ainsi, nous pouvons imaginer utiliser des grappes de machines géographiquement distribuées [1] ou des machines personnelles. Cela induit bien évidemment de nouvelles contraintes en termes de distribution du contrôle, des données, de la synchronisation qui complexifient encore la tâche des développeurs de ce type d'application.

1.2.3 Intelligence collective

Issu en partie de l'étude des systèmes biologiques notamment des insectes sociaux (Bonabeau et al., 2001) et pour une autre part des automates ou robots cellulaires (Wolfram, 1986), ce domaine tend à concevoir des systèmes informatiques ou robotiques dont « l'intelligence » réside dans le collectif plutôt que dans l'individu. (Beni, 2004) présente le principe de « swarm » , essaim dans la langue de Molière, comme un type de groupe ayant les ca-

1. Projet de grille de calcul français, utilisé entre autre dans le cadre de simulations `https://www.grid5000.fr`

ractéristiques suivantes : *« ... un contrôle décentralisé, pas de synchronisation, des membres simples et (quasi) identiques. »*. Il met aussi l'accent sur l'importance du nombre d'éléments dans le système et développe de cette manière la métaphore entomologique un peu plus loin. Nous pouvons donner en guise d'exemple les algorithmes d'optimisation basés sur le comportement d'une colonie de fourmis (Colorni et al., 1991), les algorithmes de vols ou de bancs qui simulent des comportements grégaires (Reynolds, C. W., 1987) pouvant être appliqués à la coordination multi-robot, etc. Nous pouvons prendre comme exemple, le projet ANTS (Autonomous Nano Technology Swarm) [2] du laboratoire de génie logiciel de la NASA. Il s'agit d'un concept de mission basé sur la notion d'essaim qui appartient dans un projet d'exploration spatiale de la ceinture d'astéroïdes (Rouff et al., 2006).

Tous ces exemples entrent dans le cadre des systèmes complexes : par la décentralisation de leur contrôle et de leur fonction, le grand nombre d'individus qui les composent, le grand nombre d'interactions, l'imprévisibilité de la dynamique globale du système et par le caractère émergent des solutions qu'ils proposent.

1.2.4 Résolution de problèmes

La résolution de problèmes de satisfaction de contraintes fait intervenir un nombre important de paramètres ou variables interdépendants (contraintes) dont les valeurs prises dans un domaine particulier varient en cours de résolution. La coloration d'une carte géographique est un problème classique de ce domaine ; la conjecture des quatre couleurs de Francis Guthrie (1852) ne fut prouvée qu'à l'aide d'un ordinateur dans les années 70 (Appel and Haken, 1977). Pourtant, il ne représente qu'un sous-ensemble des problèmes de satisfactions de contraintes, ceux dont le nombre de variables est fini (ensemble de pays ou région à colorier), le domaine de ces variables est discret et fini (un ensemble de couleurs) et les contraintes sur ces variables sont binaires (deux régions contiguës ne peuvent avoir la même couleur) (Russel and Norvig, 1995). Dans le cas le plus général mais aussi le plus complexe, les domaines de valeurs des variables sont continus infinis et les contraintes entre ces variables sont linéaires voire non-linéaires. Tous les problèmes liés à l'ordonnancement de tâches, d'allocation de ressource ou de logistique en général entrent dans ce cadre. Une approche pour résoudre ces problèmes est la recherche locale de solution pour laquelle il est nécessaire minimiser les conflits dont chaque élément ou variable du problème pourrait être la source. Cette approche peut être appliquée à de nombreux domaine comme l'assistance à la conception de systèmes aéronautiques, automobiles où de nombreuses contraintes physiques et budgétaires, souvent contradictoires, entrent en jeu. Ces diverses contraintes peuvent évoluer au cours du temps (Welcomme et al., 2007). La complexité de ce domaine provient du nombre de contraintes et de variables, du fait que le domaine de ces variables est potentiellement non-linéaire et que tous ces aspects peuvent évoluer dynamiquement. Nous considérons donc les systèmes voués à résoudre ce type de problèmes de satisfaction de contraintes et d'optimisation multi-critère comme particulièrement complexes.

2. `http://ants.gsfc.nasa.gov/`

Les différents domaines que nous venons d'explorer représentent un panel, certes non-exhaustif, mais somme toute représentatif des problèmes auxquels nous essayons d'apporter des réponses. Dans ces systèmes caractérisés par des dynamiques externes (de la part de l'environnement) aussi bien qu'internes (de la part de leurs constituants), l'adaptation au contexte apparaît comme une qualité nécessaire, mais sans doute pas suffisante. Le paragraphe ci-dessous présente la notion d'auto-adaptation des systèmes dont le principal objectif est de leur fournir les moyens de gérer par eux-mêmes la dynamicité et les variations de leur environnement d'exécution.

1.3 Auto-adaptation

L'adaptation est une caractéristique nécessaire aux systèmes que nous venons de présenter, afin de maintenir leur fonctionnalité face à des modifications de leur environnement. Laddaga présente la notion de logiciel auto-adaptatif, c'est-à-dire *« capable de surveiller, comprendre et modifier sa fonction à l'exécution »* (Laddaga, 2001). L'objectif est de permettre de réagir au dynamisme de l'environnement d'exécution afin de fournir une nouvelle fonctionnalité ou d'améliorer la qualité de celles déjà rendues (Robertson et al., 2001). Bien entendu, elle nécessite de la part du système une connaissance de cet environnement qui l'entoure. Dans ce paragraphe, nous nous intéressons donc particulièrement à la manière dont l'environnement des systèmes peut influencer leur fonction, leur structure ou leur état.

1.3.1 Couplage environnement-système

L'idée qu'un système naturel réagit aux pressions exercées par son environnement en s'adaptant, n'est pas nouvelle. Elle est à l'origine de la théorie de l'évolution des espèces (Darwin, 1859). Cependant, cette vision restreint la relation entre l'environnement et les systèmes qu'il contient. (Maturana and Varela, 1994) présentent cette relation comme un « couplage » où une unité structurale (un système) interagit avec son environnement pour atteindre une « congruence structurale » , c'est-à-dire un ensemble de changements structuraux mutuels déclenchés par des interactions récurrentes. Cet état de stabilité dynamique est maintenu jusqu'à ce que l'intégrité du système soit mise en défaut par une interaction destructrice. Cette vision d'interdépendance forte entre système et environnement peut être mise à profit dans le cadre de systèmes informatiques afin de garantir le fait qu'un système, tant que son intégrité n'est pas mise en jeu, s'adapte aux changements de son environnement (Glize, 2001).

1.3.2 Sensibilité au contexte

Pour qu'un système puisse s'adapter aux modifications de son environnement il lui est nécessaire de posséder une certaine « conscience » de sa situation dans le monde qui l'entoure. (Schilit et al., 1994) introduisent la notion de systèmes conscients de leur contexte : *« ..context-aware software adapts according to the location of use, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time. A system with these capabilities can examine the computing environment and react to changes to the environment »* . Cette notion, bien qu'ici employée dans le cadre de l'informatique ubiquitaire, peut se généraliser à l'ensemble des systèmes requérant des capacités d'adaptation. Ce qui, pour nous, peut donc se généraliser à l'ensemble des systèmes que nous considérons complexes (voir paragraphe 1.1). Il s'agit, de fait, de la première des qualités souhaitables pour qu'un système s'auto-adapte aux changements ; il doit être capable de les percevoir et de les représenter.

Néanmoins, comme nous l'avons déjà souligné, il est nécessaire de ne pas omettre le caractère *multi-échelle* des systèmes complexes. L'adaptation doit alors être envisagée aussi bien au niveau macro que micro. Ce principe nous a conduit à la définition des caractères primordiaux de l'adaptation à ces différents niveaux.

1.3.3 Niveaux d'adaptation

La notion d'adaptation, de couplage est relative à l'environnement. Or, la notion d'environnement est elle-même éminemment subjective et prête à confusion. Cette notion nécessite de distinguer ce qui est du ressort du système de ce qui ne l'est pas, ce qui dépend du support physique de ce qui constitue l'infrastructure logicielle ou logique (Weyns et al., 2004). En d'autres termes, où est le système et quel est son environnement, son infrastructure ? A partir du modèle en couches présenté par (Weyns et al., 2005) nous distinguons dès lors, deux aspects dans l'environnement :

- *Environnement physique* : ensemble des propriétés et des éléments liés au support physique ou à l'infrastructure logicielle sur laquelle repose le système (OS, intergiciel, protocoles réseaux, etc.). Elle correspond aux deux plus basses couches du modèle de Weyns.
- *Environnement logique* : niveau lié aux besoins applicatifs, c'est-à-dire tout ce qui ne fait pas partie de la fonction du système mais qui est nécessaire à sa réalisation, ce qui correspond à la couche *« system application »* du modèle de Weyns.

Cette vision de l'environnement implique différents niveaux ou degrés d'adaptation. En outre, les mécanismes mis en jeu dans la re-configuration topologique d'un système afin de palier, par exemple, une panne réseau, ne sont pas les mêmes que ceux mis en œuvre lors d'un changement de mode d'interaction (passage d'une communication sécurisée à non-sécurisée). Dans le premier cas, c'est le système qui doit modifier sa structure, sa topologie ; dans le second ce sont ses constituants qui doivent modifier leur manière d'interagir. Dans

les deux cas, la fonctionnalité du système se doit d'être conservée.

Les caractéristiques des systèmes complexes les rendent particulièrement difficiles à mettre en œuvre ; pourtant, leur domaines respectifs constituent des défis pour l'informatique actuelle et à venir. C'est dans l'optique de faciliter le développement de ces systèmes que nous positionnons nos travaux. Aussi, dans le chapitre qui suit, nous présentons différents modèles que nous considérons comme primordiaux pour permettre de simplifier leur conception et leur implantation.

2 Systèmes multi-agents et auto-adaptation

LES Systèmes Multi-Agents (SMA) constituent un paradigme particulièrement adapté dès lors que l'on désire définir un système décentralisé, adaptable, muni de faculté de perception lui permettant de répondre dynamiquement aux contraintes de son environnement. Ce chapitre présente tout d'abord les SMA dans leur acception la plus générale puis s'attache à souligner l'importance de l'auto-organisation et de l'auto-adaptation dans ces systèmes. Nous présentons ensuite des approches développées dans notre équipe afin de garantir ces propriétés au sein des SMA.

2.1 Définitions

Les Systèmes Multi-Agents (SMA) sont issus du domaine de l'intelligence artificielle distribuée et sont désormais bien représentés dans le paysage informatique en termes de langages, de plate-formes de développement et de conception. Cette notion est notamment reconnue comme une première approche pour maîtriser la complexité des systèmes ou du moins permettre de la modéliser, la simuler. Il s'agit d'un paradigme simplificateur à bien des égards qui permet de considérer un système comme un ensemble d'individus autonomes ayant des capacités d'interaction et de raisonnement.

2.1.1 Agent

Un système multi-agent est avant tout un système constitué d'agents. Pour être plus précis sur le concept d'agent nous pouvons nous référer à la définition de (Ferber, 1999) qui présente un agent comme une entité :

- capable d'agir dans un environnement,
- pouvant communiquer directement avec d'autres agents,
- mue par un ensemble de tendances (objectifs, fonction de satisfaction, etc.),
- possédant des ressources propres,
- capable de percevoir une partie limitée de son environnement,
- possédant une représentation partielle, voir nulle de cet environnement,
- possédant des compétences et offrant des services,
- pouvant éventuellement se reproduire,
- se comportant pour atteindre ses objectifs en fonction des perceptions, représentations et communications qu'elle reçoit et grâce aux compétences et ressources qu'elle pos-

sède.
Une première constatation qui peut être faite est qu'il ne paraît pas exister d'agent hors de son environnement. De surcroît, il possède les moyens d'acquérir et de stocker des informations à son sujet. De même, Wooldridge (1999) caractérise les agents comme des entités capables de percevoir leur environnement et de réagir en fonction de ces perceptions, c'est-à-dire réactif (voir figure 2.1). Il différencie aussi la notion d'*« agent intelligent »* comme étant pro-actif, capable de prendre des décisions et social ou interagissant avec ses semblables. Cependant, dans la suite de ce document nous ne distinguerons pas la notion d'agent intelligent de la notion d'agent, il s'agit dans les deux cas d'agents. La notion d'agent apparaît donc parfaitement convenir aux besoins exprimés en termes d'adaptation.

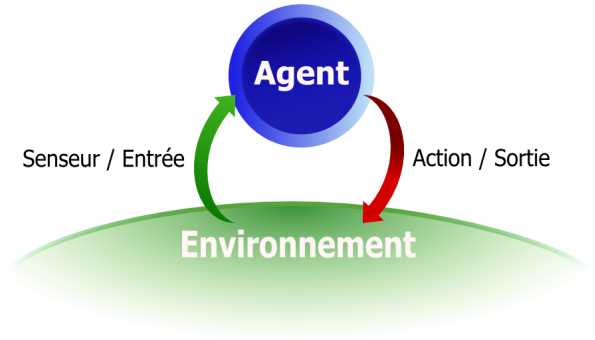

Figure 2.1 — Agent réactif et environnement (Wooldridge, 1999).

2.1.2 Système Multi-Agent

Un agent seul n'est pas plus capable qu'une autre entité de gérer la complexité des systèmes, si nous nous référons au paragraphe 1.2.3, mieux vaut compter sur le collectif. Néanmoins, un ensemble d'agents regroupés au sein d'un système dans lequel ils peuvent interagir tout en bénéficiant de leurs capacités de perception et de décision, rend l'approche multi-agent parfaitement adaptée aux problèmes que nous évoquions dans la section 1.1. (Ferber, 1999) définit un Système Multi-Agent comme étant composé :

- d'un *environnement* : un espace disposant généralement d'une métrique,
- d'un ensemble d'*objets* passifs (modifiables par les agents) situés dans l'environnement,
- d'un ensemble d'*objets* particuliers : les *agents* représentant les entités actives du système,
- d'un ensemble de *relations* unissant les *objets* entres eux,
- d'un ensemble d'*opérations* décrivant les interactions entre les objets et les agents,
- d'un ensemble de *lois* représentant les contraintes imposées par l'environnement sur les *opérations*.

L'approche multi-agent propose un paradigme permettant de distribuer le contrôle (physiquement ou logiquement) au sein d'unités autonomes en interaction avec leurs semblables

ainsi que leur environnement, capables d'en percevoir des modifications locales et de réagir en fonction. Comme l'exprime Wooldridge (1999), l'approche orientée agent permet de mettre en oeuvre *« des systèmes flexibles et autonomes »* répondant en grande partie aux besoins des systèmes complexes. Ce qui peut se résumer dans notre optique par l'idée que les systèmes multi-agents répondent aux attentes liées au développement de systèmes complexes, par leur capacité de perception du contexte et leur aptitude à formuler une réponse adaptée à ses modifications.
Par conséquent, s'appuyer sur une approche SMA constitue un premier pas vers la simplification des problèmes liés à la prise en compte de l'environnement, l'adaptation et l'auto-organisation dans les systèmes complexes. Dès lors, comment peut-on maîtriser ces aspects d'auto-organisation et d'émergence afin que ceux-ci nous servent à répondre aux problèmes présentés dans le chapitre 1 ?

2.2 Emergence et auto-organisation

Le principe d'émergence a été abordé dans le chapitre 1 comme un moyen de caractériser les systèmes complexes. Nous pouvons nous représenter cette notion par l'idée qu'à partir de l'activité d'un ensemble d'éléments localement en interactions, des propriétés nouvelles et ostensibles apparaissent au niveau global. Georgé et al. conditionnent la caractérisation d'un phénomène émergent aux critères suivants (Georgé et al., 2003) :

- l'observation de phénomènes *ostensibles* et *radicalement nouveaux* au niveau global ou macro,
- la *cohérence* de l'observation, le phénomène possède une identité propre dont la cohésion dépend fortement des parties qui le constituent (Goldstein, 1999),
- l'observation de *dynamiques*, le phénomène est maintenu dynamiquement, les dynamiques du niveau micro produisent celles du niveau macro qui les contraignent en retour (Langton, 1990).

Une des idées fortes de l'émergence est qu'il n'est pas possible de réaliser une trace évidente entre les propriétés observées et les éléments qui en sont à l'origine (Van De Vijver, 1997).
Afin d'illustrer la notion d'émergence et d'auto-organisation, nous allons présenter quelques exemples classiques de ces phénomènes issus de différents domaines.

2.2.1 Une approche intuitive de l'émergence

De nombreux exemples dans le domaine de la biologie, la sociologie, la physique ou la chimie peuvent être pris pour illustrer l'émergence. L'apparition du langage par exemple, serait un phénomène émergent (Keller, 1994). Plus proche de nos considérations, nous pouvons noter les différents phénomènes observables chez les insectes sociaux, comme le comportement d'une colonie de fourmis. En effet, leur perception limitée de l'environnement qui les entoure, leur communication par le biais de phéromones (stygmergie) permettent de découvrir le plus court chemin entre leur nid et une source de nourriture (Bonabeau et al., 2001).

Ce n'est pourtant pas l'objectif individuel de ces insectes, c'est une propriété qui émerge du fait d'un mécanisme de renforcement de la piste la plus courte par un plus grand nombre de passages. Un autre exemple classique est l'expérience de Belousov-Zhabotinsky où une organisation visuelle apparaît lors de sa réalisation dans une boîte de pétri. Cette configuration émerge grâce à un mécanisme de réaction diffusion, de boucles de rétroaction (amplification, retours négatifs) entre les éléments du système et leur environnement. Ce type de phénomènes est qualifié d'auto-organisation.

2.2.2 Auto-organisation et adaptation

L'auto-organisation est un mécanisme observable dans de nombreux domaines comme nous venons de l'illustrer ci-dessus. Heylighen (2001) présente l'auto-organisation dans les différents domaines dans laquelle elle fut observée ou étudiée, il définit ce phénomène comme *« l'émergence spontanée d'une cohérence globale à partir d'interactions locales »* . Camazine décrit l'auto-organisation comme *« un processus dans lequel une organisation au niveau global du système émerge uniquement des nombreuses interactions entre les composants de niveau inférieur du système »* (Camazine et al., 2001). Maturana et Varela présentent la notion d'autopoïèse qui consiste à considérer un système naturel comme capable de s'auto-créer, de maintenir son organisation tout en modifiant éventuellement sa structure en raison d'interactions mutuelles avec son environnement (couplage) (Maturana and Varela, 1994). Cette idée que les systèmes atteignent d'eux-mêmes un état remarquable *« autour de l'équilibre »* (Nicolis and Prigogine, 1977) avec leur environnement en s'auto-organisant nous permet d'envisager ce mécanisme comme un moyen de mener à bien l'adaptation fonctionnelle du système (cf. paragraphe 1.3.3). En effet, si nous considérons la fonction souhaitée du système comme une propriété émergeant des interactions de ses constituants alors nous pouvons admettre que cette fonction s'adaptera aux changements de l'environnement puisqu'il constitue l'élément perturbateur à l'origine de l'auto-organisation.
Le concept d'*« emergent computation »* fut proposée pour la première fois par (Forrest, 1990). L'idée est d'essayer de maîtriser l'émergence pour fournir une fonction, un service qui soit l'émanation de l'ensemble des actions réalisées par des éléments. Nous nous intéressons à ce principe et l'utilisation qu'il peut en être faite pour réaliser des calculs ou une fonction dans le cadre des SMA. C'est dans cette optique que (Georgé et al., 2004) propose la notion de SMA Adaptatif (Adaptive MAS). Les AMAS constituent un moyen de maîtriser l'auto-organisation des systèmes afin de garantir l'émergence d'une fonction globale souhaitée en respectant des critères spécifiques.

3 Objectifs et verrous pour l'ingénierie des systèmes complexes adaptatifs

NOUS l'avons vu, la notion de système complexe s'applique à un grand nombre de domaines de l'informatique. Il paraît dès lors nécessaire de considérer avec attention leur implantation et plus encore, leur conception. Il nous faut étudier les moyens à notre disposition pour permettre de concevoir ces systèmes avec le plus de facilité et de précision possible. C'est dans cette voie que nous inscrivons nos travaux. Nous souhaitons donner aux développeurs logiciels des moyens adaptés pour maîtriser la complexité croissante des systèmes et particulièrement leur besoin en auto-adaptation.

3.1 Objectifs

Notre objectif le plus large est d'apporter des moyens de faciliter la conception et le développement de systèmes informatisés complexes. Le caractère multi-échelle de ces systèmes et l'incapacité que nous possédons de lier directement ces niveaux entre eux (cf. paragraphe 1.1), constituent les premiers écueils à éviter dans le cadre de leur conception. De plus, les domaines d'application de ces systèmes doivent répondre à des contraintes d'adaptation très importantes (cf. paragraphe 1.3). Nous pouvons par conséquent décliner cet objectif général en plusieurs points :

– *Prise en compte de l'adaptation*

  Nous souhaitons proposer les moyens de garantir l'adaptation des systèmes à un niveau d'abstraction élevé et la prendre en charge dès la conception.

– *Formalisation de l'approche SMA adaptatif*

  Les caractéristiques émergentes des systèmes complexes, induisent l'utilisation de systèmes capables de s'auto-organiser. Nous souhaitons formaliser la notion de SMA Adaptatif afin qu'elle puisse être utilisée comme un moyen de spécification précis à la conception. A cette fin, il faut décrire précisément le modèle d'agent coopératif qui constitue la base de notre approche.

– *Formalisation des comportements coopératifs*

  La coopération entre agent est le moteur de l'auto-organisation du système et garantit

son adéquation fonctionnelle. Il est nécessaire d'offrir des outils de conception précis et spécifiques pour pouvoir exprimer les règles qui régissent cette coopération.

– *Automatisation du développement*

Afin d'assister les tâches des concepteurs et des développeurs, il est important de leur permettre de se focaliser sur les tâches pour lesquels ils sont particulièrement qualifiés. En automatisant celles qui ne sont pas directement de leur ressort, nous permettons de masquer une partie de la complexité du développement.

– *Classification des activités*

Dans un souci de simplifier également l'implantation de ces systèmes, nous souhaitons proposer une approche permettant de séparer les aspects comportementaux (coopération) des aspects plus *« matériels »* des agents. Ceci permettrait de masquer, pour le développeur du comportement, les mécanismes qui gèrent la communication, la migration ou le cycle de vie de l'agent, sans toutefois lui interdire d'y accéder s'il le juge nécessaire.

Ces objectifs s'inscrivent dans une démarche de génie logiciel, en proposant des moyens pour abstraire, spécifier des systèmes auto-organisateurs et les outils méthodologiques en décrivant l'utilisation. Qui plus est, nous visons à automatiser ce processus, cet objectif soulève ainsi un ensemble de problèmes auxquels nous allons tenter de répondre.

3.2 Verrous

Un aspect central dans les domaines de l'informatique et du génie logiciel est le gain d'abstraction. Il est capital de permettre au concepteur de posséder des représentations partielles du système en devenir, en ignorant les parties qui ne sont pas pertinentes pour la tâche à réaliser. Si nous considérons l'ensemble de nos objectifs, il apparaît comme nécessaire de répondre aux questions suivantes :

– **Comment bénéficier de l'abstraction du système ?**
Permettre d'abstraire les systèmes complexes consiste à offrir les moyens d'en donner des représentations partielles mais précises, de manière à en faciliter la spécification.
– **Comment définir cette abstraction ?**
Il est nécessaire de décrire la manière de définir l'abstraction du système les étapes qui jalonnent l'obtention d'un modèle correct de ce système.
– **Quels outils utiliser ?**
Nous avons besoin de décrire des outils et des langages pour guider le concepteur, l'aider à se focaliser sur les points du système les plus importants.

– **Comment utiliser les résultats de cette abstraction ?**
Nous devons proposer des moyens pour concrétiser la description abstraite du système et automatiser cette concrétisation.

Dans la suite de ce document, nous nous attachons à donner une réponse à ces questions. Nous considérons, en effet, qu'y apporter des solutions nous permettrait d'atteindre notre objectif : fournir des outils et des méthodes pour assister le développement des systèmes complexes artificiels.

Nous avons considéré le gain d'abstraction et l'automatisation comme deux besoins primordiaux pour atteindre notre objectif. A ce titre, la notion d'Ingénierie Dirigée par les Modèles et les principes qu'elle propose nous apparaissent comme particulièrement adaptés. Les prochains chapitres ont pour objet de présenter les choix que nous avons faits pour satisfaire ces besoins.

Deuxième partie

État de l'art

Introduction

Dans cette partie, nous présentons un tour d'horizon de ce qui constitue l'état de l'art et le contexte spécifique de cette étude. Nous nous sommes concentrés sur les domaines et les théories pouvant nous permettre d'atteindre notre objectif.

Concevoir un système informatique est une activité complexe, elle nécessite de répartir et d'organiser précisément l'ensemble des tâches à réaliser. Elle implique également l'utilisation de moyens appropriés pour être menée à bien. Dans le premier chapitre de cette partie, nous nous intéressons au Génie Logiciel et plus particulièrement aux apports de l'Ingénierie Dirigée par les Modèles (IDM) dans ce vaste domaine. L'IDM offre des moyens théoriques et techniques pour faciliter le développement de systèmes grâce à l'utilisation de modèles abstraits.

Comme nous l'avons vu précédemment les SMA constituent une approche pour concevoir des systèmes informatiques complexes, qu'en est-il des méthodes de développement qui y sont associées, sont-elles adaptées à notre problématique ? Le chapitre suivant présente un état de l'art des méthodes de développement de systèmes multi-agents, en se restreignant à celles que l'on peut qualifier d'*« orientée modèle »* .

Si la notion de SMA est un premier pas vers la maîtrise de la complexité des systèmes, ils ne placent pas, dans la plupart des cas, la notion d'auto-adaptation au centre de leurs préoccupations. Nous étudions dans ce cadre le contexte de nos travaux avec deux approches tendant à placer l'adaptation comme un objectif principal. Tout d'abord, nous présentons la théorie des SMA Adaptatifs, qui propose de décrire la fonction globale du système par l'auto-organisation des fonctions locales remplies par des agents coopératifs. Nous décrivons ensuite le principe d'agent flexible qui introduit l'adaptation à un niveau différent (celui de l'agent) et selon des préoccupations différentes. Enfin, nous présenterons en détail la méthodologie ADELFE dédiée à la théorie des AMAS dans laquelle s'inscrit notre travail. Nous estimons également que son processus nécessite un certain nombre d'ajustements et d'extensions.

4 Génie logiciel et ingénierie des modèles

LE développement des systèmes informatiques tels que ceux présentés dans le chapitre 1 impose de mettre en œuvre des outils et méthodes permettant de diminuer la difficulté de cette tâche. Pour (Abbott, 2006), *« systems engineering is the engineering of complex systems »* . En effet, la multiplicité des échelles, les interactions diverses entre acteurs du développement, humain ou non, sont autant de propriétés faisant de l'ingénierie des systèmes informatiques une ingénierie complexe. Dans les premiers paragraphes de ce chapitre, nous nous intéressons aux réponses proposées par l'ingénierie logicielle « conventionnelle ». Nous en considérons principalement deux aspects. Le premier concerne la notion de processus de développement (point de vue méthodologique) et le second les aspects liés à l'outillage qui y est associé (modélisation principalement). Nous verrons que ces deux considérations sont étroitement liées et permettent de réduire la difficulté de la conception de logiciel.

Le deuxième paragraphe de ce chapitre présente le domaine de l'ingénierie dirigée par les modèles qui permet de mettre en œuvre des outils spécifiques de conception à un niveau d'abstraction élevé, celui des modèles. Il offre également des moyens de garantir des propriétés et d'automatiser une partie des tâches du concepteur.

4.1 Génie logiciel

Nos objectifs se placent dans le cadre du génie logiciel pour apporter des solutions au développement des systèmes complexes. Nous brossons dans ce paragraphe un rapide portrait de ce domaine, en nous attachant à deux aspects primordiaux pour notre travail, la notion de processus de développement et celle d'outil de modélisation.

4.1.1 Processus de développement

Le développement de logiciel est généralement organisé selon un processus qui, suivant le domaine (application web, logiciel embarqué, etc.) ou l'expérience « métier » des développeurs, implique différentes tâches et différents rôles. Cette notion de processus est au centre des méthodologies de développement et représente le cycle de vie du logiciel : de l'établissement du cahier des charges avec le client, jusqu'à la maintenance ou le retrait.

4.1.2 Cycles de vie logiciels

Depuis ses débuts, le génie logiciel a toujours été associé à la notion de cycle de vie, le logiciel étant un produit comme un autre, sa production pouvait être considérée de la même manière. Cependant, les cycles de vie populaires à l'aube de cette discipline (cascade, cycle en V) sont désormais inadaptés à la conception de logiciel dont la taille et le nombre de fonctionnalités ne font qu'augmenter. Comme nous l'avons vu dans le chapitre 1, les logiciels voués à se développer dans le futur franchissent encore un seuil dans cette complexité. Quoi qu'il en soit, le processus de développement logiciel UP (Unified Process) (Jacobson et al., 1999) représente une bonne approche pour appréhender la notion de cycle de vie et de processus de développement logiciel. Il s'agit d'un exemple des plus représentatifs et utilisés dans le monde Orienté Objet. Il a été implanté, étendu ou adapté pour de nombreux objectifs tant dans le milieu industriel qu'académique (OpenUp [1], Rational UP, Essential UP [2], etc.). C'est un processus incrémental et itératif. C'est-à-dire qu'il procède par répétition d'un ensemble de tâches vouées à fournir une version du produit final ; chaque répétition (itération) fournissant une version plus évoluée que ne l'était la précédente. L'aspect incrémental provient quant à lui, du fait que le logiciel est produit par partie (incrément) et intégré au fur et à mesure du développement. Le principe général du procédé est une succession de phases dont chacunes d'elles comprend plusieurs itérations ayant pour objectif de fournir une version d'une partie du produit final. Les objectifs de ces itérations dépendent de la phase dans laquelle elles apparaissent. De plus, le processus unifié définit un ensemble de disciplines (analyse et conception, déploiement, etc) ; chacunes d'elles intervient dans l'accomplissement des itérations. Toutes les disciplines n'interviennent pas aux mêmes phases du cycle de vie, celles liées à l'établissement des besoins (*Business modelling, Requirements*) interviennent essentiellement en phase préliminaire (*Inception*). Chaque discipline est définie

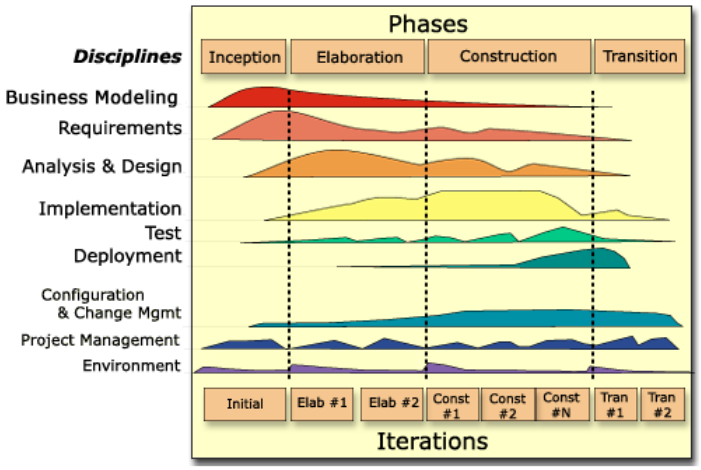

Figure 4.1 — Un exemple de mise en oeuvre du processus unifié : Rationnal Unified Process.

1. `http://www.eclipse.org/epf/general/OpenUP.pdf`
2. `http://www1.ivarjacobson.com/products/essup/`

comme un ensemble de tâches à réaliser dans un ordre précis et sous la responsabilité de rôles spécifiques. Cette organisation des tâches dans le temps mais également en termes de responsabilité permet de clarifier donc de faciliter, la production de logiciels. En outre, ces tâches sont assorties d'outils afin qu'elles puissent se dérouler dans les meilleures conditions et produire leur résultat dans les délais escomptés. L'intérêt de ces outils est présenté dans le paragraphe qui suit.

4.1.3 Outils de conception

Il est bien entendu impossible d'entamer le développement d'un logiciel en s'attachant directement au code. Il est nécessaire d'avoir des outils pour s'abstraire de ce code afin de réaliser des tâches préliminaires de spécification. Aussi, depuis de nombreuses années le développement logiciel s'est doté de moyens de spécification, de modélisation permettant par exemple, d'abstraire le schéma relationnel d'une base de données comme le modèle entités/association (EA) ou la définition des besoins du client (cas d'utilisation (Jacobson, 1995)). Ces outils sont associés et utilisés dans le cadre de méthodes spécifiques qui organisent leur utilisation tout au long du développement. Pour exemple, le modèle EA est mis en oeuvre dans la méthode MERISE(Rochfeld, 1986) pour donner une représentation abstraite des données d'un problème. De même, l'établissement des besoins dans le processus unifié peut être réalisé à l'aide de cas d'utilisation (Jacobson, 1995). Utiliser un outil adapté à la tâche que nous effectuons constitue un moyen de diminuer la complexité d'une tâche. Si cette constatation vaut pour l'ensemble des tâches de l'ingénierie, pourquoi ne pas en faire bon usage dans le monde informatique ? Un langage de modélisation ou de spécification permet d'abstraire un certain nombre de concepts spécifiques (par exemple les pré et post conditions d'un cas d'utilisation) qui n'auront plus à être pris en compte par leurs utilisateurs. C'est dans cette optique que Rumbaugh et al. (1999) définissent le langage de modélisation unifié UML et son utilisation dans le processus unifié (UP).

Pour résumer, il semble difficile d'envisager un processus ou une méthode de développement sans un formalisme associé (UML et UP, MERISE et le modèle EA, etc.). L'attrait d'un processus est de permettre un ordonnancement logique, connu des participants des tâches à réaliser. Quant à celui d'un formalisme adapté, il est de permettre l'abstraction de concepts liés au domaine d'expertise de la méthode (par exemple, le modèle EA pour représenter un schéma relationnel) facilitant la tâche correspondante dans le processus (par exemple, les cas d'utilisation pour la représentation des besoins).

Le génie logiciel offre des approches permettant de faire face à la difficulté de la conception et de la production de logiciel. D'un côté, grâce à une expression rigoureuse des tâches à réaliser, des responsabilités, des produits à fournir de l'autre, grâce à un gain d'abstraction dans la réalisation de certaines tâches comme la conception de l'architecture logicielle à l'aide d'un diagramme de classe, par exemple. La notion de modèle apparaît dès

lors comme un outil central pour le développement de logiciel. L'utilisation de modèles, pour assister réellement le développement de logiciel, ne peut pourtant pas être cantonner à un rôle descriptif. Toutes les étapes du cycle de vie peuvent bénéficier de modèles spécifiques qui constituent une couche d'abstraction suffisamment précise pour être automatiquement lier au code de l'application. C'est cette idée maîtresse qui a donnée naissance à la notion de développement dirigée par les modèles. Le chapitre suivant propose un tour d'horizon de cette notion et de ces intérêts dans le cadre de la conception de systèmes.

4.2 Ingénierie dirigée par les modèles

Depuis plusieurs années, l'Ingénierie Dirigée par les Modèles (IDM) est mise en œuvre dans de nombreux projets industriels et universitaires [3]. Il paraît évident qu'avec l'accroissement de la complexité des systèmes informatiques, l'explosion de leur utilisation dans des domaines aussi variés que la téléphonie mobile ou l'aéronautique, la tâche de concevoir un logiciel ait besoin d'abstraction et d'automatisation. L'intérêt de l'IDM réside particulièrement dans le fait qu'il offre un moyen d'abstraire un système par le biais de modèles, en simplifiant de ce fait, sa manipulation et son appréhension par les concepteurs. L'IDM apporte aussi une grande part d'automatisation des processus de développement, essentiellement grâce aux transformations de modèles. Nous présentons les principes de l'IDM et les différents aspects qui nous intéressent particulièrement dans le cadre de la définition d'un processus de développement dédié aux systèmes multi-agents adaptatifs.

4.2.1 Les principes

L'Ingénierie Dirigée par les Modèles (IDM) ne se limite pas à l'utilisation de modèles UML (omg, 2003d,c) dans les phases préliminaires du développement d'un logiciel. L'une des propositions de l'IDM est de ne pas cantonner les modèles à un rôle de spécification initiale, de représentation décorrélée du code final de l'application. Comme cela, l'IDM cherche à rendre ces modèles productifs et à maintenir leur utilité tout au long du cycle de vie (Favre et al., 2006). Selic dans sa présentation pour la conférence LMO 2008 (Langage et Modèle à Objet) [4] (Selic, 2008), avance même l'idée de modèles remplaçant complètement le code dans un futur plus ou moins proche.

4.2.1.1 Un premier exemple

Une première approche largement diffusée de ces principes est celle donnée par l'OMG : le MDA (Model Driven Architecture) (Mil, 2003). Elle consiste à mettre en œuvre le standard

3. `http://www.Eclipse.org/modeling/emft/`, `http://topcased.gforge.enseeiht.fr/`, Microsoft Software Factories, etc.

4. `http://lmo08.iro.umontreal.ca/Bran\%20Selic.pdf`

UML et un ensemble de transformations automatiques pour dériver le code d'une application. L'objet de ces transformations est de produire un modèle spécifique à la plate-forme d'accueil du logiciel (J2EE, .NET, etc.) à partir de modèles abstraits indépendants des technologies d'implantation CIM (Computation Independant Model) et PIM (Platform Independant Model) et d'un modèle représentant la technologie choisie PM (Platform Model). Ce modèle spécifique PSM (Platform Specific Model) est ensuite utilisé pour générer le code de l'application. Il ne s'agit là que d'une vision de l'IDM, celle de l'OMG, mais elle nous permet d'en mettre en évidence les idées majeures :

- faire des modèles les *« citoyens de première classe »* ,
- permettre un usage *productif* et non plus contemplatif de ces modèles,
- profiter au maximum des *traitements automatiques* de ces modèles.

Avant de rentrer davantage dans les détails de ces principes nous allons rapidement présenter les notions qui sont centrales dans une approche dirigée par les modèles.

4.2.1.2 Définitions

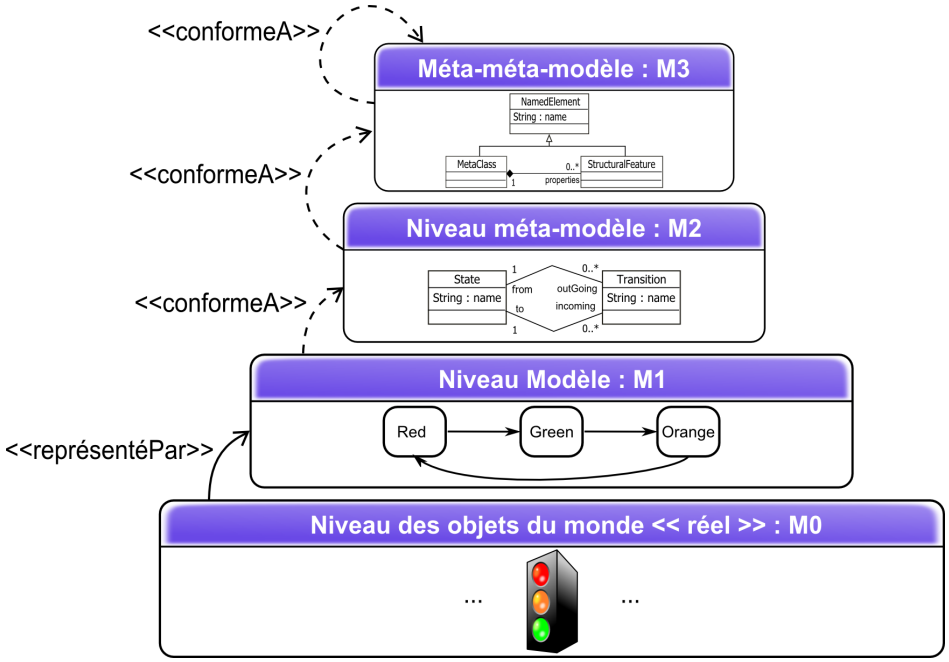

Figure 4.2 — Architecture de méta-modélisation (selon l'OMG (omg, 2003a))

La notion de modèle n'est pas spécifique à l'ingénierie qui en porte le nom. Par conséquent, il convient de replacer ce terme et ceux qui y sont étroitement liés, dans le contexte d'une démarche que nous qualifierons d'*« orientée modèle »* .

1. **Modèle** : est la simplification d'un système, décrite dans un but précis, et capable de répondre à des questions en lieu et place de celui-ci (Bézivin and Gerbé, 2001).
2. **Méta-modèle** : est un modèle dont l'objet est la représentation d'un langage de modélisation. L'ensemble des mots de ce langage sont des modèles conformes au méta-

modèle (Favre, 2004).

3. **Langage de modélisation** : est le langage grâce auquel nous définissons des modèles, un exemple des plus connus : UML.

4. **Méta-méta-modèle** : est un modèle dont l'objet est la représentation d'un langage de méta-modélisation tel que le MOF (omg, 2003a).

A travers ces définitions, nous considérons qu'un modèle *représente* un système du *« monde réel »* et qu'il se *conforme* à son méta-modèle. Un méta-modèle *représente* un langage de modélisation (Favre, 2004) dont les mots sont des modèles. Nous pouvons repositionner ces termes les uns par rapport aux autres sous la forme désormais classique de l'architecture de méta-modélisation pyramidale proposée par l'OMG dans (omg, 2003a). La figure 4.2 présente ces différents concepts comme appartenant aux niveaux suivants :

- **Niveau M0** : *« monde réel »* , de celui des instances (pour les objets) et des données.
- **Niveau M1** : correspond au niveau des modèles (cf. définition ci-dessus).
- **Niveau M2** : niveau des méta-modèles décrivant des langages de modélisation utilisés pour définir des modèles.
- **Niveau M3** : le dernier niveau, celui du méta-méta-modèle, décrit un langage de méta-modélisation. Il définit des méta-modèles, donc représente des langages de modélisation. Le méta-méta-modèle est *méta-circulaire*, c'est-à-dire qu'il peut être décrit en ses propres termes.

Favre et al. (2006) définissent comme *« espace technique »* une telle pyramide. Un *espace technique* dépend du méta-méta-modèle placé à son sommet (niveau M3) comme par exemple l'EBNF (Extended Backus-Naur Form) dans le cas des grammaires. C'est une vision particulière de l'IDM où tout est considéré comme modèle. Dans cette vison, la génération de code peut être considérée comme un pont entre deux *espaces techniques* mise en œuvre grâce à une transformation.

4.2.1.3 Des modèles au centre de la conception

Si nous examinons les définitions précédentes, nous pouvons envisager la notion de modèle comme unificatrice. Dans une approche dirigée par les modèles tout, ou presque, peut être considéré comme un modèle et appartient donc à un *espace technique* particulier (une pyramide). C'est du moins la vision de l'ingénierie dirigée par les modèles qui est proposée par Bézivin (Bézivin, 2006). Quelle que soit l'approche choisie, MDA pour l'OMG, Software Factories pour Microsoft ou Eclipse Modeling Framework (Budinsky et al., 2003), les principes sont identiques à peu de chose près. La notion de modèle y est toujours placée au centre et constitue la part la plus importante du développement logiciel. L'automatisation apparaît aussi comme indispensable pour le traitement de ces modèles (traduction, intégration, composition, génération de code, ré-ingénierie, etc.). Il s'agit d'un glissement par rapport aux approches centrées sur le code vers un niveau d'abstraction plus élevé.

4.2.2 Les langages de modélisation

Pour définir des modèles, il est nécessaire de posséder des langages adaptés. Nous distinguons deux familles de langages de modélisation, les spécifiques (Domain Specific Modeling Languages) et les génériques (General Purpose Modeling Language). Ces derniers sont particulièrement incarnés par le standard de l'OMG : UML. De nombreux auteurs (Favre et al., 2006; France and Rumpe, 2005; Bézivin and Gerbé, 2001; Clark et al., 2008) considèrent néanmoins qu'il est préférable d'organiser une approche dirigée par les modèles autour de langages spécifiques précis et adaptés aux besoins du domaine applicatif ainsi qu'à ceux des utilisateurs. Ils estiment, en effet, qu'il est difficile d'envisager la vision d'un langage unique, *« monolithique »* qui soit utilisable dans tout les cas (*« one size can not fit all »*). Clark et al. (2008) définissent la notion de méta-modélisation comme un moyen de décrire des langages au sens large : langages de modélisation ou de programmation, génériques ou spécifiques. En ce qui nous concerne, nous nous pencherons davantage sur l'expression de langages de modélisation spécifiques ou DSML et à leur définition au travers d'un processus de méta-modélisation.

4.2.2.1 Syntaxes

Comme tout langage, un DSML possède une syntaxe qui permet de décrire la structure des mots qui le constituent ou, en l'occurrence, des modèles. Nous présentons ici les deux aspects de cette syntaxe indispensable à la création d'un nouveau langage, de modélisation ou non (Clark et al., 2008).

1. **Syntaxe abstraite**
   Selon la définition que nous avons considérée précédemment, un modèle est défini dans une intention précise, il en est de même pour un méta-modèle. Bien que pour certains auteurs un méta-modèle doit permettre de décrire tous les aspects d'un langage aussi bien syntaxiques que sémantiques (Clark et al., 2008), dans la pratique, il s'agit dans la plupart des cas d'un modèle n'en représentant que la *syntaxe abstraite*. Par *syntaxe abstraite*, nous entendons l'ensemble des concepts d'un langage et des relations qui les lient. Les langages de méta-modélisation, tels que le standard de l'OMG MOF omg (2003a), offrent les concepts et relations élémentaires en termes desquels il est possible de décrire un méta-modèle représentant cette syntaxe abstraite. Le méta-modèle possède de plus un certain nombre de concepts (ou méta-classes) qui ne sont pas directement liés au domaine mais davantage à la représentation du langage et aux informations que l'utilisateur du langage devra fournir. Prenons l'exemple de l'UML, le concept de *Lifeline* n'est pas directement lié au monde Objet, en revanche il a tout son intérêt dès lors que nous souhaitons donner les moyens à un concepteur d'applications de représenter des séquences d'interactions entre objets. Définir un langage de modélisation dédié nécessite donc de prendre en compte ces aspects syntaxiques au sein du méta-modèle qui le représente. Nous disposons, à ce jour, pour décrire cette syntaxe de nombreux environnement et langages de méta-modélisation : Eclipse-EMF/Ecore (Budinsky et al.,

2003), GME/MetaGME (Ledeczi et al., 2001), AMMA/KM3 (ATLAS, 2005) ou XMF-Mosaic/Xcore (Clark et al., 2008). La *syntaxe abstraite* d'un langage fixe ses concepts ; il est cependant nécessaire d'en fournir une représentation afin de pouvoir le rendre utilisable. Ces aspects visuels d'un langage sont regroupés au sein de sa *syntaxe concrète*.

2. **Syntaxe concrète**
   La notion de « Langage de modélisation » est souvent associée avec celle de « représentation graphique » . Toutefois, nous pouvons envisager des langages de modélisation textuels tels que KM3 [5] (Jouault and Bézivin, 2006) ou Kermeta [6] (Muller et al., 2005a). La syntaxe concrète d'un langage fournit un formalisme, graphique ou textuel, pour manipuler les concepts de la syntaxe abstraite en créant des « instances ». Le modèle obtenu sera conforme au méta-modèle : à la syntaxe abstraite. La définition d'une syntaxe concrète est, à ce jour, relativement bien maîtrisée et outillée. Il existe, en effet, de nombreux projets principalement basés sur le plugin EMF (Eclipse Modeling Framework) de la plate-forme Eclipse qui s'y consacrent : GMF [7], Merlin Generator [8], GEMS [9], TIGER Ehrig et al. (2005), etc. Ils permettent de dissocier le méta-modèle (la syntaxe abstraite) de sa ou ses syntaxes concrètes, elles-mêmes décrites par des modèles. D'un point de vue pragmatique, certains outils utilisent cette vision pour permettre la génération automatique d'éditeur graphique. GMF dissocie par exemple, les *« concepts du domaine »* (syntaxe abstraite), de leurs représentations graphiques ou des actions qui y seront associées au sein de l'éditeur graphique généré. Chacune de ces préoccupations est décrite au sein d'un modèle (*domain, mapping, graphical, tooling*). D'autres outils utilisent la même démarche pour produire des syntaxes concrètes graphiques TOPCASED (Farail et al., 2006) ou textuelles Sintaks (Muller et al., 2006), TCS (Jouault et al., 2006), etc.

4.2.2.2 Sémantique

Au même titre que la syntaxe, il est nécessaire de définir les règles d'usage d'un langage. L'expression de la sémantique des langages de programmation est bien connue, pourtant il est souvent reproché aux langages de modélisation leur manque de sémantique « formelle ». C'est le cas pour UML dont la sémantique n'est souvent définie qu'en langage naturel dans les documents de spécification. Elle est donc potentiellement ambiguë et soumise à interprétation. Néanmoins, il existe des moyens d'exprimer une sémantique *« semi-formelle »* pour les langages de modélisation en utilisant les outils que l'IDM met à notre disposition. Dans le cadre du projet TOPCASED, nous avons étudié les méthodes d'expression de la sémantique des langages de programmation et les avons transposées dans le monde orientée modèle. Nous avons pu identifier plusieurs approches pouvant être envisagées en utilisant les

5. http ://wiki.Eclipse.org/index.php/KM3
6. http ://www.kermeta.org/
7. Generic Modeling Framework, `http://www.Eclipse.org/modeling/gmf/`
8. `http://sourceforge.net/projects/merlingenerator/`
9. Generic Eclipse Modeling System, `http://sourceforge.net/projects/gems/`

moyens que l'IDM met à notre disposition. Les résultats de cette étude sont présentés dans (Combemale et al., 2006) et sont résumés dans les paragraphes suivants.

1. **Structurelle**
   La sémantique axiomatique ou structurelle des langages de modélisation peut s'exprimer sous la forme de règles de bonne formation. Elles permettent de déterminer si des constructions appartiennent effectivement à un langage et de restreindre l'ensemble des mots valides pour celui-ci. Il s'agit de vérifier des propriétés structurelles qui ne peuvent être garanties par la syntaxe abstraite. Les langages de méta-modélisation tels que le MOF ou Ecore, son émanation dans le monde Eclipse (Budinsky et al., 2003), permettent d'exprimer une première forme de cette sémantique par le biais de multiplicités. Celles-ci permettent, par exemple, d'exprimer qu'un concept ne peut être présent qu'une seule fois dans un modèle. Cependant, ce type de contraintes reste limité ; l'OMG a proposé au travers de son langage OCL (Object Constraint Language) (omg, 2003b) le moyen d'exprimer des règles plus complexes sur les modèles et d'en vérifier la validité. Ce langage permet de manipuler des ensembles et des expressions de logique équivalentes à la logique des prédicats tout en exprimant la navigation au sein d'un modèle. OCL est utilisable tant au niveau modèle que méta-modèle. De cette façon, il exprime des règles complexes de bonne formation au niveau d'un méta-modèle pour en vérifier le respect sur les modèles qui s'y conforment.
2. **Dynamique**
   L'objectif de la sémantique dynamique est d'exprimer ce que doit être l'évaluation d'une expression d'un langage vis-à-vis d'une exécution sur une machine abstraite. Tous les langages de modélisation ne sont pas voués à être exécutés ou n'expriment pas de dynamique (cf. diagrammes de classes UML). A ce niveau nous pouvons distinguer deux approches :
   – *Dénotationnelle* :

     Le principe de la sémantique dénotationnelle est de s'appuyer sur un formalisme rigoureusement (i.e. mathématiquement) défini pour exprimer la sémantique d'un langage donné. Une traduction est alors réalisée des concepts du langage d'origine vers ce formalisme. C'est cette traduction qui donne la sémantique du langage d'origine. Dans le cadre de l'IDM, il s'agit d'exprimer des transformations vers un autre langage ou formalisme sémantiquement bien défini. Nous parlons également de sémantique de traduction (Clark et al., 2008). Ces transformations permettent alors d'appuyer un langage de modélisation sur des formalismes pouvant offrir des outils de simulation, de vérification ou d'exécution.

   – *Opérationnelle* :

     La sémantique opérationnelle permet de décrire le comportement dynamique des constructions d'un langage. Dans le cadre de l'IDM, elle vise à exprimer la

sémantique comportementale d'un méta-modèle afin de permettre l'exécution, la simulation des modèles qui lui sont conformes. Pour exprimer la sémantique opérationnelle, nous pouvons envisager des approches différentes comme la définition du comportement de chaque concept de manière impérative à l'aide d'un langage de méta-programmation (Kermeta (Muller et al., 2005a), xOCL (Clark et al., 2008)). Une autre approche est de définir des règles de réduction sur le méta-modèle, comme cela a été réalisé dans le cadre des langages de programmation (Structural Operational Semantics Plotkin (1981), sémantique naturelle Kahn (1987)). Cette approche représente l'évaluation d'une expression sous la forme d'une règle de transformation qui réduit l'expression (le modèle) jusqu'à obtenir une expression irréductible, une valeur.

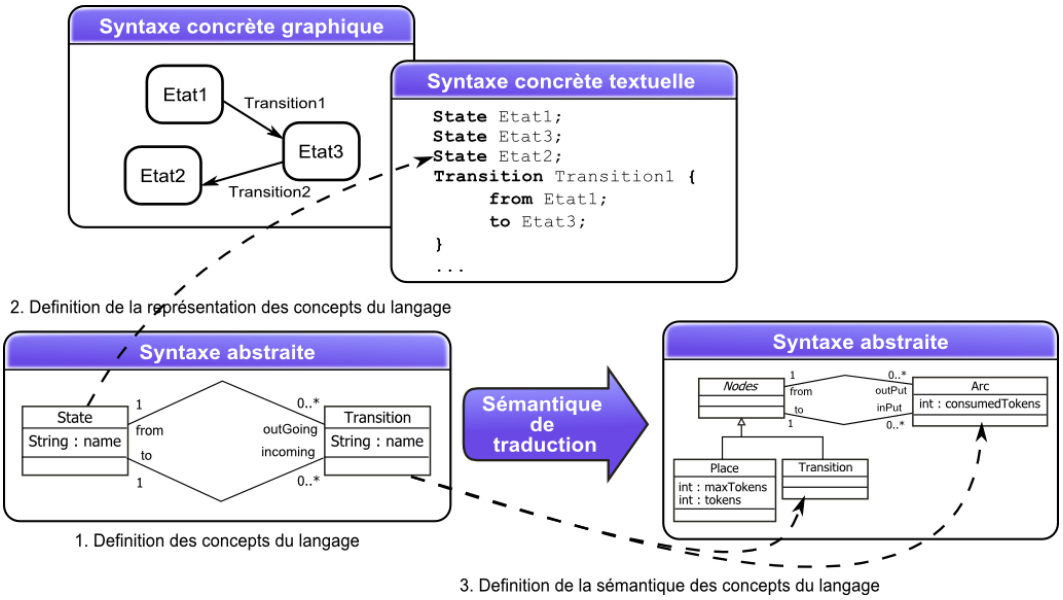

Figure 4.3 — Résumé des principes de définition d'un langage de modélisation spécifique (DSML).

L'IDM place au centre du développement les modèles et par conséquent la notion de langage de modélisation qui en supporte la définition. Nous pouvons décrire des modèles comme l'abstraction de systèmes complexes sous différents points de vue. L'IDM nous offre de surcroît, les moyens de décrire nos propres langages, dédiés à des domaines spécifiques, au travers d'un processus de méta-modélisation (définition de la syntaxe abstraite, concrète et de la sémantique voir figure 4.3). Le premier objectif de l'IDM est ainsi rempli, il permet grâce à des langages qui capturent les concepts clefs de domaines technologiques de s'en abstraire. Le second objectif, l'automatisation, peut être atteint grâce à des transformations de modèles, celles-ci seront présentées dans le paragraphe suivant.

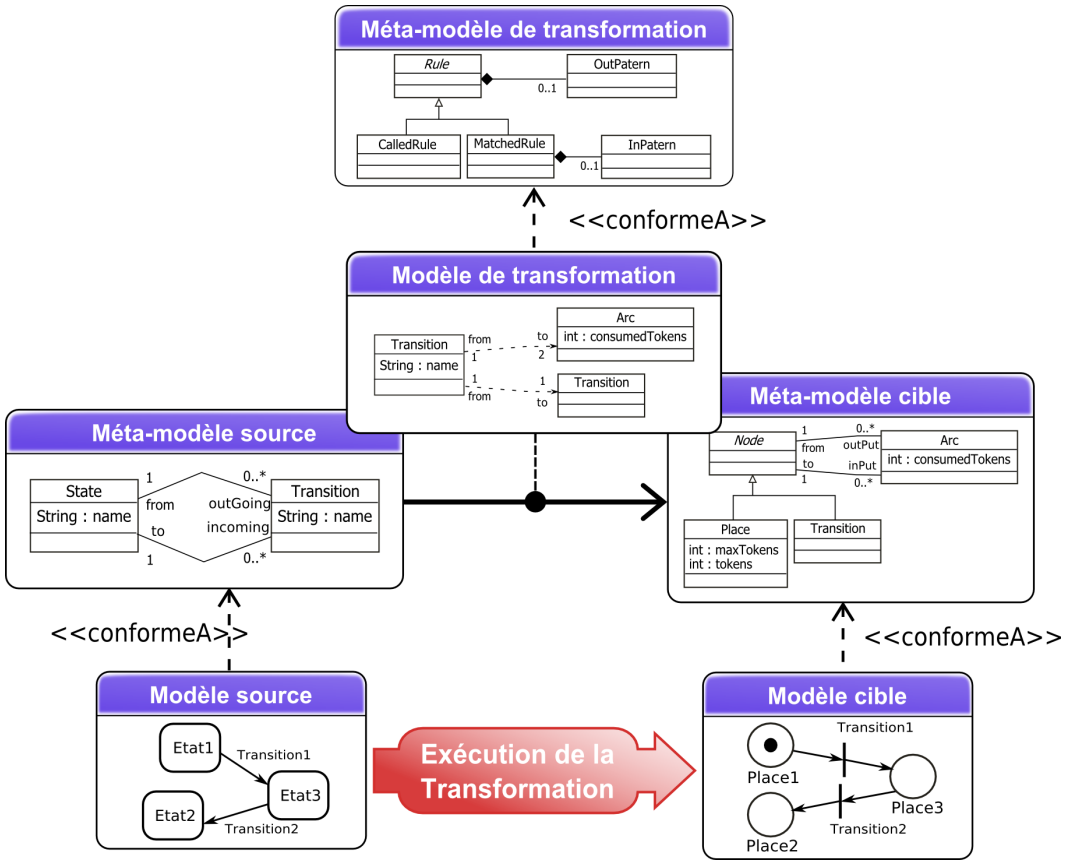

Figure 4.4 — Transformation de modèles.

4.2.3 Transformation : un outil de premier ordre

L'IDM promeut le traitement automatique de modèles comme un moyen de faciliter la production de code, de maintenir la cohérence entre celui-ci et le modèle qui le représente, de passer d'un espace technique à un autre, etc (Favre et al., 2006). L'outil principal au centre de ces traitements automatiques est la transformation de modèles. Une transformation est définie entre plusieurs méta-modèles, potentiellement différents. Dans le cas le plus général, celui représenté par la figure 4.4, une transformation spécifie la traduction d'un modèle (conforme à un méta-modèle source) en un autre modèle (conforme à un méta-modèle cible). Elle est exprimée grâce à un langage de transformation. Comme l'a promu l'OMG dans sa spécification (omg, 2005a) cette transformation est elle-même un modèle conforme à un méta-modèle de transformation (voir figure 4.4). L'intérêt des transformations est, entre autres, de factoriser les connaissances d'experts de différents domaines afin d'automatiser des tâches de traduction ou d'intégration de modèles qui seraient extrêmement fastidieuses. Nous présentons ici quelques exemples des capacités des transformations de modèles sans tenir compte des technologies ou des langages qui pourraient les supporter. Le choix d'un langage de transformation dépend essentiellement de l'objectif de la transformation. (Mens and Gorp, 2006) présentent une taxonomie des transformations et différents critères afin de déterminer les propriétés qu'un langage doit posséder pour les mener à bien.

4.2.3.1 Établissement de ponts technologiques

Il s'agit ici d'une transformation dont les méta-modèles cible et source n'appartiennent pas au même espace technique. C'est le cas d'une génération de code pour laquelle à partir d'un modèle conforme à un méta-modèle décrit par le MOF par exemple, une portion de code est générée respectant la syntaxe de la grammaire du langage cible exprimé grâce à l'EBNF.

4.2.3.2 Vérification de modèle

Les langages de transformation proposent pour la plupart des expressions permettant la navigation au sein d'un modèle à l'image d'OCL (omg, 2003b). Nous pouvons donc parcourir le modèle et vérifier sa validité pour générer le cas échéant un rapport d'erreur ou un modèle plus riche qu'un simple booléen (Bézivin and Jouault, 2005).

4.2.3.3 Re-factorisation

Dans ce cas, les transformations peuvent être utilisées pour améliorer la qualité d'un modèle. Par exemple, en y incluant des patrons de conceptions, en y appliquant des métriques pour en évaluer la qualité (Monperrus et al., 2008), corriger des erreurs de conception, etc. Pour ce type de traitement, le modèle source est aussi la cible.

4.2.3.4 Expression de la sémantique

Dans les cas d'une sémantique de traduction ou dénotationnelle, il s'agit d'exprimer un pont technologique vers un espace technique dont la sémantique est connue et définie formellement comme par exemple, les réseaux de pétri. Il est aussi envisageable de décrire la sémantique d'un modèle comme une re-factorisation, dont l'objet serait d'évaluer le résultat d'une exécution du modèle (cf. paragraphe 4.2.2.2).

La liste des applications potentielles ou effectives des transformations de modèles pourrait encore être étendue. Cependant, ces quelques exemples permettent de concevoir l'utilité de celles-ci dans une approche orientée modèle. Elles permettent en effet d'améliorer le qualité des modèles de les rendre éventuellement exécutables, de générer du code, etc. De plus, ce domaine est en plein essor depuis quelques années, de nombreuses conférences et ateliers s'y consacrent (MTIP, GRAMOT, etc.) et de nombreuses solutions sont d'ores et déjà disponibles que ce soient des langages de transformation de modèles déclaratifs (Jouault and Kurtev, 2005), impératifs (Muller et al., 2005b), des langages de *template* [10] ou autre.

10. Java Emitter Template (JET) `http://www.Eclipse.org/modeling/m2t/?project=jet`, Acceleo `http://www.acceleo.org/pages/accueil/fr`

4.3 Synthèse et analyse

L'IDM promeut des principes incarnés par de nombreux langages et outils, à l'image de ceux cités dans ce chapitre. Adopter une démarche dirigée par les modèles permet de profiter des caractéristiques des modèles :

- ils constituent un niveau d'*abstraction* plus élevé que le code du système qu'il représentent,
- ils proposent des concepts spécifiques et précis pour la conception,
- ils sont définis dans un cadre strict : ils se conforment à leur méta-modèle.

Ces caractéristiques sont de plus adaptables à n'importe qu'elle domaine. Il est possible de proposer de nouveaux méta-modèles afin de définir des *langages spécifiques* et adaptés aux besoins d'un domaine particulier. Ces langages constituent un moyen de décrire un problèmes en utilisant les concepts essentielles du domaine considéré au sein de modèles. Au delà des aspects liés à la définition de modèles et de méta-modèles, l'IDM promeut l'utilisation de *traitements automatiques* :

- ils permettent d'améliorer la qualité des modèles (cf. paragraphe 4.2.3),
- ils peuvent regrouper et automatiser des savoir-faire et des bonnes pratiques,
- ils possèdent un support d'implémentation sous la forme de *transformations de modèles*,
- ils permettent la génération automatiques de code.

En raison de ces propriétés, principalement l'automatisation et le gain en abstraction (Selic, 2008), l'IDM est à présent largement répandue et permet de faire face pour une grande part à la difficulté du développement de logiciels qui peut s'avérer extrêmement complexe (Abbott, 2006). Néanmoins, le potentiel de cette approche n'est pas encore totalement exploité (Selic, 2007). Pour notre part, nous souhaitons bénéficier des intérêts de cette approche pour formaliser les aspects spécifiques de la théorie des AMAS, en proposant un méta-modèle qui permettra de regrouper et de préciser le concept d'agent coopératif et son comportement. En d'autres termes, nous voulons permettre la définition de modèles et leurs traitements automatiques par le biais de transformations, en vue de produire un logiciel auto-adaptatif capable de satisfaire aux contraintes des systèmes complexes artificiels.

Comme nous l'avons vu dans le paragraphe initial de ce chapitre, posséder des outils spécifiques est une chose, savoir les utiliser à bon escient en est une autre. Pourtant ces deux points sont indissociables pour atteindre notre objectif. Aussi, nous avons étudié les différentes méthodes du monde SMA mettant en œuvre une démarche de méta-modélisation, ou proposant des approches génératives. Le chapitre suivant présente le résultat de cette étude.

5 Ingénierie des systèmes multi-agents

LE développement des systèmes multi-agents est une tâche complexe. Contrairement au paradigme objets il n'existe pas réellement d'approche, de processus ni même de langage qui fasse l'unanimité. Depuis plusieurs années, de nombreuses équipes s'attachent à investir le domaine du génie logiciel orienté agent (Luck et al., 2004). Ce chapitre, présente certaines initiatives que nous considérons avec un intérêt particulier du fait de la proximité de leurs objectifs. Quoi qu'il en soit, nous considérons la théorie des AMAS comme un domaine en soi. C'est pour cette raison que notre équipe a mené des travaux dans la voie du génie logiciel sous la forme d'une méthodologie spécifique intégrée à un atelier de développement (ADELFE). Nous constatons cependant qu'afin d'atteindre les objectifs que nous nous sommes fixés dans les chapitres précédents de nombreuses tâches restent encore à être accomplies. Nous présentons ici un rapide tour d'horizon des méthodologies qui possèdent de près ou de loin, des objectifs, des mises en œuvre proches des nôtres. Il ne s'agit pas d'une liste exhaustive mais elle présente essentiellement des travaux dont l'approche peut être qualifiée d'*« orientée modèle »* .

5.1 Ingenias

Ingenias offre à la fois une méthodologie et un outil de développement complet et extensible : IDK (Ingenias Development Kit [1]) (Gomez-Sanz and Pavon, 2003). Une partie de l'IDK s'appuie sur des principes liés à l'IDM notamment en terme de l'utilisation de méta-modèles spécifiques définis par spécialisation du méta-modèle UML (stéréotypes). De plus, l'IDK maintient un lien permanent entre le code et les modèles qu'il permet de définir. Par exemple, il est possible de positionner des points d'arrêts au niveau du modèle qui seront pris en compte à l'exécution.

Ingenias définit cinq perspectives ou points de vue et par là même, cinq méta-modèles : agents, organisation, buts/tâches, interactions et environnement.

1. **Perspective agent** : Présente les aspects spécifiques à l'agent : état mental, responsabilité et capacité. L'état mental d'un agent est constitué d'entités mentales telles que les buts, les croyances, les faits et les compromis. Cet état est géré par un Manager et est maintenu et utilisé pour déterminer les actions à mener par le processeur d'état mental. Les états mentaux sont exprimés en termes d'entités tel que celles proposées par (Shoham, 1996) et l'approche BDI (Belief Desire Intention) (Kinny et al., 1996).

1. `http://ingenias.sourceforge.net/`

2. **Perspective buts/tâches** : Propose une décomposition des tâches et une justification de celles-ci au travers de leur implication ou non dans la satisfaction d'un but (relation de satisfaction/échec). Les tâches peuvent regrouper plusieurs activités pouvant être liées au maintien de l'état mental. Elles utilisent des ressources et sont *« déclenchées »* par des pré-conditions exprimées en termes d'entités mentales (décrites dans le point de vue agent). En d'autres termes et pour simplifier les choses, une tâche consomme, produit des faits ou satisfait un but poursuivi par l'agent qui les réalisent.
3. **Perspective organisation** : Cette perspective représente l'organisation structurelle, l'architecture du système. Cette organisation est basée sur le modèle AGR (Agent Groupe Role) mis en œuvre dans AALAADIN (Ferber and Gutknecht, 1998). Les aspects dynamiques de cette organisation sont exprimés sous forme de workflows, représentant les relations entre tâches, ressources et rôles par le biais de responsabilités. De plus, il est possible de décomposer une organisation en groupes d'agents, pouvant à leur tour définir rôles, ressources etc. Sur cette structure, il est possible, de surcroît, de venir plaquer des relations dites sociales.
4. **Perspective environnement** : Ce méta-modèle décrit le diagramme d'environnement et utilise la notation TAEMS pour représenter les ressources les applications et les agents dans l'environnement. Les applications sont considérées comme des sortes d'objets encapsulant tout ce qui n'est ni ressource ni agent. Cela permet de définir les interactions entre ces « objets »,ressources et les agents du système.
5. **Perspective interaction** : Elle a pour objet la description des interactions entre agents, ainsi qu'entre agent et élément de l'environnement. Elles peuvent être définies à l'aide de plusieurs langages tels que les diagrammes de collaboration UML, diagrammes d'interaction GRASIA [2] ou des diagrammes de protocole AUML. Une interaction a un objectif qui doit être partagé ou partiellement poursuivi par les participants, elle est normalement liée aux objectifs d'une organisation.

Ingenias a été utilisé pour plusieurs projets et possède un outillage complet, centré sur la notion de modèle. Pourtant, il ne met pas nécessairement en avant la nécessité d'adaptation aux environnements complexes, ni le principe d'émergence par auto-organisation puisqu'un diagramme est dévolu à la description de cette organisation. De plus, la coopération n'est pas établie comme un principe de base. Les problématiques qu'il aborde sont toutefois très proches des notres, autant dans les objectifs que dans la réalisation.

5.2 Tropos

Tropos [3] est une méthode de développement orientée agent (Bresciani et al., 2004) mettant particulièrement l'accent sur l'expression des besoins. Elle se décompose en cinq phases :

2. `http://ingenias.sourceforge.net/`
3. `http://www.troposproject.org/`

1. **Besoins préliminaires** : fournit une description du domaine d'application.
2. **Besoins finals** : permet d'introduire le système en devenir au sein de son domaine et d'analyser les relations de celui-ci avec son environnement.
3. **Architectural Design** : décrit l'architecture interne du système en représentant ses composants et les relations qui les lient.
4. **Conception détaillée** : définit de manière précise les capacités des agents du système et de leurs interactions.
5. **Implémentation** : fournit le code de l'application en fonction de la conception détaillée et de la plate-forme cible choisie.

A cette méthode est associé un outil développé sur la base des plugins EMF (Eclipse Modeling Framework) et GEF (Graphical Modeling Framework) d'Eclipse et appelé TAOM4E (Tool for Agent Oriented visual Modeling for the Eclipse platform) (Bertolini et al., 2006). Tropos adopte un langage de modélisation orienté agent qui fut tout d'abord mis en œuvre dans l'outil TAOM. TAOM4E introduit une approche dirigée par les modèles en se basant sur les principes proposés par le MDA de l'OMG (cf. paragraphe 4.2.1.1). (Perini and Susi, 2005) présente les différents types de transformations mises en œuvre dans le cadre de Tropos, pour le raffinement des modèles préliminaires et pour la traduction de modèles Tropos en modèle UML2.0. TAOM4E propose un outil additionnel [4] de génération de code permettant de produire lors de la phase d'implantation du code pour la plate-forme d'agent BDI JadeX [5] (Penserini et al., 2007).

5.3 PASSI

PASSI (a Process for Agent Societies Specification and Implementation) est une méthodologie de développement étape par étape dédiée à la conception de systèmes multi-agents, qui couvre le cycle de vie à partir de l'établissement des exigences préliminaires jusqu'à l'implémentation. Elle est composée de cinq modèles basés sur la notation UML et de douze phases impliquées dans la définition de ces derniers (cf. figure 5.1). Elle propose un outil appelé PTK (Passi Tool Kit) qui est composé d'un add-in pour Rational Rose et d'un outil supplémentaire pour la réutilisation d'agents. PASSI considère un agent comme une entité du niveau abstrait (classe d'agent) et du niveau concret (instance d'agent). C'est pourquoi ils sont présents au sein des cinq modèles :

1. **Modèle des besoins système** : Ce modèle permet la description des exigences fonctionnelles du système et est principalement axé sur la notion de cas d'utilisation, leur description et leur réalisation en termes de tâches. Les fonctionnalités du système sont représentées par des cas d'utilisation (Domain Description) et leurs responsabilités sont réparties parmi les agents (Agent Identification). Le rôle associé à chaque responsabilité

4. http://se.itc.it/morandini/home.html
5. http://vsis-www.informatik.uni-hamburg.de/projects/jadex/

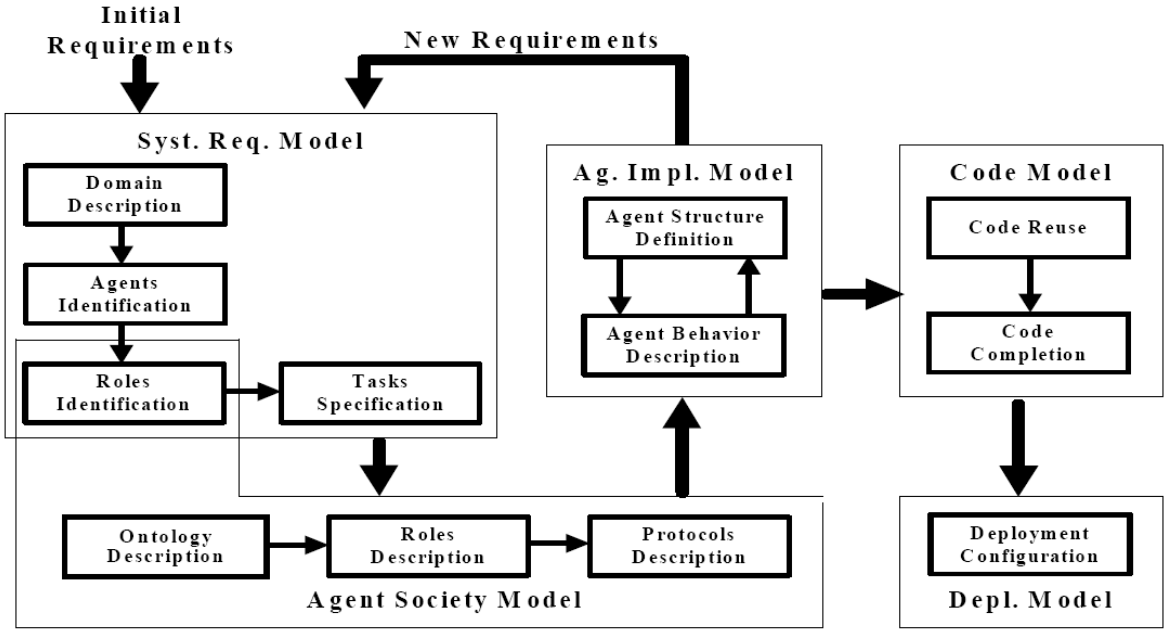

Figure 5.1 — Résumé des différentes étapes de PASSI

est détaillé par un diagramme de séquence (Role Identification) puis les capacités des agents sont exprimé au travers de tâches et d'interactions dans un diagramme d'activité (Task Specification).

2. **Modèle de la société d'agents** : Ce modèle est dédié à la définition des concepts du domaine manipulé par les agents du système et à celle de leurs communications et de la séquence des rôles qu'ils jouent. La première étape consiste à décrire les connaissances de chaque agent grâce à un diagramme de classe (Ontology Description). Les rôles sont ensuite détaillés en termes des tâches, des communications et des dépendances qu'ils impliquent (Role Description). Des diagrammes de séquence sont ensuite utilisés pour décrire les différents protocoles de communications (Protocol Description).

3. **Modèle d'implémentation des agents** : Ce modèle représente l'architecture de la solution choisie en termes de classes et de méthodes. On y décrit tout d'abord l'architecture des agents sous forme de diagramme de classes (Agent Structure Definition) puis leurs comportements grâce à des diagrammes d'activité ou état transition (Agent Behavior Description).

4. **Modèle du code** : il s'agit du code de la solution établie. Il implique la mise en œuvre de bibliothèques pour réutiliser du code (Code Reuse Library), ou bien son implantation (Code Completion Baseline).

5. **Modèle de déploiement** : ce modèle permet de représenter la distribution des agents sur les différentes machines disponibles. Il utilise non seulement les diagrammes de déploiement pour représenter la disposition des agents sur les différentes unités mais également des contraintes sur leur mobilité éventuelle (Deployment Configuration).

PASSI est un processus complet et détaillé ; en outre, il est profondément ancré dans les standards tels que l'UML et les spécifications FIPA (AIP, et FIPA OS). Il se focalise principalement sur la fonctionnalité du système à travers la définition de tâches et l'assignation de rôles, les aptitudes sociales et l'adaptation ne constituent pas la pièce maîtresse du processus de conception. (Chella et al., 2006) présente les évolutions de la méthodologie PASSI qui ont permis d'y associer les principes issus de la programmation agile. Le processus a été adapté pour intégrer une phase de tests beaucoup plus importante. De surcroît, (Cossentino et al., 2008) définit une nouvelle approche pour le développement de SMA basée sur la méthodologie PASSI et sur la notion de simulation.

5.4 MetaDIMA

DIMA (Développement et Implémentation de Systèmes Multi-Agents) [6] est un environnement de développement de systèmes multi-agents. DIMA a été utilisé pour développer plusieurs applications réelles (Ventilation artificielle, simulation de modèles économiques, etc.) ; ces applications peuvent être des simulations, des résolutions de problèmes ou des systèmes de contrôles ayant éventuellement des contraintes temps réel. Pour présenter cet environnement, il est important de présenter le concept de système et d'agent sur lequel il repose.

1. **Modèle d'agent** : L'architecture d'agents DIMA propose de décomposer chaque agent en différents modules dont le but est d'intégrer des paradigmes existants, notamment des paradigmes d'intelligence artificielle. Ces modules représentent les différents comportements d'un agent tels que la perception (interaction agent-environnement), la communication (interaction agent-agent) et la délibération. Un agent peut avoir un ou plusieurs modules qui peuvent être réactifs (décrits par un ensemble de méthodes au sens objets) ou cognitifs (possèdent mécanismes de raisonnement, par exemple, un moteur d'inférence). Pour gérer les interactions entre ces différents comportements, DIMA propose un module de supervision représentant le méta-comportement de l'agent. Ce méta-comportement réifie le mécanisme de contrôle de l'agent, il gère les interactions entre les différents modules et permet à l'agent d'observer ces comportements. Un agent est une entité pro-active et autonome. Des agents hybrides peuvent être conçus pour allier des capacités réactives à des capacités cognitives, ce qui leur permet d'adapter leur comportement en temps réel à l'évolution de leur univers.
2. **Modèle du système** : Un système multi-agent peut comporter un nombre quelconque d'agents hétérogènes (tailles différentes, mécanismes de décision/raisonnement différents, etc.). Des agents peuvent être créés dynamiquement ou peuvent disparaître parce qu'ils ont atteint leur but ou qu'ils n'ont plus de ressource. Ces agents sont également dotés de mécanismes pour adapter leurs comportements aux changements de leur univers. Un agent peut posséder une base de règles figées mais il peut également l'acquérir

6. http://www-poleia.lip6.fr/~guessoum/dima.html

automatiquement en se basant sur sa mémoire, en utilisant un raisonnement à base de cas par exemple. D'autre part, pour optimiser le contrôle un méta-comportement utilisant les algorithmes génétiques a été introduit.

3. **Environnement de développement** : La principale caractéristique de DIMA est son architecture modulaire et des librairies offrant les briques de base pour construire des modèles d'agents divers. Ces différentes briques ont pour but d'offrir à l'utilisateur d'une part, une grande variété de paradigmes (par exemple automate, règle, etc.), d'autre part, une implémentation des différentes propositions conceptuelles introduites par la communauté multi-agents (BDI, KQML, ACL, etc.).
4. **Environnement d'exécution** : Une version de DIMA fondée sur la méta-modélisation a été développée : MétaDIMA (Jarraya and Guessoum, 2007). Elle introduit une approche MDA pour une implantation quasi automatique sur une plate-forme donnée. L'objectif est d'offrir à un utilisateur qui ne soit pas nécessairement familier des systèmes multi-agents et des langages de programmation tel que JAVA, un outil pour décrire son système. Cette description est ensuite transformée en un système multi-agents. Ce travail est basé sur l'outil MétaGen.

5.5 ASPECS

ASPECS [7] (Agent-oriented Software Process for Engineering Complex Systems) est une méthodologie dédiée à la conception et l'implémentation de systèmes complexes basée sur PASSI (cf. ci dessus)(Cossentino et al., 2007b). Cette méthodologie est particulièrement adaptée au développement de SMA organisationnels et holoniques (Holonic MAS). Un holon est une entité auto-similaire qui peut être vue comme une organisation contenant des éléments autonomes (holons), ou comme une entité autonome à part entière. Ce concept peut être rapproché de la notion d'agent composé ou récursif. ASPECS propose en outre, une approche MDA en s'appuyant sur trois méta-modèles ou domaines distincts :

1. **Problem domain** : ce méta-modèle permet de décrire l'organisation du problème indépendamment de la solution. Il propose des concepts qui seront utilisés dans les phases préliminaires de la méthodologie.
2. **Agency domain** : il s'agit d'un premier pas vers la solution, le méta-modèle introduit les concepts liés aux agents, il décrit la solution par raffinement des concepts du problème. La solution peut éventuellement s'appuyer sur le principe de holon.
3. **Solution domain** : est directement lié à l'implantation de la solution sur une plate-forme spécifique. Pour ASPECS, la plate-forme privilégiée est appelée Janus [8], elle permet de faciliter le développement et l'implantation de systèmes holoniques ou organisationnels.

Les concepts de ces trois domaines sont utilisés à différentes étapes de la méthodologie. ASPECS offre une méthodologie complète et s'appuie sur une plate-forme dédiée aux SMA

7. http://www.aspecs.org
8. http://www.janus-project.org

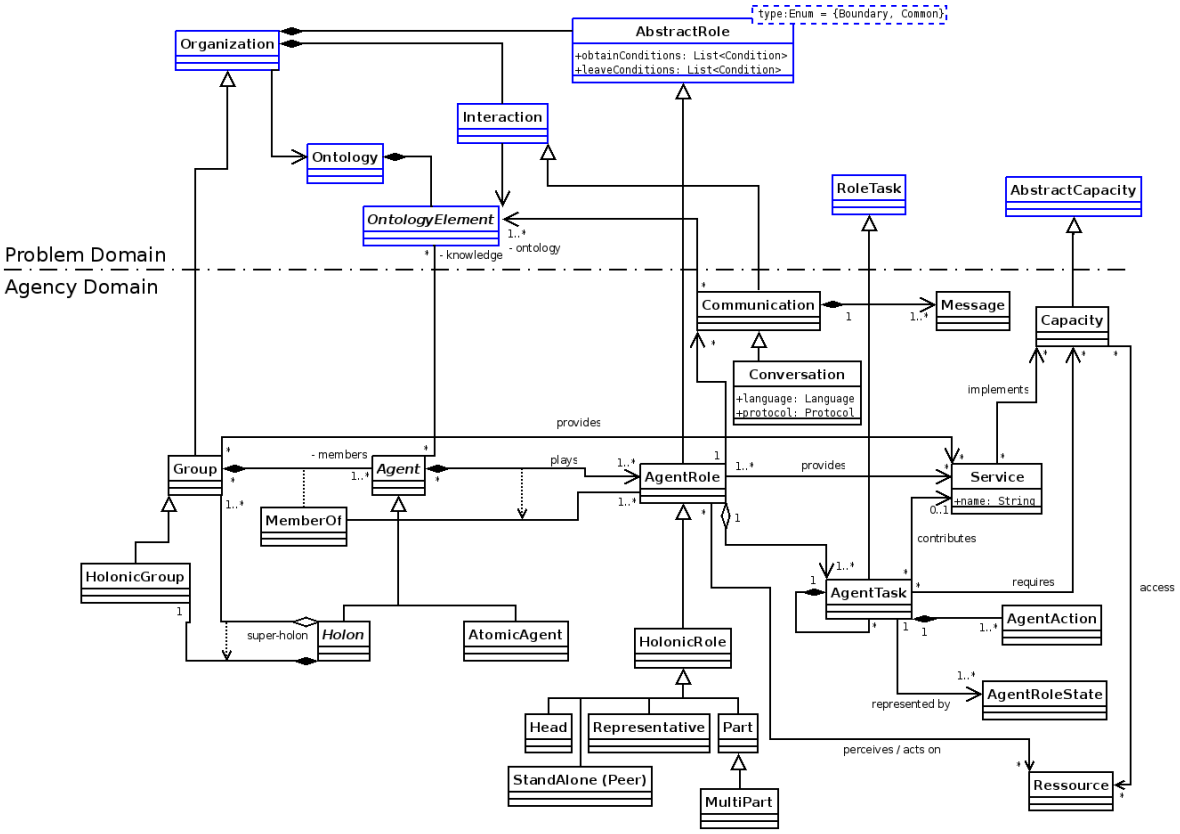

Figure 5.2 — Méta-modèle ASPECS : Problem/Agency domains

holonique : Janus. Elle propose une approche MDA ou chaque niveau d'indépendance (PIM, PSM) est représenté par un méta-modèle. Le mapping entre ces niveaux est réalisé a priori et ce, jusqu'à la plate-forme d'implantation ou du moins jusqu'au domaine de la solution. Cette méthodologie propose de s'appuyer sur une structure et une organisation (holonique) déterminée à la conception. Les agents peuvent modifier leur rôle ou intervenir à différents niveaux dans cette organisation. Cependant, il nous semble difficile d'allier cette vision à celle de systèmes auto-organisateurs.

5.6 Prometheus

Prometheus est une méthode de développement orientée agent (Padgham and Winikoff, 2002) implémentée par un outil dédié : Prometheus Design Tool (PDT) [9] (Padgham et al., 2005). Prometheus couvre toutes les phases depuis les spécifications du système jusqu'au test ou au debugging. En revanche, PDT ne permet de mettre en œuvre que les trois principales phases liées à la conception du système (cf. figure 5.3) :

1. **Spécification du système** : l'objectif de cette phase est de déterminer les scénarios d'utilisation du système, les acteurs interagissant avec le système au travers de ces scé-

9. `http://www.cs.rmit.edu.au/agents/pdt/`

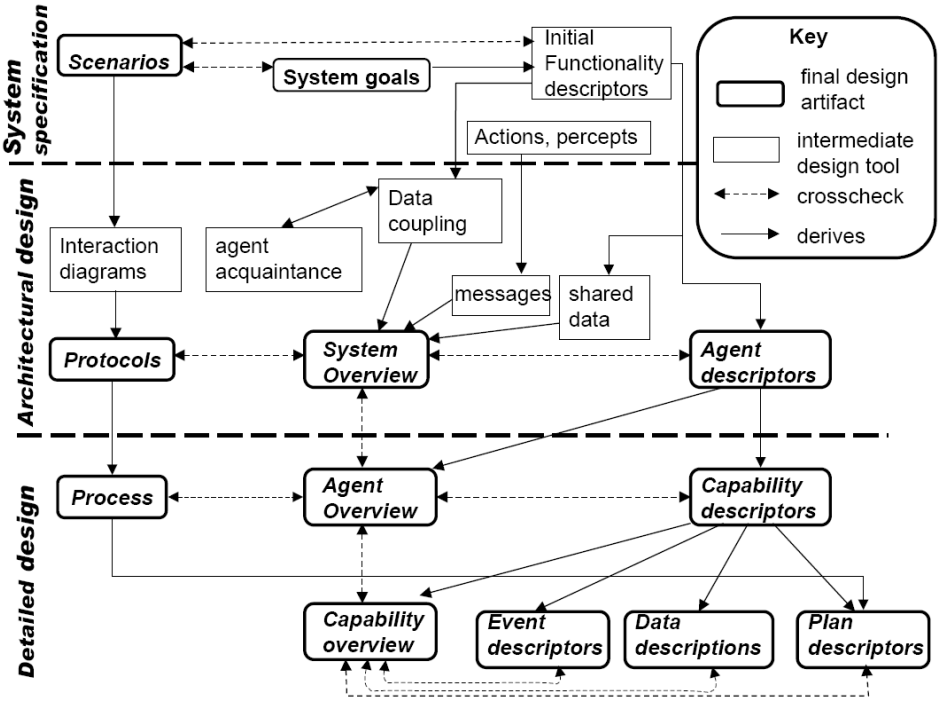

Figure 5.3 — Les phases de la méthodologie Prometheus

narios, les objectifs du systèmes, les agents, leurs perceptions et actions et les données externes au système. Il s'agit d'une vision préliminaire du système qui détermine les différents rôles (fonctionnalités).

2. **Conception de l'architecture** : cette phase a pour objet la définition des types d'agents précédemment identifiés. Elle permet de préciser les différents protocoles d'interaction mis en jeu entre les agents du système.
3. **Conception détaillée** : il s'agit de décrire la structure interne des agents en termes de capacité et leur comportement en fonction des différents protocoles dans lesquels ils interviennent.

PDT permet entre autres, de vérifier la cohérence inter diagramme de son modèle, il offre aussi des moyens pour faciliter la résolution de noms au sein des entités du modèle. Bien qu'il soit possible de définir un mapping entre le résultat de la modélisation et des plate-formes spécifiques agent (Sudeikat et al., 2004), tel que JACK (Busetta et al., 1999), l'outil ne fournit pour l'instant aucun moyen de générer ce code.

5.7 IODA

IODA (Interaction-Oriented Design of Agent simulations) [10] est une méthode de conception de système multi-agent dédié à la simulation (Kubera et al., 2008). Elle place au centre

10. `http://www2.lifl.fr/SMAC/projects/ioda/generalites/ioda.html`

de la conception la notion d'interaction. Ainsi, ces interactions orientent l'ensemble du processus de conception. L'idée est de porter davantage d'attention aux interactions qui peuvent avoir lieu dans le système qu'aux agents qui les réalisent(Mathieu and Picault, 2006). Une interaction possède un élément déclencheur qui motive son exécution, des pré-conditions à vérifier et une liste d'actions à mener. Chaque agent possède des caractéristiques et des primitives qui seront déduites des interactions qu'il est capable de réaliser et des actions qu'il aura à subir. De plus, ils spécifient un ensemble d'*affectations* qui caractérisent les interactions qu'il peut réaliser en en précisant la cible, la priorité et la distance à laquelle elle peut intervenir. Les agents doivent donc être identifiés comme étant la cible ou la source d'une interaction, IODA propose à cet effet une représentation tabulaire (*matrice d'interactions*) qui est raffinée au cours des différentes étapes de conceptions :

1. **Identification des agents et interactions :** cette première étape permet de spécifier les entrées de la *matrice d'interactions* (les agents) et de décrire de manière informelle les interactions ayant lieu entre ces agents.
2. **Description des interactions** Les interactions identifiées sont détaillées en termes de *précondition*, *déclencheur* et *actions*.
3. **Description des agents** Cette étape permet de décrire les perceptions, les actions et les caractéristiques des agent à partir du détail des interactions. Elle permet également d'identifier des caractéristiques communes à un ensemble de type d'agents et de définir des liens d'héritage si nécessaire.
4. **Affectation des interactions aux agents** Le moteur de simulation fournit à chaque étape les interactions potentielles pour chaque agent. Décrire le comportement des agents revient ainsi à décrire la sélection de ces interactions. Cette étape permet, en outre, de spécifier les priorités et les gardes pour les différentes interactions ; c'est-à-dire de définir l'ensemble des affectations de chaque agent.
5. **Détermination de la dynamique du modèle** La dernière étape de la méthodologie permet de décrire, si nécessaire, la dynamique de la simulation. Cela consiste à exprimer de quelle manière évolue la *matrice d'interactions* et par conséquent, de quelle manière sont modifiées dans les interactions potentielles des agents.

Le résultat de ces étapes peut être exécuté sur une plate-forme dédiée à la simulation appelée JEDI (Java Environment for the Design of agent Interactions) [11]. JEDI met en œuvre un moteur de simulation en temps discret reprenant les principes d'interaction exprimés dans IODA. Cette similitude dans les concepts a permis de définir un outil de génération JEDI builder [12] qui permet de construire une matrice d'interaction en suivant la méthodologie IODA et de produire une implémentation JEDI du modèle défini. Cependant, la spécification IODA peut également être implanter dans différentes plate-formes de simulation à l'image de NetLogo. En outre, IODA et la plate-forme JEDI permettent également de garantir, dans le cadre de simulations, la reproductibilité de phénomènes émergents. C'est à dire, que deux

11. `http://www2.lifl.fr/SMAC/projects/ioda/jedi/`
12. `http://www2.lifl.fr/SMAC/projects/ioda/jedi/`

exécutions d'une simulation paramétrée de la même manière possède le même résultat en termes de phénomènes émergents observables. Cette caractéristique est particulièrement importante pour la simulation puisqu'elle permet de déterminer l'influence des paramètres sur l'obtention de ces phénomènes. La méthodologie IODA permet un développement rapide de simulation en offrant des concepts simples et une implémentation efficace de ces derniers sur une plate-forme dédiée. Toutefois, il semble difficile d'appliquer, en l'état, cette méthodologie pour d'autre domaine que celui de la simulation. Elle paraît dès lors « trop spécifique »pour les objectifs de développement de systèmes complexes que nous nous sommes fixés.

5.8 ADELFE

ADELFE propose une méthodologie pour concevoir des logiciels à fonctionnalité émergente basés sur l'approche AMAS (Picard, 2004). La fonction globale du système à développer est considérée comme émergente au regard des interactions des agents qui s'organisent par coopération pour produire cette fonction. Les agents coopératifs, comme nous l'avons vu dans le chapitre 2, ne s'attachent qu'à la réalisation de leur fonction locale et au maintien d'un état coopératif.

La méthodologie orientée agent ADELFE vise à guider les concepteurs d'AMAS dans le cadre d'un processus de développement fondé sur le processus unifié (Unified Process) (Jacobson et al., 1999). ADELFE couvre les phases habituelles de la conception d'un logiciel, des exigences à la conception, il utilise la notation UML et l'extension d'UML proposée dans AUML, en particulier les notations liées aux AIP (Agent Interaction Protocoles) (Odell et al., 2000). Son objectif était de travailler sur les aspects qui ne sont pas encore pris en compte, tels que l'adaptation à un environnement complexe et dynamique. le processus ADELFE est constitué de quatre phases :

1. **Etude des besoins préliminaires :** cette phase permet de concevoir un cahier des charges précis avec le client, de dégager les limites du système en devenir et de spécifier un vocabulaire commun lié au domaine d'application du système.
2. **Etude des besoins finals :** elle permet, à partir du cahier des charges précédemment établi, d'identifier les entités (actives ou passives) entrant en jeu dans le système et de déterminer celles qui constitueront son environnement. Les interactions entre le système et ces entités sont étudiés plus précisément.
3. **Analyses :** cette étape est particulièrement importante puisqu'elle permet de vérifier l'adéquation du problème avec la théorie des AMAS et de fournir une première version de l'architecture logicielle en conséquence.
4. **Conception :** cette phase est dédiée à la conception détaillé de l'application et notamment des agents et de leurs interactions. Elle permet également d'établir les situations de non coopérations et de les décrire sous forme tabulaire.

La méthode ADELFE concerne les applications conçues suivant la théorie AMAS, certaines activités ou tâches spécifiques ont ainsi été ajoutées au processus unifié afin de prendre en compte l'adaptation. Par exemple, lors de l'établissement des besoins préliminaires et finals, des tâches concernant la modélisation de l'environnement et l'expression de situations qui peuvent être inattendue voir dangereuses pour le système (situations de non-coopération) ont été ajoutées. De même, dans la phase d'analyse, deux activités sont spécifiques aux AMAS, en particulier l'activité 11. ADELFE permet au concepteur de décider si l'utilisation de la théorie AMAS est nécessaire à l'application. ADELFE fournit également des guides pour identifier les agents coopératifs parmi toutes les entités définies au cours des tâches préliminaires. Concernant la phase de conception, trois activités ont été ajoutées. La première concerne l'étude des relations entre agents, la deuxième concerne la conception de détaillée des agents. Dans cette activité, les échecs à la coopération sont définis. Enfin, la dernière activité de prototypage rapide contribue à la description et la vérification du comportement de l'agent.

5.9 Synthèse

Notre objectif est de fournir les moyens de concevoir et d'implanter des systèmes complexes. Or si nous considérons ce type de système, il est un aspect inévitable : leur caractère émergent. Ceci implique que l'on ne peut pas déterminer précisément de trace et, par conséquent, d'algorithme capable de décrire l'apparition d'un phénomène (structure particulière, fonction ou propriété). De plus, l'organisation de ces systèmes ne peut être connue à priori, et par conséquent décrite à la conception puisqu'elle est issue de l'auto-organisation des parties le constituant. Il est donc nécessaire de focaliser l'attention du concepteur sur le niveau macro (l'agent) et sur les critères qui favorise l'auto-organisation.
Dans la plupart des approches présentées en début de chapitre, l'analyse du problème est réalisée par une démarche *« top-down »* devant conduire à partir de l'étude et de la définition du système à la conception de l'agent au sein d'une organisation connue où il joue un rôle lui aussi déterminé. Nous pensons que ces approches ne sont pas adaptées aux systèmes auto-adaptatifs que nous considérons, à cause de leur caractère émergent. En revanche, en s'attachant aux critères et à la description des règles qui régissent l'auto-organisation du système au niveau de l'agent, le processus ADELFE apporte une réponse différente à cette problématique. Il propose en effet d'attacher une importance plus grande aux mécanismes de coopération et d'auto-organisation qui doivent permettre de faire émerger la fonction globale du système. La fonctionnalité du système est donc le produit attendu de l'auto-organisation du système et non un ensemble de tâches prédéfinies réparties aux sein des agents qui le composent.
Si le processus ADELFE constitue le choix le plus adapté à notre objectif, il ne possède pas toutes les capacités que nous désirons. Il utilise la notion de modèle, comme nous le souhaitons, mais ne fournit pas une formalisation complète de l'agent en termes suffisamment précis. Il ne propose pas de moyen de décrire, au sein de modèle, les propriétés des

agents et leur implication précise dans le processus de coopération. De plus, il ne propose pas de guide méthodologique pour l'implémentation du système. L'utilisation d'ADELFE et les améliorations que nous pouvons y apporter sont discutées dans un chapitre ultérieur.

6 Contexte

LE but de notre travail est de parvenir à fournir des outils de conception et d'implémentation pour les systèmes complexes. Nous considérons pour atteindre cet objectif que l'approche des systèmes auto-organisateurs et plus particulièrement le paradigme des SMA adaptatifs est un moyen de faire face aux problèmes liés à l'auto-adaptation. De même, le modèle d'agent flexible autorise l'adaptation dynamique des capacités opératoires des agents. Ce chapitre en précise les fondements ; ils constitueront la base sur laquelle s'appuie notre démarche de génie logiciel. Le dernier paragraphe de ce chapitre présente, en outre, la méthodologie ADELFE en détail ainsi que les améliorations que nous devons y apporter afin de faciliter son utilisation et de compléter ses compétences.

6.1 La théorie des AMAS

Un système multi-agent adaptatif ou AMAS est avant tout un SMA. Il bénéficie donc des intérêts de l'approche, décentralisation du contrôle de la fonction au sein d'agents capables de perception et de décision. La notion clef des AMAS est l'auto-organisation, celle-ci garantit l'adéquation d'une part, de la fonction globale par rapport aux besoins et d'autre part, du système avec son environnement (Gleizes, 2004; Capera et al., 2003). Le principe est que la fonction du système est issue de la composition des fonctions locales. Changer l'organisation de ces fonctions locales, modifie par là même la fonction globale. En s'auto-organisant face aux pressions de son environnement et en suivant un critère de coopération, le système définit une fonction globale dite adéquate. Ce critère de coopération implique qu'un AMAS ne peut entamer d'action qui soit préjudiciable à son environnement. Maintenir un état coopératif au sein des constituants du système garantit de cette façon qu'il sera toujours en adéquation avec son environnement.

6.1.1 Théorème de l'adéquation fonctionnelle

Un système est dit adéquat s'il remplit la fonction pour laquelle il a été conçu. De tels systèmes n'ont aucune action antinomique (Glize, 2001) envers leur environnement, il compromettrait alors leur adéquation. La théorème de l'adéquation fonctionnelle promeut l'idée qu'il existe toujours un système coopératif et par conséquent un SMA Adaptatif, capable de rendre cette fonction adéquate (Gleizes et al., 1999). Nous explicitons ensuite ce que représente le principe de coopération dans les AMAS.

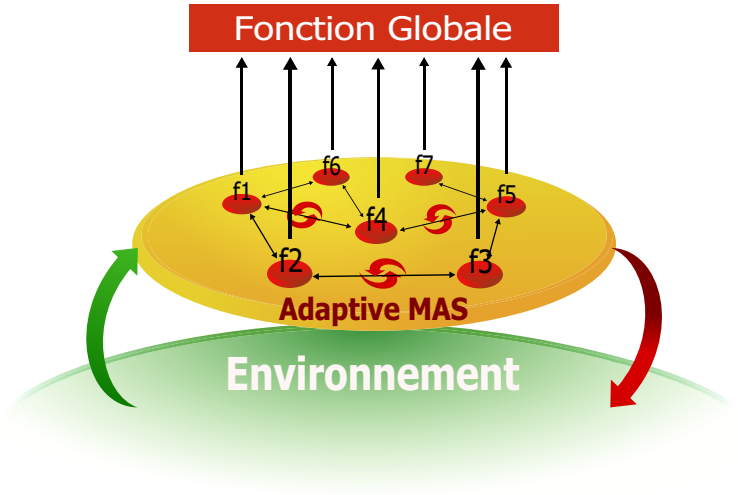

Figure 6.1 — La théorie des AMAS : la fonction globale (niveau système) est le produit de l'auto-organisation des fonctions locales (niveau agent).

6.1.2 La coopération au sein des AMAS

Ferber (1999) donne une définition de la coopération comme une situation d'interaction particulière, en pénurie de ressources ou en manque de compétences pour les utiliser. Pour un agent AMAS une attitude coopérative consiste à résoudre au mieux ces situations conflictuelles avant qu'elles n'occasionnent de dysfonctionnement. Ces situations sont appelées Situation Non-Coopératives (SNC) et sont classifiées en fonction du moment où elles sont susceptibles d'être détectées dans le cycle de vie perception-décision-action de l'agent (Glize, 2001) :

1. **Perception**
   - *Incompréhension* : l'agent n'est pas capable d'extraire l'information d'un signal.
   - *Ambiguïté* : l'agent attribue plusieurs interprétations à un même signal.
2. **Décision**
   - *Incompétence* : l'agent n'est pas capable d'exploiter pour son raisonnement l'information extraite d'un signal.
   - *Improductivité* : l'information extraite d'un signal ne permet de tirer aucune conclusion.
3. **Action**
   - *Concurrence* : l'agent pense que son action aura les mêmes conséquences que celles d'un de ses semblables (objectifs communs).
   - *Conflit* : l'agent pense que son action sera incompatible avec celle d'un de ses semblables (objectifs antagonistes).
   - *Inutilité* : l'agent pense que son action n'aura aucune conséquence sur le monde qui l'entoure.

Le comportement d'un agent consiste ainsi à réaliser son objectif local tout en gérant ces situations particulières. Le paragraphe qui suit en présente davantage les principes.

6.1.3 Le modèle d'agent coopératif

Un agent coopératif au sein d'un SMA Adaptatif est guidé par le principe de coopération. Il possède d'une part, un objectif personnel constituant son comportement *nominal* (fonction locale), d'autre part, un comportement dit *coopératif* dont l'objectif est l'auto-adaptation et la correction de SNC. Le rétablissement d'un état coopératif prend le pas sur le comportement *nominal* et provoque une modification dans les interactions de l'agent pour pallier cette situation. Cette activité coopérative au sein de l'ensemble des agents est responsable de l'auto-organisation du système et par conséquent de l'adaptation de la fonction globale.

Un agent coopératif suit un cycle de vie classique qui débute par une phase de *perception* de son environnement entrant en jeu dans la prise de *décision* des *actions* à mener. Si une SNC a été détectée durant une de ces phases alors les actions issues du processus de décision de l'agent auront pour but de la réparer ou de la prévenir. Dans le cas contraire, il s'agira d'un ensemble d'actions permettant à l'agent de réaliser sa fonction locale. Le comportement *coopératif* subsume le comportement *nominal* garantissant alors pour le système le maintien de cette coopération.

6.1.4 Analyse

L'utilisation de SMA peut être envisagée comme une première étape pour simplifier la conception de systèmes complexes, notamment grâce aux principes qu'ils promeuvent :

- concevoir un système comme un ensemble d'entités autonomes interagissantes (multi-échelle),
- décentraliser le contrôle et la fonction du système,
- considérer le système en *couplage* avec son environnement,
- définir des agents percevant et agissant sur cet environnement (context-awareness).

Au delà de ces caractéristiques communes à l'ensemble des SMA, nous avons présenté le principe de SMA Adaptatif qui place l'*adaptation fonctionnelle* au coeur des préoccupations. Cette adaptation est obtenue par un processus d'auto-organisation guidée par le principe de coopération. L'approche AMAS permet de cette manière :

- de centrer la conception d'un système sur les *constituants* et leurs *règles de coopération*,
- de prendre en charge des *environnements très dynamiques* grâce au mécanisme d'*auto-organisation*,
- de considérer la *fonction globale* du système comme *émergeant des fonctions locales*, à l'image des caractéristiques des systèmes complexes.

Ainsi, les SMA Adaptatifs constituent un moyen d'abstraire des concepts issus des systèmes complexes tels que l'émergence et l'auto-organisation. De plus, ils permettent de

concevoir des systèmes multi-agents flexibles, adaptatifs répondant aux besoins de *« context-awareness »* et de réactivité exprimé dans le paragraphe 1.3.

Les AMAS constituent un moyen de mener à bien l'*« adaptation fonctionnelle »* et *« structurelle »* des systèmes (voir paragraphe 1.3.3), grâce au principe d'auto-organisation coopérative. Qu'en est-il de l'adaptation des constituants de ces systèmes ? C'est-à-dire, des agents coopératifs eux-mêmes. Le processus de décision permet d'adapter la fonction à des modifications de l'environnement logique ou physique de l'agent. Toutefois, rien dans la théorie des AMAS ne les autorise à modifier la manière de rendre cette fonction *« adaptation non-fonctionnelle »* . Nous proposons le principe d'agent flexible permettant, entre autres, de fournir aux agents cette capacité.

6.2 Agent flexible

Leriche (2006) propose le principe d'agent *« flexible »* , à savoir un agent dont l'architecture est adaptable à l'exécution tout comme à la conception et la maintenance. Ce principe d'agent flexible adaptable a été mis en oeuvre dans le middleware **JavAct** [1]. Pour rendre un agent capable de s'adapter à l'exécution, différents principes sont mis en jeu et présentés dans les paragraphes suivants.

6.2.1 Séparation des préoccupations

Dans le modèle d'agent flexible, adaptable nous distinguons deux niveaux tous deux liés à des préoccupations différentes :

- **Niveau opératoire** : ce niveau représente les mécanismes opératoires de l'agent. Il correspond aux capacités élémentaires de l'agent comme la communication, la mobilité, la création dynamique, etc.
- **Niveau fonctionnel ou comportemental** : ici il s'agit du comportement de l'agent, l'utilisation des mécanismes opératoires afin de remplir la fonction pour laquelle il est destiné.

Ici, l'intérêt est de permettre l'abstraction de mécanismes de base, comme l'envoi de messages, pour les utiliser dans le but de décrire le comportement de l'agent. Ce comportement peut être défini sans avoir à entrer dans les détails de mise en œuvre de cette communication.

6.2.2 Micro-Composant

Le paradigme *Composant* a prouvé ses qualités pour mettre en pratique la réutilisation, l'encapsulation, le branchement à froid et à chaud, etc. (Szyperski et al., 2002). Le modèle d'agent flexible est construit autour du principe de micro-composant, c'est-à-dire de composant à grain fin mettant en œuvre des services d'ordre opératoire (Leriche and Arcangeli,

1. `http://www.irit.fr/PERSONNEL/SMAC/arcangeli/JavAct_fr.html`

2007). Chaque composant est en charge d'au moins un service opératoire (mobilité, création dynamique, etc.) et peut être remplacé à l'exécution ou la conception par un autre, sous réserve qu'il rende les mêmes services (implante la même interface).

6.2.3 Médiateur

Pour mener à bien la reconfiguration dynamique, l'agent flexible est organisé suivant le patron de conception médiateur (Gamma et al., 1994). Le médiateur implémente des liens de délégation entre les différents micro-composants du niveau opératoire. Depuis le niveau comportemental de l'agent il est possible d'utiliser les services opératoires grâce à ce même médiateur. De surcroît, il a en charge l'adaptation puisqu'il connaît l'ensemble des micro-composants.

6.2.4 Style d'architecture des agents

Un style architectural est un ensemble de règles et de contraintes concernant la composition et l'évolution d'un système logiciel (dans notre cas d'un agent) (Medvidovic and Taylor, 2000). Notre style d'architecture a été présenté dans (Arcangeli and Leriche, 2007) ; il est dérivé de l'architecture des agents de JavAct, un intergiciel pour la programmation à base d'agents mobiles adaptatifs [2] développé dans notre équipe. Ce style favorise la modularité et l'adaptation ; il est caractérisé par les propriétés suivantes :

- séparation entre le niveau comportemental ou fonctionnel et le niveau opératoire (mécanismes internes) également appelé méta-niveau (c'est le niveau de l'intergiciel),
- organisation du niveau opératoire en composants de grain fin (c'est-à-dire offrant un service unique) appelés micro-composants,
- assemblage des micro-composants en étoile autour d'un connecteur appelé *Mediator* conformément au patron de conception Médiateur (Gamma et al., 1995).

Ce style fixe le cadre et les limites de l'adaptation (axe fonctionnel-opératoire et axe statique-dynamique). L'agent est configurable statiquement et il peut remplacer dynamiquement un micro-composant par un autre connu initialement ou acquis dynamiquement. L'articulation entre niveau comportemental et niveau opératoire s'effectue via une classe intermédiaire spéciale -*QuasiBehavior*- qui fournit au niveau comportemental l'accès aux services offerts par les micro-composants.

2. `http://www.irit.fr/PERSONNEL/SMAC/arcangeli/JavAct_fr.html`

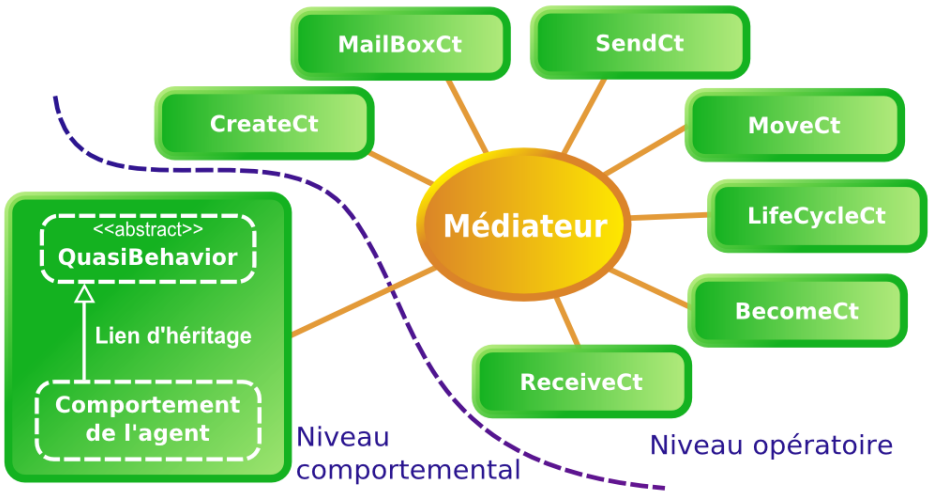

Figure 6.2 — Micro-architecture d'un agent JavAct. Un exemple d'agent flexible.

La figure 6.2 illustre ces principes et présente la *micro-architecture* des agents constituant le middleware JavAct. Un agent JavAct est capable, de se déplacer (*MoveCt*), de communiquer par messages (*SendCt, ReceiveCt, MailBoxCt*), de changer de comportement dynamiquement (paradigme acteur, *BecomeCt*) et de créer de nouveaux agents (*CreateCt*). Les *micro-composants* du niveau opératoire réifient les capacités éponymes de l'agent Javact. Le *Mediator* permet d'implémenter le principe de délégation pour faciliter l'évolution de l'architecture ; il fournit un ensemble de primitives permettant le remplacement des micro-composants. Il permet également par le biais du *QuasiBehavior*, d'accéder aux services du niveau opératoire qui peuvent de cette façon être utilisées pour programmer le *niveau comportemental*(*Behavior* dans la figure 6.2).

6.2.5 Analyse

La notion d'agent flexible permet grâce à un ensemble de micro-composants assemblés autour d'un médiateur de définir une *micro-architecture* adaptable appelée *modèle d'agent*. Ce modèle peut ensuite être animé en définissant le comportement des agents suivant les services définis au sein du *modèle*. Le principe de délégation et le style architectural en étoile autour d'un médiateur permettent en outre, de modifier dynamiquement l'architecture en remplaçant un micro-composant opératoire si nécessaire. De plus, il est possible à la conception de confectionner un modèle d'agent original adapté à des besoins spécifiques pour qu'il soit utilisable dans différents domaines (intelligence ambiante, simulation,etc.).

Grâce à la notion d'agent flexible, nous possédons un moyen pratique d'adapter un agent selon des préoccupations d'ordre non-fonctionnel à la conception et à l'exécution.

Nous souhaitons prendre en charge l'auto-adaptation le plus tôt possible dans le cycle de vie de nos systèmes. Pour ceci, nous voulons mettre à profit les capacités des AMAS et du principe d'agent flexible dès la conception. Le chapitre suivant présente la dimension génie logiciel qui contribue pour une grande part à l'ensemble de ces travaux.

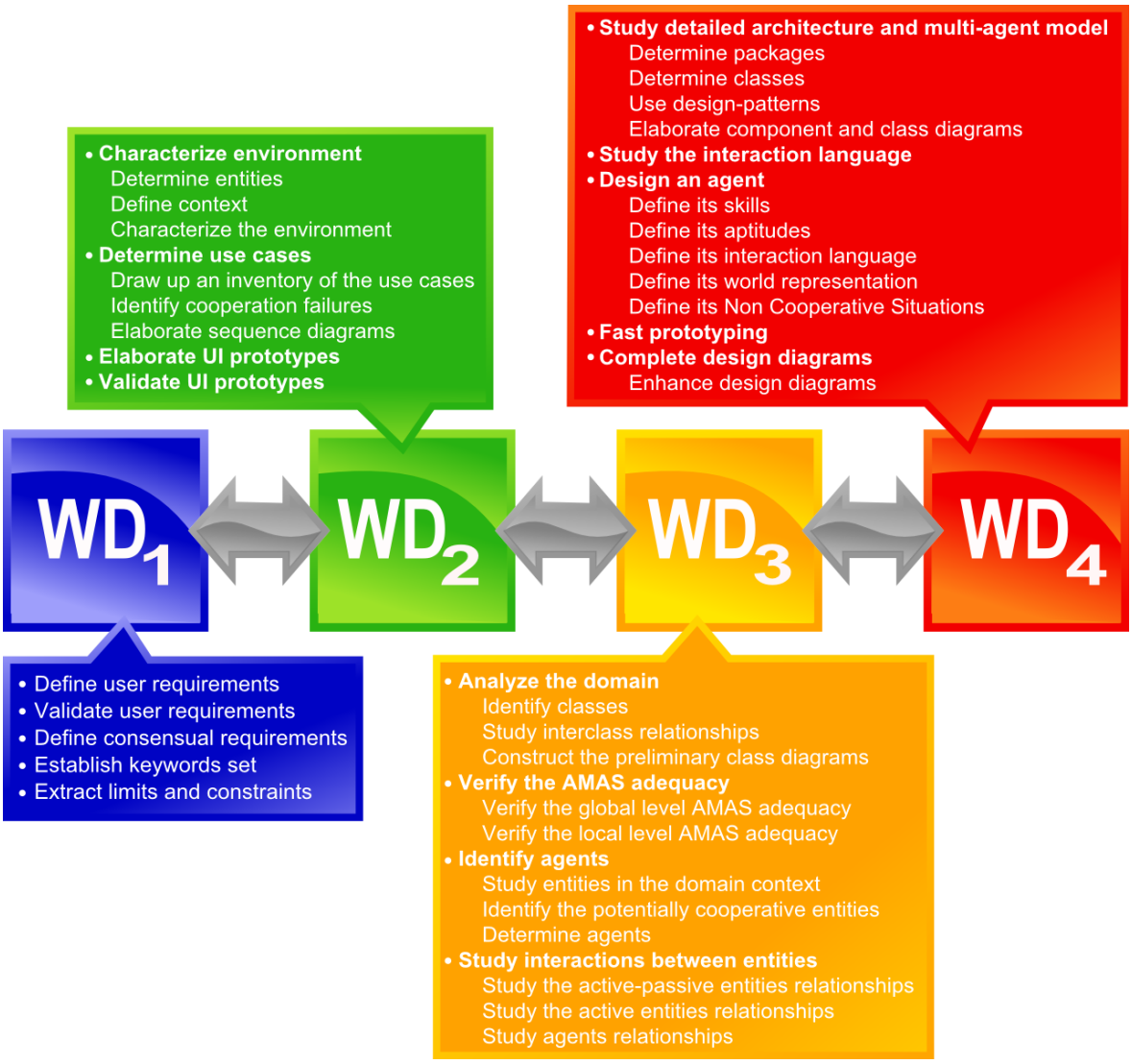

Figure 6.3 — Les phases de la méthode ADELFE.

6.3 Méthodologie Adelfe

6.3.1 Présentation

ADELFE [3] est une méthode orientée agent pour la conception de SMA Adaptatifs (Adaptive Multi-Agent System AMAS). Pour illustrer la notion de fonction globale émergente, nous pouvons prendre un exemple d'application qui fût développé grâce à ADELFE. Il s'agit d'un problème de coordination où des robots doivent transporter des marchandises d'une

3. ADELFE est un acronyme pour : Atelier de Développement de Logiciels à Fonctionnalité Emergente. Il s'agit à l'origine d'un projet RNTL (2000-2003) impliquant des partenaires industriels : ARTAL Technologies (`http://www.artal.fr`) and TNI-Valiosys (`http://www.tni-valiosys.com`), aussi bien qu'académiques : IRIT (`http://www.irit.fr/SMAC`) et L3I (`http://www-l3i.univ-lr.fr`). Voir `http://www.irit.fr/ADELFE`

zone de stockage à un dépôt (Picard and Gleizes, 2005). Le problème réside dans le fait que l'accès entre ces différents lieux ne permet pas à deux robots de se croiser. Il est donc nécessaire que les robots affectés à cette tâche s'organisent afin de ne pas bloquer le processus de transport dans son intégralité. Ainsi, la solution proposée en utilisant l'approche AMAS (Capera et al., 2003) considère chaque robot comme un agent coopératif dont le comportement nominal (fonction locale) est d'apporter les marchandises d'une pièce à une autre et le comportement coopératif est d'éviter les collisions tout en essayant de ne pas bloquer les autres robots. A partir de ces règles de coopération simples, un sens de circulation émerge sans que celui-ci n'ait été précisé dans le code de l'application. Ainsi, la fonction globale du système de transport a émergé à partir l'auto-organisation des fonctions locales réalisées par chaque robot.
Elle propose un processus de développement basé sur le processus unifié (cf. paragraphe 4.1), dont l'objectif est de guider et d'assister le développeur de SMA Adaptatifs grâce à des activités spécifiques ainsi qu'un ensemble d'outils associés (Bernon et al., 2005). Ce processus est organisé selon quatre phases qui sont brièvement présentées dans le paragraphe qui suit.

6.3.1.1 Déroulement du processus

Le processus de développement ADELFE a été défini grâce au langage de modélisation de SPEM 1.1 (Software and System Process Engineering Meta-model) (omg, 2005b), cette présentation utilise par conséquent les concepts de ce langage (voir figure 6.3). Ici, le déroulement du processus est présenté comme un enchaînement de *WorkDefinition* (WD). Il s'agit de l'élément de base d'un processus, il permet de regrouper et d'organiser un ensemble de tâches plus élémentaires (*Activity*, *Step*) liées entre elles en vue de fournir un produit (*WorkProduct*). Par exemple, le cahier des charges préliminaire est un produit issu de l'étude préliminaire des besoins (WD1). La première phase du processus est vouée à l'établissement des besoins préliminaires du logiciel avec le client : les limites fonctionnelles et les contraintes non fonctionnelles (WD1). Cette phase n'est pas spécifique à l'approche AMAS ; elle provient directement du processus unifié. Cependant, ADELFE est une méthodologie spécifique, dédiée aux AMAS ; elle définit dès lors, un certain nombre d'activités elles aussi spécifiques à cette approche. Ces tâches s'intègrent parmi celles plus conventionnelles issues du processus unifié, seule l'étude des besoins préliminaires (WD1) n'intègre pas de tâche spécifique :

1. Pendant l'établissement des **besoins finals** (WD2) :
   - *Activité 6* : il s'agit de la caractérisation de l'environnement (est-il accessible, dynamique, déterministe, etc ?),
   - *Activité 7 - étape 2* : cette étape permet d'identifier les interactions pouvant provoquer des erreurs dommageables, des *« échecs de coopération »* pour l'approche AMAS,
2. Pendant la phase d'**analyse** (WD3) :
   - *Activité 11* : cette tâche consiste à établir l'adéquation du problème et des besoins identifiés avec l'approche AMAS, ainsi elle conditionne le reste des activités,

- *Activité 12* : une fois l'adéquation déterminée, il est nécessaire d'identifier les agents impliqués dans le système parmi les entités déterminées au cours des activités préliminaires,
- *Activité 13 - étape 3* : dès lors que les agents ont été mis en exergue, il est possible d'étudier précisément leurs relations.

3. Pendant la phase de **conception** (WD4) :
   - *Activité 15* : il s'agit d'étudier les langages d'interaction entre agents de l'application, c'est-à-dire de quelle manière ils échangent de l'information (messages, protocoles, etc.),
   - *Activité 16* : cette activité permet la conception complète des agents coopératifs et de leurs constituants (Représentation, compétences, protocoles d'interactions, situation de non-coopération, etc.)
   - *Activité 17* : à partir de la conception détaillée des agents il est possible de réaliser des prototypes en vue de rendre compte du comportement des agents.

Toutes ces activités sont menées en employant des langages et des outils permettant d'en faciliter la réalisation.

6.3.1.2 Les outils associés

Le processus ADELFE est basé sur UP ; il met donc en œuvre pour mener à bien les premières de ses activités le langage UML (Rumbaugh et al., 1999), notamment les cas d'utilisation. De même, la caractérisation de l'environnement et des échanges qu'il réalise avec le système utilise le diagramme de séquence UML. ADELFE propose aussi un assistant permettant de déterminer l'adéquation du problème avec la théorie grâce à un ensemble de questions dont les réponses conditionnent le degré d'adéquation. (Picard, 2003) propose un profil UML spécifique permettant de prendre en compte au sein de diagramme UML le modèle d'agent coopératif principalement au sein du diagramme de classe. Il étend également la notation AUML (Odell et al., 2000) de manière à prendre en charge la coopération dans la description des protocoles d'interaction (Agent Interaction Protocol). Certes, ces outils sont intégrés dans un atelier mais les évolutions du standards UML ont récemment remis en question les extensions qui avaient pu y être apportées.

6.3.2 Les besoins actuels

ADELFE en étendant le méta-modèle UML1.X par le biais d'un profile (Picard, 2003), s'assujettit à ses éventuelles évolutions. Cela ne porte pas à conséquence, si ces évolutions ne pouvaient remettre en cause les adaptations réalisées pour accorder l'UML aux besoins spécifiques des AMAS. L'objet de ce paragraphe n'est pas une critique du principe de profile UML, mais bien une justification de l'usage de langages de modélisation dédiés (cf. paragraphe 4.2.2) hors du giron de l'OMG et de son standard UML. Pour étayer nos propos, nous prendrons pour exemple l'expression des interactions agents dans l'actuelle version

d'ADELFE qui, nous allons le voir, posent un certain nombre de problèmes.

Un autre aspect important que nous avons soulevé dans les chapitres précédents est l'automatisation de tâches comme la génération de code. Jusqu'à présent ADELFE ne propose pas une démarche intégralement dirigée par les modèles, la dernière étape concernant l'implantation des modèles réalisés n'y est pas définie. C'est précisément à cette étape que l'intérêt de l'automatisation prend tout son sens. Dans le cadre de nos objectifs, cette dernière étape constitue donc un besoin important.

6.3.2.1 Expression des interactions

En quelques mots, la théorie des AMAS établit la *coopération* comme principe fondamental. Selon la théorie, un système multi-agent conçu pour s'auto-organiser en usant d'un maximum de coopération, c'est-à-dire en recherchant un équilibre dynamique entre les besoins du collectif par rapport à ceux de l'individu, fait émerger une solution adaptée au niveau global à partir des fonctions locales (Capera et al., 2003). Dès lors que la théorie installe la coopération comme critère d'auto-organisation, il paraît indispensable d'en faire état dès la conception et particulièrement dans l'expression des interactions agent mettant potentiellement cette coopération en défaut (*Situations de Non-Coopération*).

Une manière de représenter, de modéliser les interactions, est d'utiliser les diagrammes d'interaction de l'UML notamment le diagramme de séquence, approprié à la spécification d'échanges de messages. Ainsi, la FIPA a défini une notation spécifique sous la forme d'un profile UML1.X pour adapter ce diagramme à la définition de protocole d'interaction spécifique aux agents (AIP Agent Interaction Protocol)(Odell et al., 2000). Il s'agit d'un pas réalisé dans la direction d'une modélisation spécifique aux agents, mais il n'y est toutefois pas encore question de coopération. C'est pour cette raison qu'ADELFE a elle aussi défini des extensions aux AIP afin de prendre en charge l'expression des *échecs à la coopération* dans les interactions agent. Les dernières évolutions de l'UML (omg, 2007b) (version 2.0 et 2.1) ont pris en compte certaines des adaptations souhaitées par la FIPA dans la représentation de protocoles, rendant les AIP définies sur UML1.X obsolètes et caduques les modifications qu'y apportait ADELFE. Nous pouvons bien sûr objecter que rien n'empêche l'utilisation des versions compatibles de l'UML. Il y a peu d'intérêt à s'inscrire dans une démarche visant à s'appuyer sur des standards si ce n'est pour suivre ses évolutions. Les évolutions potentielles des méta-méta-modèles comme Ecore, bien que toujours envisageables, ne portent que sur un nombre réduit de concepts. De plus, EMF et les plugins qui y sont rattachés tels que GMF, ont toujours fourni des outils de migration en cas de refonte majeure des modèles qui les supportent. L'évolution de ces langages de méta-modélisation apparaît plus facile à maîtriser et moins fréquente. A cet égard, nous étudions dans le paragraphe qui suit l'intérêt que nous avons pu trouver à l'utilisation de langages spécifiques hors du mécanisme de profile UML.

L'expression des interactions agents constitue une part importante de la conception de nos systèmes (WD4 *Activité 15*). Nous l'avons vu, il est important d'y faire figurer les éventuelles mises en défaut de la coopération. Cependant, ADELFE ne propose pas pour l'instant, de formalisme adapté à la définition du règlement de telles situations, outre une représenta-

tion tabulaire peu utilisable dans un cadre de développement dirigé par les modèles.

6.3.2.2 Expression du comportement coopératif

Les *situations de non-coopération* se caractérisent par un état de connaissance et de perception de l'agent mettant à l'épreuve son attitude coopérative : par exemple, l'action qu'il décide d'entreprendre interfère avec celle d'un de ces congénères (Glize, 2001). Une telle situation peut s'interpréter comme une condition sur les variables d'état de l'agent (ses connaissances et ses perceptions) et doit conduire au choix d'une action « réparatrice ». C'est-à-dire, une action permettant de rétablir la coopération. Ce principe est au cœur de nos préoccupations et constitue, bien entendu, une pièce majeure dans l'établissement du comportement des agents coopératifs. De ce fait, il est nécessaire de permettre au concepteur du système d'en spécifier les détails ; la question d'un formalisme adapté à cet objectif est alors posée. Si nous considérons un schéma général du comportement coopératif d'un agent, il s'agit d'un ensemble de règles, dont chacunes d'elles correspond au traitement d'une situation précise (une situation de non-coopération représentée par un état de l'agent implique une ou plusieurs actions). Il apparaît clairement qu'il est nécessaire de fournir au concepteur une sorte de langage logique, lui permettant de définir des règles de coopération.

UML ne fournit pas de notation adaptée à cet effet, ce qui implique qu'un profile n'apporterait rien du point de vue de la réutilisation de diagramme, il nécessiterait l'ajout de nouveaux concepts et potentiellement de nouvelles représentations graphiques. Pourtant, il doit être possible de définir les prémices de ces règles par le biais d'expressions du langage OCL (Object Constraint Language) (omg, 2003b) au sein de l'actuel profile ADELFE ; cependant en termes de représentation et de réutilisation cela reste difficilement exploitable. UML permet toutefois de représenter des états consécutifs d'objets au sein de diagrammes d'états transitions, mais cela ne constitue pas réellement une notation adaptée aux besoins de l'expression des règles de coopération.

Une solution, celle que nous privilégions, est de composer un langage spécifique à partir des besoins que nous venons d'exprimer et de fournir pour celui-ci une syntaxe graphique, une notation adaptée et nous l'espérons intuitive. C'est là l'un des avantages de cette approche, offrir un langage le plus proche possible des préoccupations de l'activité à mener et de son utilisateur.

Troisième partie

Contribution au développement des systèmes multi-agents adaptatifs

Introduction

Nous souhaitons proposer une démarche de développement des systèmes complexes en nous appuyant sur la théorie des AMAS et le principe d'agent flexible. Dans cette partie, nous présentons l'ensemble de nos contributions qui ont permis de mettre cet objectif en pratique au travers d'une démarche dirigée par les modèles.

Le premier chapitre précise les extensions que nous avons apportées à la méthodologie ADELFE. Nous y avons intégré l'utilisation combinée de langages génériques (GPML) tel qu'UML 2.0 et de langages spécifiques (DSML). En effet, UML 2.0 est utilisé dans les phases préliminaires d'ADELFE alors que l'utilisation d'un langage spécifique AMAS-ML a enrichi la phase de conception. Nous présentons également une extension du processus par une phase d'implémentation dirigée par les modèles basée sur un méta-modèle spécifique aux agents flexibles (μADL).

Dans le chapitre suivant, nous décrivons précisément les concepts qui constituent la base de ces deux langages (leur méta-modèle). Ces méta-modèles constituent une description semi-formelle des domaines des AMAS et des agents flexibles. Ils permettent, en outre, de décrire avec précision les aspects comportementaux et opératoires des agents coopératifs.

Le dernier chapitre constitue le cœur de notre approche dirigée par les modèles. Il présente l'ensemble des transformations de modèles qui permettent d'intégrer au sein d'un même processus les langages AMAS-ML et μADL. De plus, ces transformations autorisent la génération automatique d'une partie du code de l'application (comportement des agents) et de son support (mécanismes opératoires).

7 Extension de la méthodologie ADELFE

ADELFE est une méthodologie qui permet la conception d'applications à fonctionnalité émergente. Nous avons vu dans les chapitres précédents l'intérêt de posséder une vision méthodologique du développement de logiciels considérés comme complexes. Cependant, dans sa version actuelle, ADELFE ne propose pas de « guide » pour l'implantation du système conçu. Dans cet objectif, ce chapitre s'articule donc de la manière suivante. Nous présentons d'abord notre réflexion sur l'intérêt de combiner des langages de modélisation génériques et spécifiques selon les phases de la démarche. Les diagrammes d'UML2.0 que nous avons conservés dans la démarche sont ensuite présentés ainsi que les phases dans lesquels ils sont utilisés. Enfin, nous décrivons la nouvelle version d'ADELFE et sa formalisation en SPEM2.0 qui décrit précisément comment les traitements et tâches de modélisation s'enchaînent. A cette occasion, nous décrivons la phase d'implantation qui étend la précédente version d'ADELFE.

7.1 Justification

Nous souhaitons nous placer dans le cadre méthodologique d'ADELFE en y intégrant une plus grande part d'automatisation et des langages spécifiques nous permettant de combiner l'approche AMAS et le modèle d'agent flexible. Dans la partie précédente, nous avons vu l'importance de l'ingénierie dirigée par les modèles pour mettre en pratique de tels buts. Elle permet, en effet, de décrire des langages de modélisation spécifiques (DSML Domain Specific Modelling Language) et d'automatiser le traitement des modèles qui en sont issus. Ce paragraphe regroupe les choix que nous avons réalisés et leurs implications pour atteindre nos objectifs, tout en respectant la méthodologie ADELFE telle qu'elle a été conçue.

7.1.1 Intérêts des langages spécifiques

Au vu de ses dernières évolutions, UML semble prendre une envergure qui dépasse de loin, les besoins que nous pouvons exprimer en termes de modélisation spécifique aux AMAS. Le mécanisme de profil permet de sélectionner et de stéréotyper (étiqueter) les concepts UML afin qu'ils s'adaptent à ces besoins. De plus, il implique un certain nombre de compromis ainsi qu'un couplage étroit avec UML, ce qui peut s'avérer dommageable comme nous l'avons vu dans le paragraphe 6.3.2.1. Bien qu'il soit séduisant de se conformer à un standard supporté par un grand nombre d'outils, nous soutenons que, dans le cadre du

développement de langages de modélisation très précis, une approche spécifique au domaine (DSML) peut être profitable. En effet, il n'est pas toujours possible de se conformer à UML et d'en définir une extension pour prendre en charge des besoins caractéristiques du domaine comme il est apparu pour la spécification de règles de coopération.

En outre, faire le choix de définir un DSML, ce n'est pas renoncer aux standards. Dans le cadre d'une démarche dirigée par les modèles et par le biais d'éventuelles transformations il est tout à fait envisageable et même souhaitable de bâtir des ponts entre langages généralistes et spécifiques. De surcroît, une telle approche, permet de définir et de fournir les moyens en termes de langage, de manipuler des concepts plus proches du domaine étudié et surtout de garder la maîtrise de ces évolutions.

Le processus de définition d'un profil UML est sensiblement identique à celui de la création d'un DSML, d'ailleurs un profil est un DSML à ceci près qu'il emprunte sa syntaxe et une partie de sa sémantique à UML. Toutefois, dans les deux cas, une analyse fine du domaine est nécessaire, elle ne constitue pas un surcoût pour l'approche *Domain Specific*. En effet, ces deux approches ne diffèrent que sur le choix de la réalisation technique. Il s'agit d'une part, de définir un méta-modèle à l'aide d'un langage de méta-modélisation, par exemple Ecore (Budinsky et al., 2003) ; d'autre part, de définir une extension ou une restriction du méta-modèle d'UML (sémantique ou conceptuel) en respectant les règles de bonne formation. Quoi qu'il en soit, nous emploierons par soucis de clarté, l'acronyme DSML pour les langages définis indépendamment d'UML et parlerons de profil dans les autres cas.

7.1.2 Un langage spécifique aux AMAS

Le cadre du développement dirigé par les modèles dans lequel nous inscrivons nos travaux, autorise et encourage l'utilisation de langages de modélisation spécifique ou généraliste ; de plus, il favorise le dialogue ou l'interaction entre ceux-ci grâce aux transformations de modèles (elles-mêmes définies par des langages spécifiques, voir paragraphe 4.2.3). Ce n'est que faire preuve de bon sens que d'encourager l'utilisation de langages adaptés au moment où ils sont les plus utiles au sein d'un processus de développement, qu'ils soient spécifiques ou non. Pour ADELFE, faire cohabiter UML et un langage spécifique que nous appellerons AMAS-ML (AMAS Modelling Language) consiste à extraire du modèle préliminaire, notamment de la caractérisation des entités, les informations nécessaires pour créer un nouveau modèle AMAS-ML. Bien sûr, cette extraction est conditionnée par l'analyse de l'adéquation du problème avec la théorie des AMAS, c'est pourquoi cette activité apparaît comme primordiale dans la méthodologie (Figure 6.3).

7.1.3 Langages spécifiques et généralistes dans ADELFE

Les phases initiales du processus ADELFE peuvent et doivent être prises en charge, pour les activités qui le nécessitent, par un langage généraliste (GPML General Purpose Modelling Language) afin de garantir une indépendance dans les choix d'implantation et d'autoriser le

choix de ne pas implanter un AMAS si le système en devenir ne le nécessite pas. En effet, si l'adéquation aux AMAS n'est pas établie en phase d'analyse, la phase de conception suivra le déroulement du processus unifié et pourra s'appuyer sur les phases préliminaires sans avoir à y apporter de modifications. En revanche, dans le cas où un AMAS apparaît comme une solution d'implantation préférable, il convient d'utiliser des outils spécifiques pour la conception et le développement. Utiliser un langage spécifique à cette étape n'implique pour autant pas d'ignorer les modèles UML réalisés durant les phases d'expression des besoins.

7.1.4 Automatisation

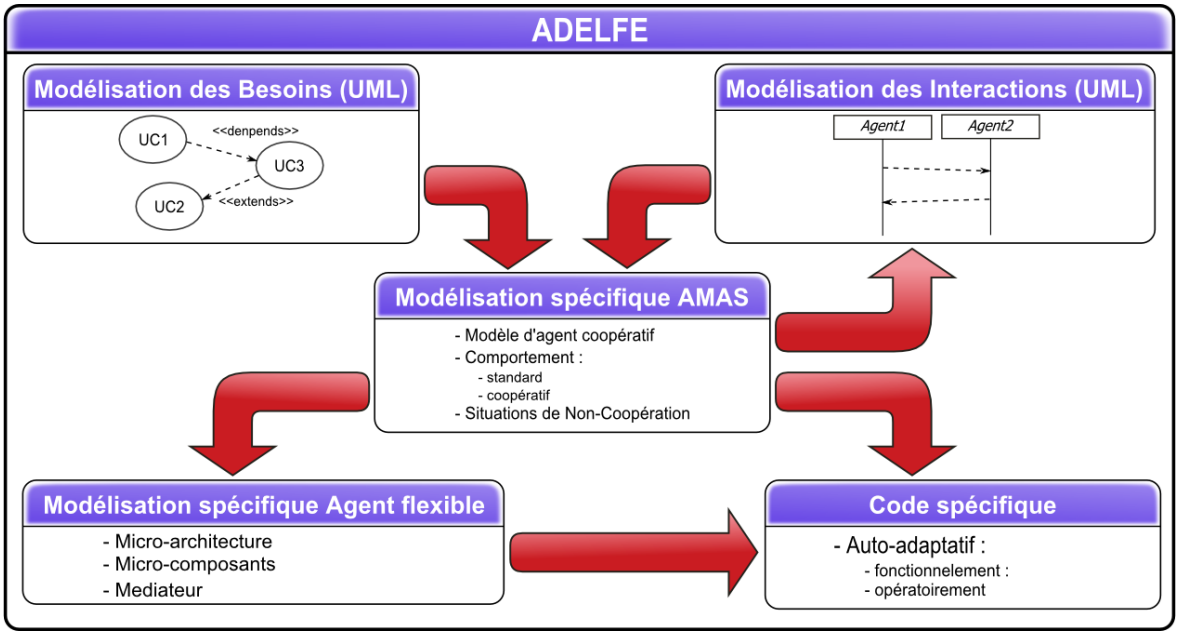

Figure 7.1 — Intégration automatique de DSMLs et GPMLs dans ADELFE.

Nous considérons que le développement de logiciel dirigé par les modèles et les outils qui y sont liés, de part leur maturité, autorisent une vision plus large de l'ingénierie du logiciel qu'elle ne le fût jusqu'alors. Comme nous venons de le voir, il paraît tout à fait logique d'utiliser le langage le plus approprié à la tâche à réaliser (UML pour l'établissement des besoins, AMAS-ML pour la conception : voir figure 7.1). L'IDM nous rend possible la définition de langages adaptés à des domaines précis, de les lier par transformation les uns aux autres et de les intégrer au sein d'une même méthode. Cela permet, en outre, de répondre aux préoccupations et aux besoins des concepteurs, des développeurs et du domaine (dans notre cas de systèmes multi-agents adaptatifs).

Pour mener à bien notre objectif de produire des systèmes auto-adaptatifs, nous pouvons bénéficier des transformations afin de combiner la théorie des AMAS au modèle d'agent flexible. Ainsi, nous permettons l'implantation du système modélisé, en suivant le processus ADELFE, sur une plate-forme d'agent spécialisée et adaptée à ses besoins. Pour atteindre ce but, il est nécessaire d'abstraire à la fois la théorie des AMAS et le modèle d'agent flexible.

Dans une démarche dirigée par les modèles, ceci se traduit par la définition de méta-modèles (cf. paragraphe 4.2.1.2), grâce auxquels il sera possible d'édifier des ponts technologiques.

7.2 ADELFE 2.0

ADELFE 2.0 reprend l'ensemble des activités de la méthodologie ADELFE (cf. paragraphe 5.8) auxquelles nous apportons un ensemble d'extensions. La phase de conception d'ADELFE était auparavant effectuée à l'aide d'un profil UML 1.4, pour tenir compte des agents coopératifs et de leurs spécificités. En outre, les AIP d'AUML ont été étendus afin d'intégrer les principes de la coopération. Toutefois, depuis sa dernière version, UML2.0 (omg, 2003d) a intégré bon nombre des caractéristiques souhaitées de la FIPA pour AUML, se rapprochant ainsi des besoins liés à la définition d'AIP. La version 2.0 de l'UML (omg, 2003d) propose, par exemple, le concept de *CombinedFragment* et l'intègre au diagramme de séquence. Ce concept permet de représenter des structures de contrôle sur les échanges de messages (alternative, séquence, boucle, etc.).

7.2.1 UML 2.0 dans ADELFE

L'UML constitue un langage générique bien qu'il ait été initialement défini pour supporter le développement d'applications orientées objet (Booch et al., 1999) ; son utilisation actuelle s'est étendue avec l'avènement du modèle objet. Il est utilisé pour la conception d'applications web, embarquées ou critiques, etc. On ne peut pas le considérer comme dédié au *« monde Objet »* , celui-ci est trop vaste pour constituer un domaine précis. UML fut défini, comme son nom l'indique, pour unifier différentes notations du monde objet (Use Cases (Jacobson, 1995), OMT (Rumbaugh, 1995), etc.). Il est toutefois possible de définir des dialectes de ce langage unifié afin qu'ils correspondent aux besoins spécifiques de certains domaines, c'est la notion de profil. L'OMG propose de nombreux profils [1] liés à des domaines ou des technologies spécifiques, par exemple pour le standard CORBA, ou pour les systèmes temps-réels critiques (MARTE), etc.

UML dans sa version actuelle propose treize diagrammes (omg, 2007b) dont nous ne présenterons pas l'intégralité ici. Cependant, un certain nombre d'entre eux sont utilisés dans le cadre de la méthodologie ADELFE ; c'est à ces derniers que nous nous attachons ici.

7.2.1.1 Modélisation des interactions

Les premières phases de la méthodologie permettent de déterminer les acteurs et entités qui sont impliqués dans l'activité du système en devenir. Les diagrammes d'interactions UML, *collaboration* et *séquence*, sont utilisés afin de préciser les relations entre ces divers

1. `http://www.omg.org/technology/documents/profil_catalog.htm`

éléments. Les entités sont qualifiées d'*actives* ou de *passives* en fonction de leur activité supposée. Les interactions entre entités actives et système sont exprimées grâce au *diagramme de séquence*, il s'agit en effet de décrire des échanges de messages pour lesquels ce diagramme est conçu. Quant aux flots de données provenant des entités passives, ils sont décrits au moyen de *diagrammes de collaboration*. Les diagrammes de séquence sont aussi utilisés pour préciser à la foi les scénarios d'utilisation du système et les relations entre les entités du système.

7.2.1.2 Modélisation des besoins

Le processus d'ADELFE met en œuvre plusieurs types de diagrammes UML pour la description et l'établissement des besoins. Cet usage de l'UML est issu du processus unifié sur lequel il se base. Les *cas d'utilisation* permettent de décrire les besoins fonctionnels, ils regroupent des scénarios d'utilisation et déterminent qui en sont les acteurs. UML propose un diagramme afin de visualiser ces *cas d'utilisation* et leurs relations (inclusion, extension, etc.). La détermination de ces fonctionnalités est capitale pour le processus unifié, et par conséquent pour ADELFE, puisqu'elles guident l'ensemble des activités de conception et de développement qui en découle.

7.2.1.3 Stéréotypes

UML propose les concepts de stéréotype et de profil pour permettre sa spécialisation. La notion de stéréotype peut être assimilé à un étiquetage des méta-classes de l'UML afin de leur associer une sémantique différente. Les profils quant à eux peuvent être vus comme des méta-modèles à part entière se basant sur celui de l'UML afin de bénéficier de concepts présents dans ce dernier. Un profil étend le méta-modèle UML en y ajoutant des concepts, de nouvelles règles de bonne formation (OCL), une sémantique différente pour des concepts existants, etc. Dans le cadre de notre étude, nous avons redéfini quelques stéréotypes permettant simplement d'étiqueter des éléments de modélisation UML 2.0 afin de les identifier pour la suite de la conception. Il s'agit essentiellement de distinguer les entités actives des passives et de discriminer les agents coopératifs. Ces stéréotypes proviennent de ceux proposés par le profil UML 1.X qui fut développé dans le cadre de la méthodologie ADELFE (Picard, 2003). Nous considérons les stéréotypes suivant :

- *«PassiveEntity»* ou *«ActiveEntity»* : peuvent s'appliquer sur une classe ou un acteur UML,
- *«CooperativeAgent»* : peut s'appliquer sur une classe uniquement,
- *«CooperationFailure»* : permet de discerner des interactions potentiellement non coopératives, il peut s'appliquer sur une association (dans le diagramme de cas d'utilisation) ou sur un message (diagramme de séquence).

Ces quelques stéréotypes permettent de mener à bien les premières tâches spécifiques de la méthodologie ADELFE et facilitent d'autant le passage à une modélisation dédiée.

Dans le processus ADELFE, le langage de modélisation générique UML est utilisé pour les premières phases. La plupart des activités qu'elles impliquent ne sont pas spécifiques aux principes des AMAS (cf. paragraphe 6.2.5). Nous considérons que pour l'ensemble de ces tâches, l'UML reste parfaitement adéquat. En effet, aucune tâche ne nécessite réellement de modélisation particulière jusqu'à ce que l'adéquation du système avec la théorie des AMAS ne soit établie. L'UML apparaît comme le langage le plus adapté aux premières phases d'ADELFE ; en outre elles sont directement issues du processus unifié qui en implique l'utilisation.

7.2.2 Formalisation SPEM 2.0

Dans la version 2.0 d'ADELFE, la phase de conception a été enrichie par l'utilisation d'AMAS-ML et une nouvelle phase d'implémentation a été ajoutée afin de bénéficier de l'approche dirigée par les modèles. Pour décrire formellement ces extensions et l'ensemble du processus obtenu, nous avons fait le choix d'utiliser le nouveau standard de l'OMG SPEM (omg, 2007a). D'un point de vue technique, nous avons redéfini le processus ADELFE à l'aide du plugin EPF (Eclipse Process Framework). Il fournit un éditeur pour spécifier des modèles de processus conformes au SPEM 2.0 et permet, entre autres, de bénéficier de ses mécanismes de réutilisation. SPEM 2.0 et par conséquent EPF, utilise le concept de *Library*, bibliothèque contenant des méthodologies (*MethodPlugin*), à l'intérieur desquelles tout ou presque est réutilisable, de la simple tâches (*task*) jusqu'au processus dans son intégralité (*DeliveryProcess*). En outre, EPF propose un outil de publication permettant de générer automatiquement à partir du modèle défini un guide méthodologique sous la forme d'un site internet. Cette fonctionnalité permet ainsi de mettre à jour automatiquement la documentation liée au processus, dès lors que le modèle subit des modifications. La figure 7.2 donne un exemple de cette formalisation et représente la phase d'analyse d'ADELFE à l'aide des concepts et des icônes proposées par le plugin de base de SPEM 2.0. Par la suite, les figures représentant les différentes phases que nous présentons utilisent elles aussi les notations et les concepts proposés par le standard de l'OMG.

7.2.3 Enrichissement de la phase de conception

AMAS-ML est utilisé dans plusieurs étapes de la phase de conception, de la conception détaillée de la structure de l'agent à la définition de son comportement coopératif. L'ensemble de ces tâches de conception sont représentées dans la figure 7.3. Le modèle du système est réalisé grâce à des diagrammes spécialisés :

- **Diagramme agent** : il est utilisé pour modéliser la structure d'un agent coopératif, ainsi que ses relations avec les entités de l'environnement (identifier lors de l'établissement des besoins). Il définit toutes les spécificités des agents coopératifs : leurs *représentations, caractéristiques*, leurs *compétences* et leurs *aptitudes*.
- **Diagramme de règles de comportement** : il permet la spécification de règles impli-

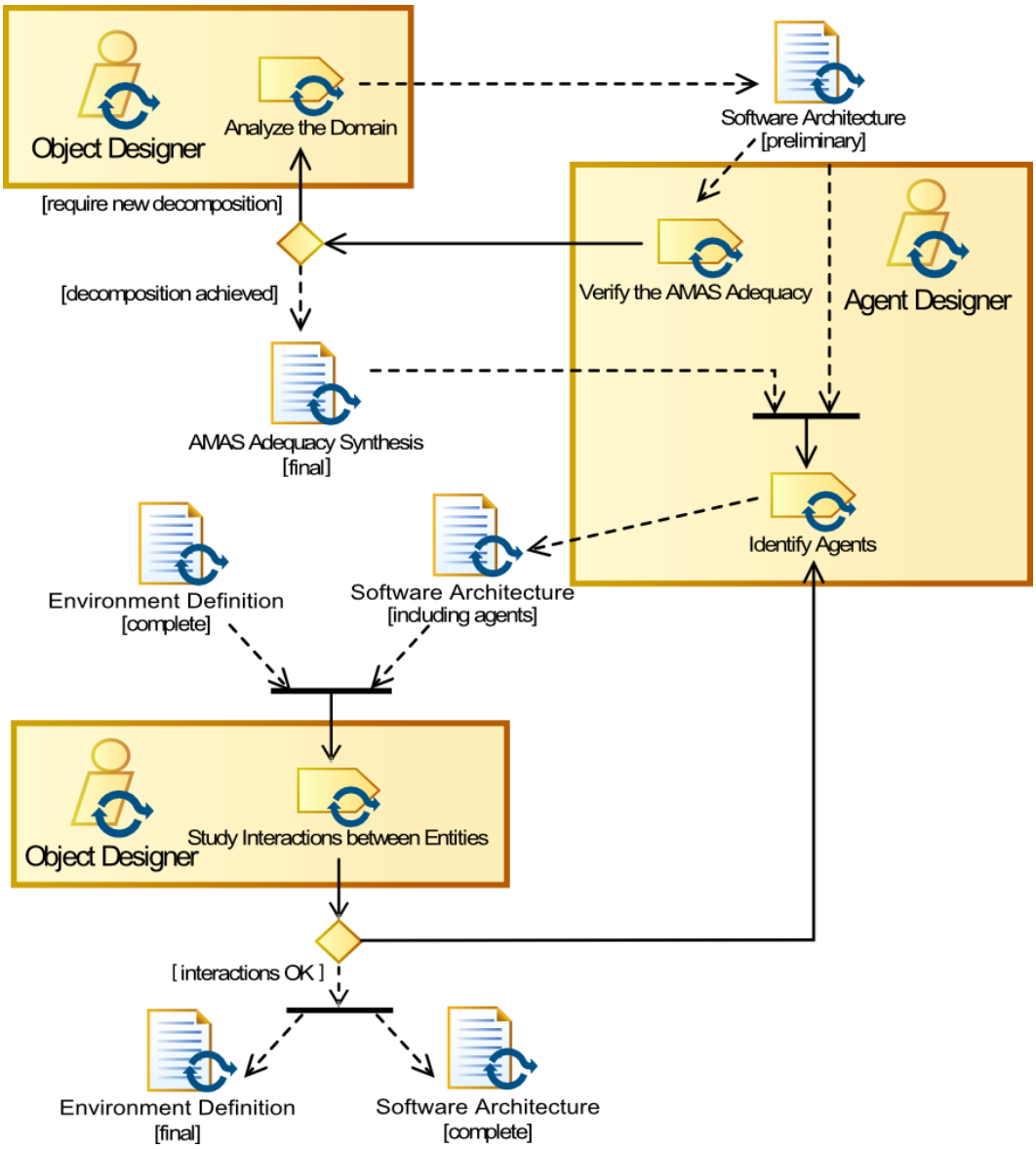

Figure 7.2 — Phase d'analyse d'ADELFE 2.0.

quées dans le processus de décision d'un agent. Ces règles sont exprimées en fonction des caractéristiques des agents. Sur la base de leurs connaissances et de leur perceptions, les agents doivent décider des prochaines actions à mener. Ces actions peuvent être réalisées dans le but de palier une SNC (*CooperativeRule*) ou non (*StandardBehaviorRule*).

– **Diagramme d'interaction** : il correspond à un diagramme de séquence UML 2.0. Nous avons défini une transformation qui nous permet d'intégrer les protocoles et les messages définis dans le modèle UML dans notre modèle AMAS-ML, afin de les faire apparaître en tant que *CommunicationAction* et *CommunicationPerception* (cf. paragraphe 1).

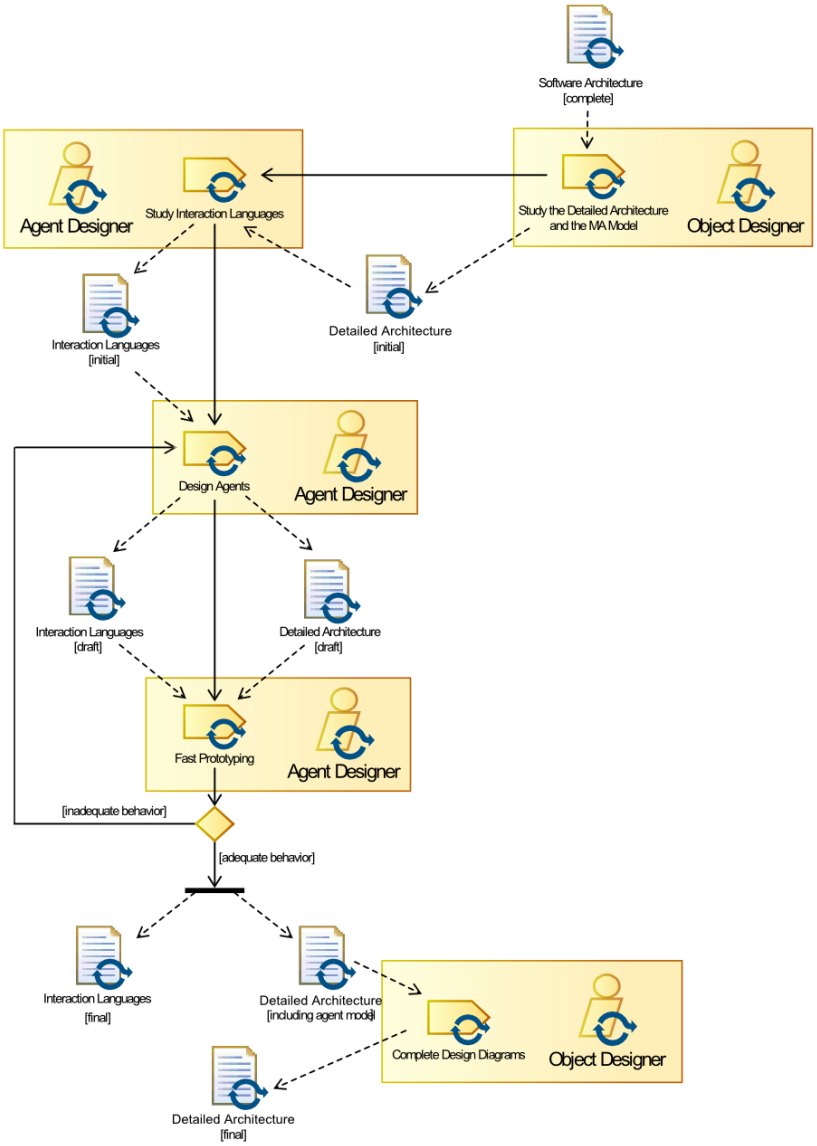

Figure 7.3 — Phase de design d'ADELFE 2.0.

La prochaine phase d'implémentation dépend du résultat de cette conception, c'est-à-dire du modèle AMAS-ML.

7.2.4 Ajout de la phase d'implémentation

Comme nous l'avons présenté dans (Rougemaille et al., 2007), cette phase est guidée par une idée principale : la séparation des préoccupations. Plus précisément, nous voulons allier les capacités du modèle d'agent coopératif des AMAS à celles du modèle d'agent flexible. Pour ce faire, nous devons séparer ce qui est du domaine des mécanismes de base de l'agent (niveau opératoire), du comportement (la manière dont les agents utilisent leurs outils pour atteindre leurs buts locaux). Cette phase est en partie basée sur l'outil de modélisation et de génération que nous avons développé : MAY (Make Agents Yourself). Il intègre le langage

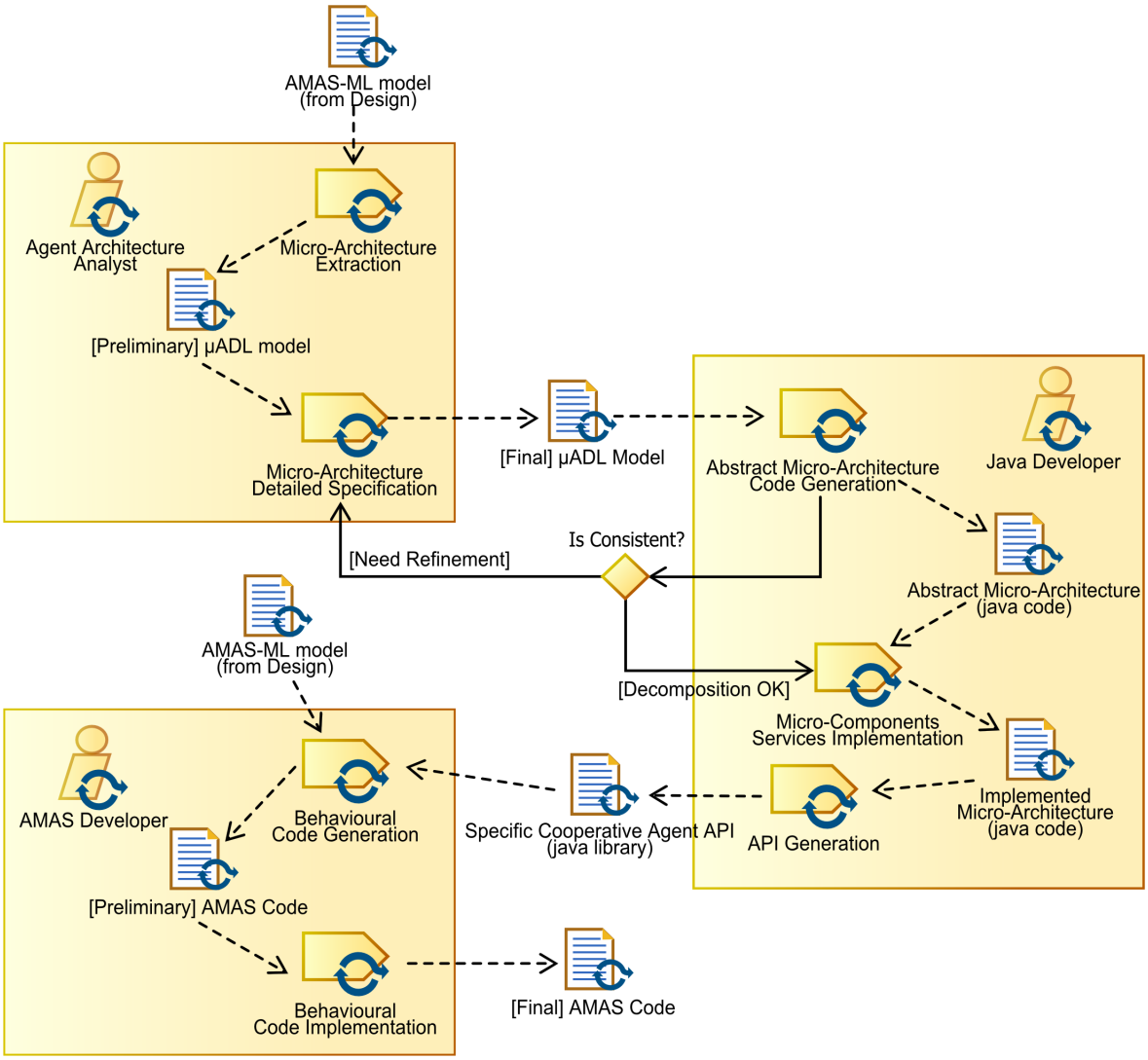

Figure 7.4 — Phase d'implémentation d'ADELFE version 2.0.

spécifique μADL (présenté dans le paragraphe 8.1.1) qui permet la description de micro-architectures d'agents (représentation des mécanismes opératoires). Le style architectural (assemblage de micro-composants) et le processus de génération MAY produisent une sorte de *« machine abstraite d'agent »* (nous la considérons comme une API). Elle peut être utilisée par les développeurs comme une couche d'abstraction pour implanter le comportement des agents d'un système. Cette phase comporte plusieurs générations ou transformations de modèles qui sont représentées dans la figure 7.4 :

– *Extraction de la micro-architecture* : c'est la première étape de transformation de modèles. Elle met en relation AMAS-ML avec μADL et a été mis en œuvre avec ATL (Atlas Transfmorming Language) (Jouault and Kurtev, 2005). Cette transformation facilite la tâche de l'*analyste d'architecture agent* par la création d'un modèle μADL contenant les éléments du modèle AMAS-ML que nous considérons comme des mécanismes opératoires (voir le paragraphe 9.3).

- *Génération du code de la micro-architecture abstraite* : cette étape de génération produit un squelette de code pour chaque micro-composant, ainsi que le code complet du *médiateur* et du *QuasiBehavior*. Cette classe abstraite implante l'accès aux services comportementaux. Le code du comportement en étendant cette classe récupère l'ensemble des services fournis par différents micro-composants et définis comme étant de niveau comportemental dans le modèle μADL. Une fois que le modèle de l'architecture est considéré suffisamment précis, le *développeur Java* a en charge l'implémentation des services fournis par les micro-composants.
- *Génération de l'API* : à ce niveau MAY génère l'ensemble de l'API, c'est-à-dire la micro-architecture complète (micro-composant, médiateur, etc.) intégrée au sein d'un ensemble d'outils pour exécuter, créer et déployer le type d'agent spécifique qui a été modélisé. L'ensemble de ces outils génériques (ne dépendant pas de l'agent modélisé) est appelé plate-forme d'accueil, et permet d'intégrer plusieurs types d'agents spécifiques.
- *Génération du code du comportement* : à partir des règles comportementales exprimées dans la phase de conception avec AMAS-ML, nous proposons de générer un squelette de code pour faciliter la tâche des développeurs AMAS. L'objectif est de fournir les bases pour implanter les processus de décision de chaque agent dans le système.

Nous avons présenté les extensions que nous souhaitons apporter au processus d'ADLEFE. Ces extensions ont nécessité d'une part, la définition de nouveaux langages de modélisation spécifiques (AMAS-ML et μADL) et d'autre part, l'implémentation d'un ensemble de traitements pour automatiser leur intégration à ADELFE et la génération de code. Ces deux aspects sont exposés de manière détaillée dans les deux chapitres suivants.

8 Langages de modélisation

LES modèles représentent les citoyens de première classe dans une approche dirigée par les modèles. Dans ce cadre, les langages de modélisation permettant de les décrire prennent une importance considérable. Ce chapitre présente le cœur de notre contribution, c'est-à-dire, les langages de modélisation que nous souhaitons utiliser pour atteindre notre objectif : décrire des applications capables de faire face à de fortes contraintes d'adaptation, en combinant les approches SMA adaptatifs et agent flexible. Nous avons vu dans le chapitre précédent qu'une méthode de développement a déjà été mise en place pour faciliter la conception des AMAS : ADELFE. Cependant, il reste à répondre à certains besoins, notamment en terme de modélisation dédiée aux agents coopératifs. Ces différents langages de modélisation sont présentés dans ce chapitre. En premier lieu, nous présentons notre classification de l'adaptation, puis les langages spécifiques qui en découlent. Ces langages AMAS-ML et μADL ont été développés grâce à une démarche de méta-modélisation (cf. paragraphe 4.2.2).

8.1 Une classification de l'adaptation

Nous proposons une caractérisation de l'adaptation se basant sur l'étude des caractéristique des systèmes complexes et des modèles d'environnements définis dans le cadre des SMA (Weyns et al., 2004). En fonction des deux niveaux de l'environnement que nous avons caractérisé au paragraphe 1.3 et en tenant compte du caractère *multi-échelle* des systèmes étudiés, nous proposons les dimensions suivantes pour qualifier l'adaptation :

1. **Structurelle** Il s'agit d'une re-configuration du système. Cette modification de la structure du système est liée à une perturbation (changement d'état de l'environnement) perçue par le système et est vouée à maintenir ou améliorer la fonction de celui-ci. On peut considérer comme exemple un re-déploiement dynamique pour améliorer les performances du système.
2. **Fonctionnelle** Il s'agit d'une ré-organisation du système afin de modifier sa fonctionnalité pour satisfaire un objectif ou une requête de l'environnement. Les constituants du système modifient leur comportement ou leurs interactions adaptant ainsi la fonction du système.
3. **Opérationnelle** La fonctionnalité du système est maintenue, mais les moyens mis en œuvre pour la réaliser sont adaptés, en raison de modifications de l'infrastructure du

système (environnement physique), comme par exemple dans le cas d'un changement de communication lié à la mobilité d'un dispositif.

4. **Dynamique/statique** Cette dimension est orthogonale aux précédentes. Il s'agit de définir les moments auxquels interviennent les modifications de la structure, de la fonction ou de la manière de rendre cette fonction. Ont-elles lieu à l'exécution, à la conception ou la maintenance ? En d'autres termes, s'agit-il de re-configuration dynamique ou statique du système ?

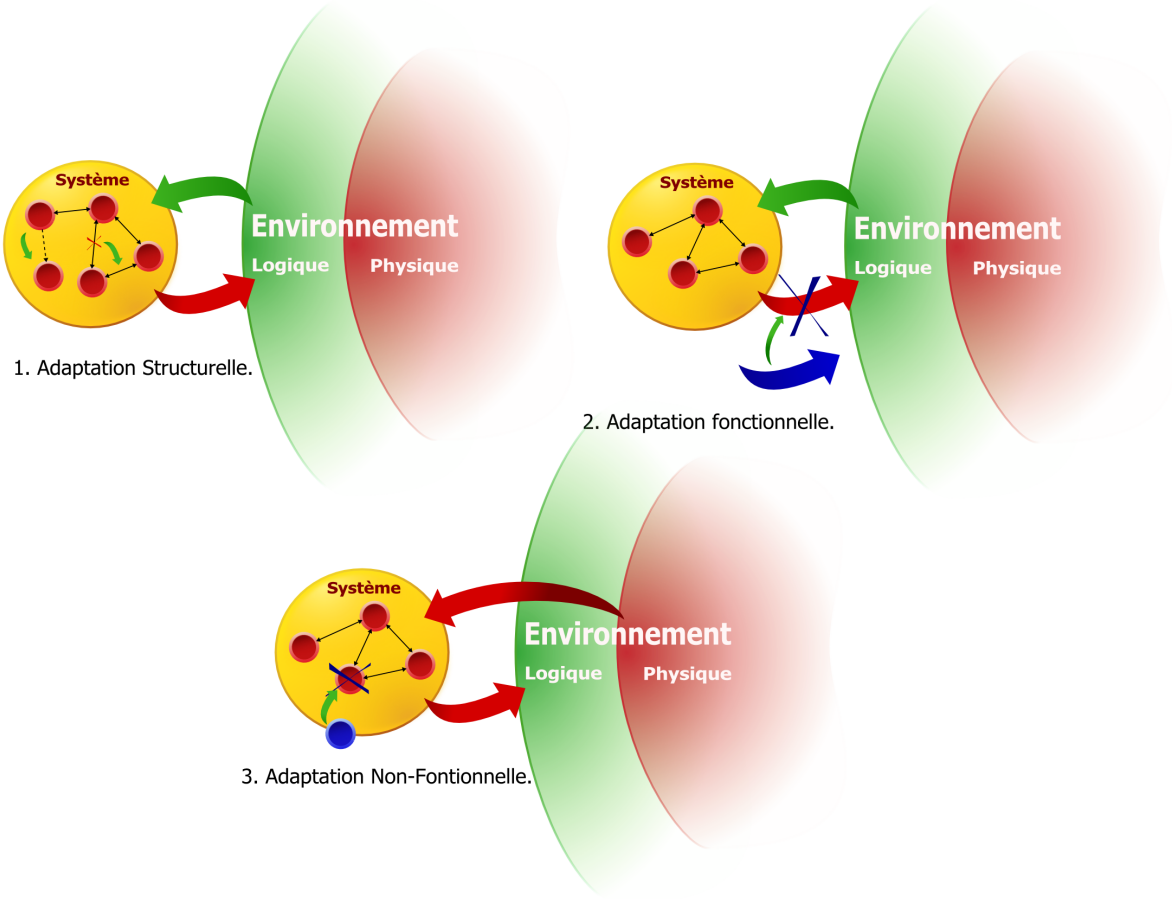

Figure 8.1 — Adaptation dynamique des systèmes à leur environnement (représentation inspirée par (Maturana and Varela, 1994))

La maîtrise de la complexité dans les systèmes informatiques que nous avons entrevue dans le paragraphe précédent, passe par une maîtrise de l'adaptation du système au niveau global et de ses constituants. Comme nous l'avons vu, il est capital pour mener à bien cette adaptation qu'un système possède des facultés de perception de son contexte, de re-configuration, de décentralisation du contrôle, etc.

En combinant les approches SMA adaptatif et agent flexible, nous pouvons décrire des systèmes capables de maîtriser l'auto-adaptation selon trois axes :

– **L'axe système/agent** : au niveau du système, l'auto-organisation est mise en oeuvre

via la coopération des agents et l'auto-adaptation des agents grâce à l'architecture flexible.

– **L'axe fonctionnel/opératoire** : la coopération des agents tend à faire émerger une *fonction adéquate* pour le système (adaptation fonctionnelle). Chaque agent peut adapter ses *mécanismes opératoires* afin de satisfaire de nouvelles contraintes de son *environnement physique*. Cette adaptation locale peut se propager à l'ensemble du système par le biais de la *coopération*.
– **L'axe conception/exécution** : prise en compte de l'auto-adaptation dès la conception et mise en pratique à l'exécution.

Nous pouvons de cette manière atteindre les degrés d'adaptation que nous avons identifiés dans le paragraphe 1.3. La figure 8.2 présente les approches qui interviennent pour la mise en pratique de l'adaptation à l'exécution selon les deux premiers axes.

	Fonctionnelle	Opératoire
Système	Approche AMAS	Approche AMAS Agent flexible
Agent	Apprentissage	Agent flexible

Figure 8.2 — Axes d'auto-adaptation et mise en œuvre.

8.1.1 Vers la définition de langages spécifiques

Les langages de modélisation spécifiques permettent de fournir un moyen d'abstraction adapté à un domaine précis. Ils sont basés sur des méta-modèles qui permettent de décrire dans un cadre semi-formel les concepts précis d'un domaine et de caractériser les relations entre ces concepts (cf. paragraphe 4.2.1.2). Ils apportent de plus, une garantie quant au respect de la structure qu'il définisse au sein des modèles qui s'y conforme. Le paragraphe 4.2.2 a exposé le processus de définition de ces langages dans le cas général ; ici nous présentons la définition de deux langages de modélisation dédiés, en premier lieu à l'approche AMAS (AMAS-ML) et en second lieu au modèle d'agent flexible (μADL). Ces deux langages interviennent dans le processus de développement dirigé par les modèles que nous souhaitons définir. Tout d'abord en permettant une abstraction du système et de ses constituants suivant les principes de l'approche AMAS. Enfin, en permettant la définition d'un type d'agent spécifique possédant les caractéristiques opératoires nécessaires à leur mise en pratique sur une plate-forme dédiée. Ces deux langages et leur méta-modèles respectifs sont présentés plus en détail dans les chapitres suivants.

8.2 AMAS-ML : un langage spécifique aux AMAS

Il s'agit, comme son nom l'indique, AMAS Modeling Language, d'un langage de modélisation dédié aux systèmes multi-agents adaptatifs. Son objectif est de proposer à son utilisateur les moyens de spécifier les concepts essentiels des AMAS, à savoir : les agents, leurs propriétés, leurs interactions avec l'environnement et les règles qui régissent leur comportement coopératif. Ces concepts et leur relations doivent apparaître dans la syntaxe abstraite du langage ou, en d'autres termes, dans le méta-modèle. Le méta-modèle AMAS-ML constitue une description semi-formelle des principes des AMAS qui permet de les décrire à des degrés de précision différents. Il permet de définir, aussi bien, des relations entre agents, entres agents et entités que de décrire la manière dont une caractéristique particulière d'un agent intervient dans son processus de décision. Cela permet de définir un modèle du système très précis et prenant en compte la notion de coopération et par conséquent, d'auto-organisation (cf. paragraphe 6.1).

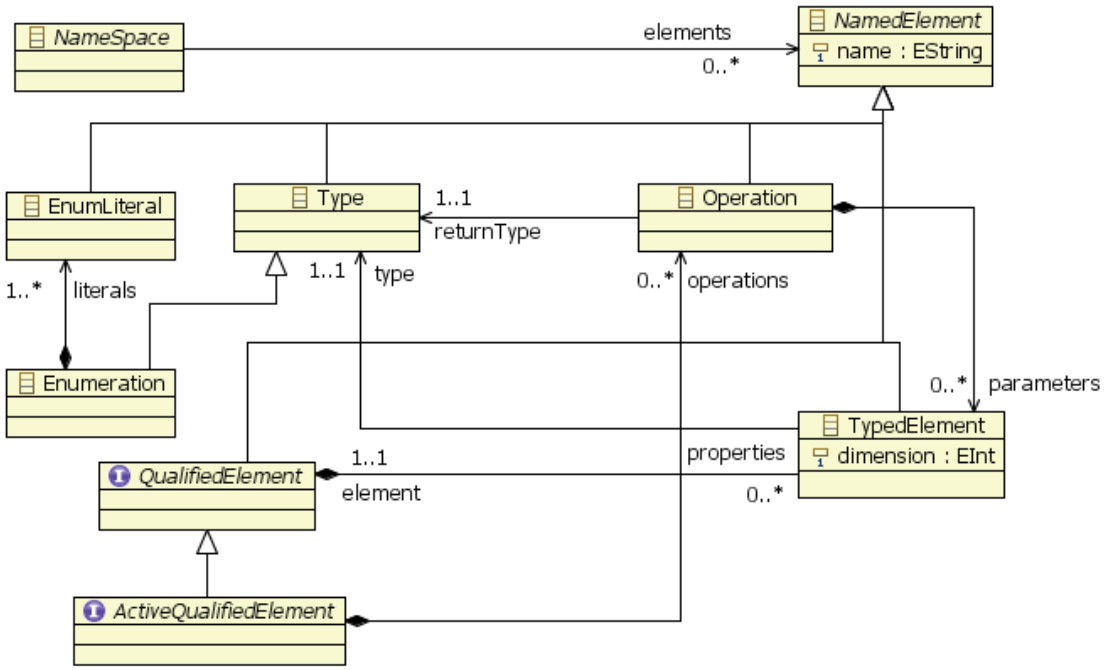

Figure 8.3 — Package core du méta-modèle AMAS-ML

8.2.1 Méta-modèle

Les éléments du méta-modèle sont regroupés en point de vue (*Paquetage*) en fonction des propriétés des AMAS qu'ils modélisent. Techniquement parlant, ce méta-modèle a été réalisé grâce au langage de méta-modélisation Ecore du *plugin* EMF (Eclipse Modeling Framework) d'Eclipse. Ceci, nous permet de bénéficier de nombreux outils pour décrire la syntaxe

concrète du langage correspondant. En outre, il permet une représentation standard des modèles sous la forme de fichiers XMI (XML Model Interchange) garants de l'inter-opérabilité au sein des *plugins* orientés modèle de la plate-forme Eclipse.

8.2.1.1 Paquetage core

A la différence des autres paquetages ou points de vue, ce paquetage regroupe des concepts qui ne sont pas liés au domaine des AMAS mais davantage à l'utilisation souhaitée du langage que l'on décrit. Il s'agit de concepts utilitaires génériques qui seront spécialisés par les autres points de vue (voir figure 8.3). Ce paquetage contient les méta-classes permettant de décrire des éléments nommés (*NamedElement*), des types (*Type*), des propriétés typées (*TypedElement*), des opérations (*Operation*), leurs paramètres (*parameters*) et des éléments susceptibles d'être caractérisés par ces propriétés ou opérations (*QualifiedElment, ActiveQualifiedElement*).

8.2.1.2 Point de vue environnement/système

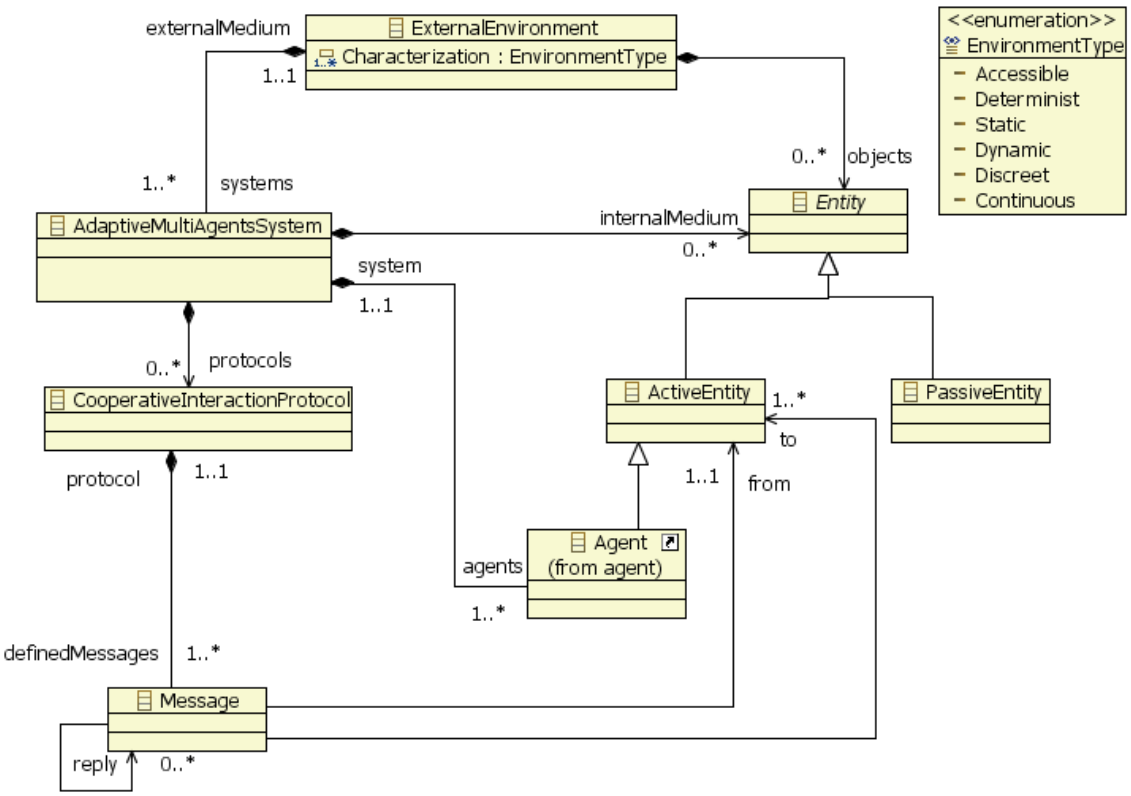

Figure 8.4 — Point de vue système du méta-modèle AMAS-ML

Un système multi-agent adaptatif (*AdaptiveMultiAgentSystem*) est plongé dans un environnement (*ExternalEnvironment*) dont on veut pouvoir représenter les propriétés utiles à la

conception du système lui-même. En effet, les entités (*Entity*) situées dans cet environnement interagissent avec le système et plus particulièrement avec les agents de ce système par un jeu de perceptions et d'actions (voir paragraphe 8.2.1.3). Nous avons définit deux types d'entités active (*ActiveEntity*) et passive (*PassiveEntity*). Parmi les entités actives nous distinguons particulièrement les agents (*Agent*) qui font preuve d'autonomie dans leur décision et de coopération. Ces concepts permettent d'insister sur le couplage entre système et environnement qui constitue un principe clef des AMAS. C'est, entre autres, à cause de cette interaction que l'on parle de système « adaptatif ». Un système peut contenir des objets, au sens large, n'étant pas des agents (*elements*). Ils constituent le milieu intérieur du système et interagissent avec ses agents. Les agents (*Agent*) d'un AMAS sont voués à coopérer pour faire émerger la fonction globale du système, une action primordiale pour mettre en œuvre cette coopération est la communication. Celle-ci est rendue possible soit au travers d'envois de messages (*Message* : communication explicite, directe) prenant part à des protocoles précis (*CooperativeInteractionProtocol*), soit par des jeux d'interactions sur les éléments de leur environnement (communication indirecte). Ces deux modes de communication ne sont pas exclusifs.

8.2.1.3 Point de vue agent

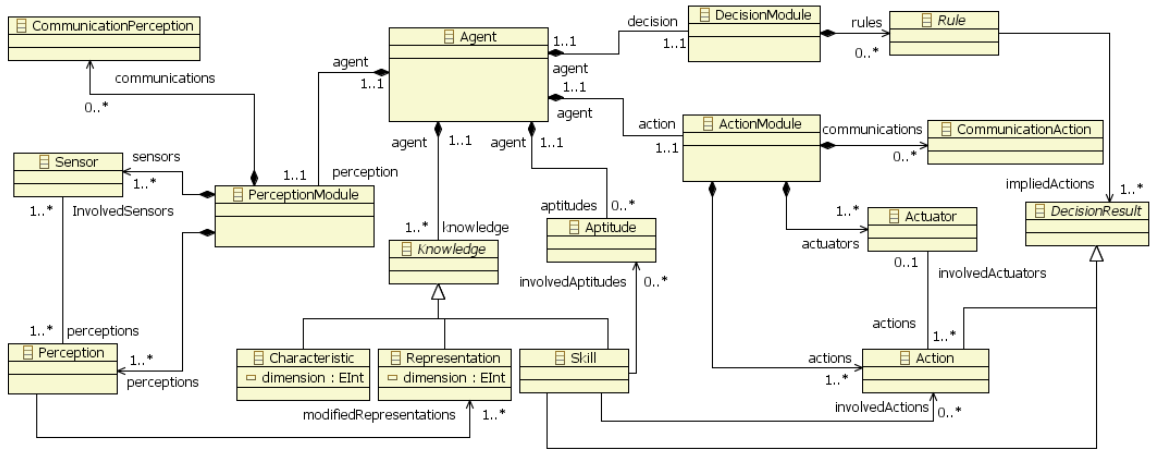

Figure 8.5 — Point de vue agent du méta-modèle AMAS-ML

Nous définissons un agent coopératif à partir de plusieurs modules, chacun prenant en charge une partie de ses activités et de son cycle de vie. Nous avons vu dans le paragraphe 6.1.3 que le cycle de vie d'un agent coopératif est défini selon trois phases : *perception*, *décision* et *action*. Nous dissocions ainsi les concepts entrant en jeu dans chacune de ces phases au sein de modules différents (voir figure 8.5), afin de clarifier et de préciser quels sont les éléments qui interviennent dans le déroulement de chaque phase du cycle de vie. De cette façon, à partir des trois phases pré-citées et de leurs besoins respectifs, nous avons défini les concepts suivants :

- **PerceptionModule** : il contient les différentes perceptions (*Perception*) qu'un agent peut avoir de son environnement, plus précisément des entités qui le peuplent. Un agent peut posséder un dispositif physique ou logique *Sensor* pour mener à bien cette récupération d'information (*involvedSensor*). En outre, un agent coopératif peut percevoir les messages qui lui sont destinés (*CommunicationPerception*).
- **ActionModule** : il représente les actions qu'un agent peut mener dans son environnement ainsi que les moyens mis à disposition pour les réaliser (*Actuator*). On distingue un type d'action particulier (*CommunicationAction*) qui consiste à interagir directement avec d'autres agents du système par émission de messages (*Message*) dont le protocole est défini au niveau du système (*CooperativeIteractionProtocol* voir paragraphe 8.2.1.2).
- **DecisionModule** : ce module défini les règles comportementales (*Rule*) qui permettent à un agent de déterminer, en fonction de ses connaissances (*Knowledge*) l'action à mener *DecisionResult*. En vue de palier une situation non coopérative éventuellement détectée ou de réaliser son comportement nominal, ces aspects sont détaillés dans le point de vue coopération (voir 8.2.1.4).
- **Knowledge** : regroupe l'ensemble des connaissances d'un agent : ses représentations du milieu qui l'entoure ainsi que de lui-même (*Representation*), ses capacités (*Skill*) et ses caractéristiques intrinsèques (*Characteristic*) éventuellement visibles (*isPerceptible*) pour les autres agents du système ainsi que ces aptitudes (Aptitude). Ces dernières représentent les outils génériques à la disposition de l'agent, comme par exemple un tirage aléatoire, des calculs statistiques, etc. Tout ce qui ne concerne pas directement le domaine de l'agent mais qui est utilisé pour réaliser, mettre en œuvre ses compétences.

8.2.1.4 Point de vue coopération

Ce paquetage contient les éléments permettant de définir les règles comportementales des agents. Nous avons définit ces règles comme un ensemble d'actions à mener pour régler une situation donnée (coopérative, ou non). Une situation particulière est décrite comme une valeur d'état de l'agent. L'état d'un agent coopératif (*CooperativeAgentState*) est déterminé en fonction d'une expression « logique » (*Condition*) définie sur ses connaissances (*StateVariable*). Chaque condition est composée d'opérateurs logiques n-aires (*LogicalOperator*) ou unaires (*UnaryCondition*). Ces opérateurs permettent définissent des conjonctions ou des disjonctions entre des expressions de comparaison de variables d'état (*StateExpression*). Ainsi, un état correspond à une condition particulière. Il définit le comportement de l'agent et par conséquent le type d'action que celui-ci devra mener (cf. paragraphe 8.2.1.3 *DecisionResult*). Chaque état constitue la prémisse d'une règle comportementale qu'elle soit coopérative ou non (référence *trigger*).

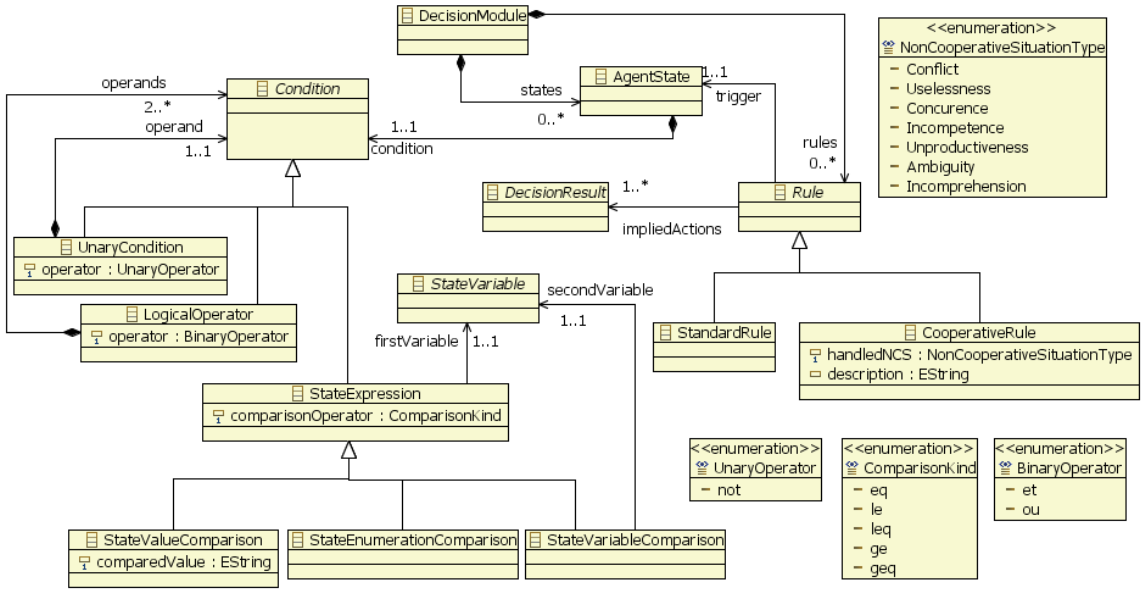

Figure 8.6 — Point de vue coopération du méta-modèle AMAS-ML

8.2.2 Syntaxe du langage

AMAS-ML propose trois diagrammes différents permettant de concevoir un agent coopératif et son comportement. Chacun de ces trois diagrammes correspond à un point de vue distinct du méta-modèle.

8.2.2.1 Diagramme agent

Le diagramme agent permet de décrire des agents coopératifs et l'ensemble de leurs caractères spécifiques (connaissances, outils d'interaction, etc.). Il représente un agent sous la forme d'un rectangle contenant un compartiment spécifique pour chaque type de connaissance, ainsi qu'un compartiment spécifique à ses aptitudes. Un agent est lié à ses modules d'interaction (*action* et *perception*) qui regroupent l'ensemble des actions et des perceptions permettant de capter et de gérer l'environnement, ainsi que les communications inter-agent. Ces interactions sont mise en pratique grâce à des actionneurs et des capteurs qui permettent de percevoir ou d'agir sûr différentes entités de l'environnement. La figure 8.7 représente un exemple d'utilisation du diagramme agent du langage AMAS-ML. Le rectangle situé sur la gauche de la figure représente l'agent *unAgent* et ses propriétés dans des compartiments différents. Il possède une seule capacité *skill1()*, une caractéristique *carac1*, une représentation *representation1* et une seule aptitude *aptitude1*. Il est capable de percevoir l'entité *Entity2* par le biais du capteur *capteur1* qui permet de mener à bien la perception *perception1()*. Symétriquement, l'actionneur *actionneur1* permet à l'agent d'utiliser son action *action1()* sur l'entité *Entity1*. Les actionneurs et les actions, respectivement les capteurs et les perceptions, sont contenus par le module d'action, respectivement de perception. Ces modules regroupent

,en outre, les actions et perceptions de communication (*«communications»*), bien que l'agent présenté dans la figure ne possède pas de capacité de communication directe. Ce diagramme représente aussi la définition d'un type de donnée *Type1* qui peut être utilisé pour typer une caractéristique de l'agent ou les paramètres d'une action, par exemple.

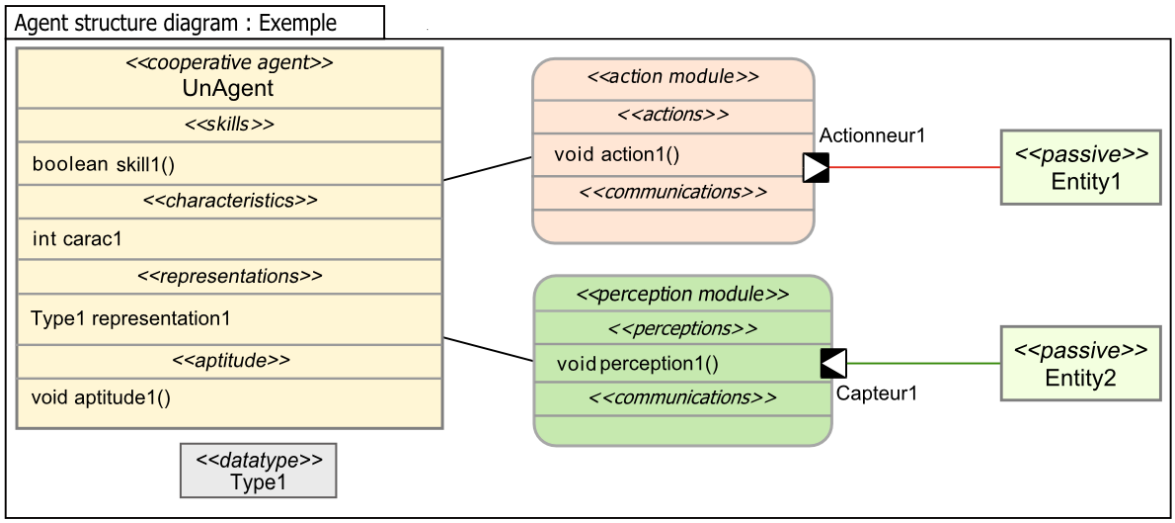

Figure 8.7 — Diagramme agent AMAS-ML.

8.2.2.2 Diagramme système agent

Il s'agit d'un diagramme général, qui permet de représenter le système et les différents types d'agent qui doivent y participer, ainsi que les entités (passives ou actives) avec lesquels ils interagissent. Il permet de représenter les interactions entre les entités et les agents sous forme de flèches. L'origine de cette flèche détermine l'entité ayant l'initiative des interactions. Par exemple, la figure 8.8 représente un agent coopératif (*Agent1*) en interaction avec trois entités (*Entity1, Entity2, Entity3*) de son environnement. L'entité *Entity3* appartient u milieu intérieur du système (c-à-d gérer par ce dernier) alors que les autres entités constituent l'environnement externe du système. De plus, l'agent *Agent1* est capable de percevoir (de récupérer des informations) toutes les entités représenté dans la figure. En revanche, il ne peut agir que sur l'entité *Entity1* (flèche bidirectionnelle).

8.2.2.3 Diagramme de règles comportementales

L'objet de ce diagramme est de décrire le contenu du module de décision d'un agent coopératif, c'est-à-dire, les règles comportementales qui régissent les actions de cet agent. Ce diagramme permet de représenter des états spécifiques d'un agent. Un état est représenté par un rectangle aux angles arrondis contenant une expression définissant les conditions de son existence (voir figure 8.9). Ces conditions sont exprimées à l'aide d'opérateurs logiques (ou, et, non) et d'expressions de comparaison entre les connaissances de l'agent : *représentations,*

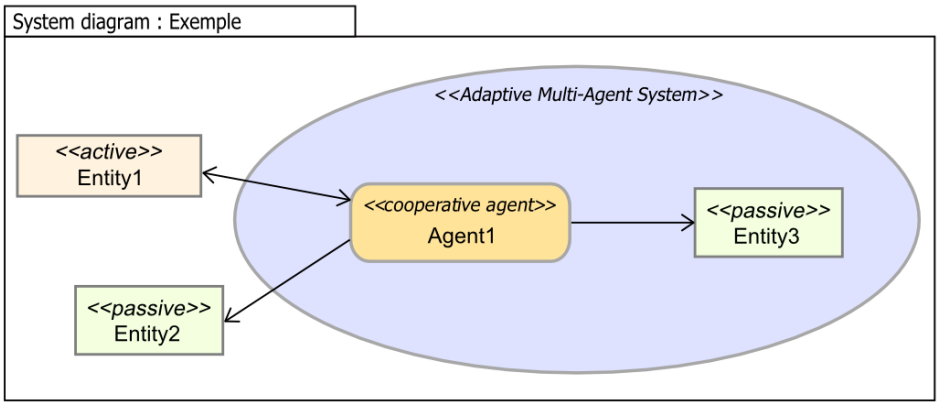

Figure 8.8 — Diagramme système AMAS-ML.

caractéristiques et compétences, ou des valeurs : booléennes, numériques, etc. Chaque règle décrit un comportement sous la forme d'un ensemble d'actions à mener lorsque l'agent se trouve dans un état particulier. Aussi, une règle lie-t-elle un état à des actions ou des compétences (pouvant elles-mêmes mettre en œuvre plusieurs actions élémentaires). L'expression d'une règle de coopération ne diffère pas de celle du comportement standard, à l'exception que l'état déclencheur correspond à une situation de non-coopération, et que la règle est graphiquement représentée différemment. La figure 8.9 constitue un exemple des capacités du diagramme de règles comportementales. Elle décrit deux règles distinctes, la première *cooperativeRule1* est une règle de coopération déclenchant la réalisation de l'action *action1* si l'état courant de l'agent vérifie les conditions suivantes :

- la valeur sa représentation *rep1* est inférieure à celle de sa représentation *rep2*,
- la valeur de retour de sa capacité *Skill1* est égale à vrai.

La règle *standardRule1* représente quant à elle, une partie du comportement nominal de l'agent. Elle consiste à exécuter *skill2()* et *action2()* dès lors que les conditions caractérisant son état déclencheur sont vérifiées.

8.2.3 Modélisation de l'environnement

Nous avons vu dans les premiers chapitres que les systèmes complexes informatisés possèdent des caractéristiques nécessitant une prise en compte de l'environnement et des dynamiques qui l'animent. Ce besoin est d'une part pris en compte par la théorie des AMAS et d'autre part, par la notion d'agent flexible (cf. chapitre 2). Néanmoins, cette prise en compte de l'environnement à l'exécution pourrait elle aussi bénéficiée d'une approche dirigée par les modèles. C'est pourquoi nous souhaitons étendre le langage AMAS-ML avec des concepts permettant de concevoir le couplage environnement/système. Nous avons initié dans ce sens, des travaux de méta-modélisation et proposé une première version d'éditeur, pour prendre en charge la modélisation de l'environnement du système.

Nous nous sommes particulièrement intéressés au domaine de l'informatique ambiante pour lequel la prise en compte de l'environnement est essentiel. L'environnement est consi-

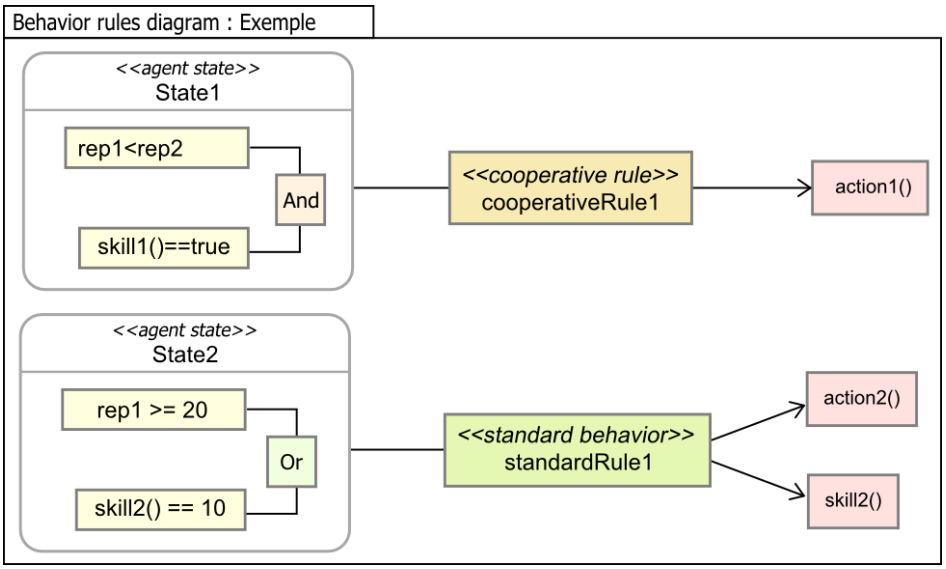

Figure 8.9 — Diagramme de règles comportementales AMAS-ML.

déré comme un ensemble de *ressources* et de *services* pouvant être manipuler par le système en développement aussi bien que par des entités externes à celui-ci. L'objectif de cette modélisation est de déterminer les frontières du système, c'est-à-dire quels sont les services et ressources pris en charge par le système, quels sont ceux qu'il requiert ? Dans le cadre d'une étude préliminaire de l'environnement, nous proposons quatre diagrammes correspondant à des niveaux d'abstraction différents :

1. **Diagramme d'états de l'environnement** : l'utilité de ce diagramme est de décrire de manière abstraite un scénario pouvant se dérouler dans l'environnement et face auquel le système doit réagir. Il est composé d'une succession d'états liés par des événements. Un événement étiquette une transition d'un état à un autre. L'environnement possède un état *« stable »* du point de vue du système ou initial. Le système doit agir et s'adapter aux variations de l'environnement pour rétablir cet état qu'il considère comme « stable ».
2. **Diagramme d'interaction environnement/système** : ce diagramme décrit les interactions susceptibles de se produire entre le système et son environnement. Ces derniers sont initialement considérés comme possédant un état stable. Le système doit être capable de revenir à cet état stable qui subira les perturbations provenant des variations d'états de son environnement. Il doit percevoir tout changement d'état qui influera sur son propre fonctionnement. Cette perception va se traduire par un événement qui, à son tour, changera l'état du système. Le système agit sur l'environnement par le biais d'actions dont l'objet est de l'état courant de l'environnement directement ou indirectement.
3. **Diagramme de composition de l'environnement** : ce diagramme décrit l'environnement en termes de ressources et de services ainsi que les relations liant ces éléments. Un service peut être fournit par le système ou par une entité externe et peut être requis par le système. Un service peut permettre d'agir sur une ressource de différentes manières, une ressource doit, quant à elle, être en relation avec au moins un service.

4. **Diagramme d'entités** : cette vue sur le modèle distingue les entités passives et actives qui vont participer aux interactions entre le système et l'environnement. Pour qu'un service soit en mesure d'agir sur une ressource, l'entité active mettant en œuvre ce service doit faire appel à l'entité passive gérant l'accès à la ressource. Ces entités correspondent à une représentation du contexte, c'est-à-dire une représentation de l'environnement à un instant donné résultant de la perception du système en termes de ressources, de services et d'événements.

Ces diagrammes sont organisés selon un processus permettant d'aborder la conception du système à partir de son environnement selon deux axes (cf. figure 8.10) :

– *Orientée système* : Dans ce cas, la conception d'un modèle d'environnement est orientée par les fonctionnalités du système en devenir. Cette approche utilise une approche plus progressive en s'attachant à abstraire en premier lieu les états du système et de son environnement. Les concepteurs déduiront les éléments constituant l'environnement (ressources et services) en analysant les interactions.
– *Orientée domaine* : la conception du modèle d'environnement est guidée par le domaine pour lequel le système est conçu. Il s'agit des éléments imposés par le contexte de développement, par exemple dans le cas d'une application domotique : l'ensemble des capteurs, des appareils présents dans le bâtiment, etc.

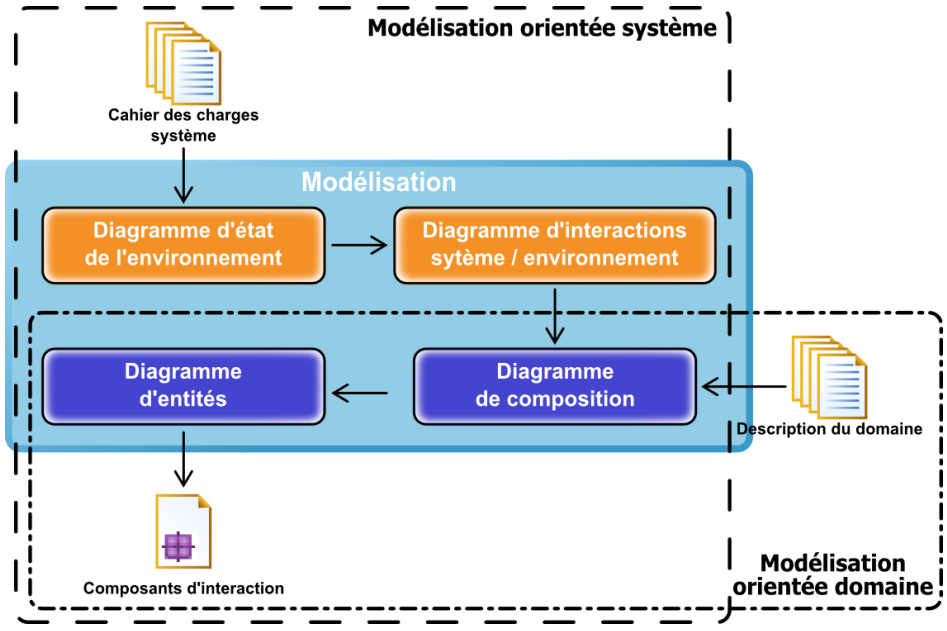

Figure 8.10 — Processus et axes de modélisation de l'environnement.

Cette démarche de modélisation de l'environnement n'a, pour l'instant, pas atteint la maturité que nous désirions. Les expérimentations présentées dans les chapitres suivants ne prennent pas cette approche en compte. Cependant, nous continuons à considérer l'étude de l'environnement comme une phase préliminaire particulièrement importante dans le contexte des systèmes informatiques complexes. Ainsi, nous souhaiterions profiter de cette

modélisation du couplage pour pouvoir extraire, à partir de ces modèles spécifiques, les interfaces des micro-composants d'interaction (capteurs et actionneurs).

8.3 μADL : un langage spécifique aux agents flexibles

μADL (micro-Architecture Description Language) est un langage de modélisation de micro-architecture basé sur les concepts architecturaux précédemment exposés d'agent flexible (cf. chapitre 6.2). Son méta-modèle permet de formaliser un certain nombres de propriétés structurelles, il garantit notamment le respect du style architectural, à savoir, une ensemble de micro-composants connectés à un même médiateur. Chaque modèle μADL correspond à l'abstraction d'une micro-architecture, c'est à dire à un nouveau type d'agent. Le langage μADL grâce au concept de micro-composant, autorise et facilite la réutilisation d'éléments de modélisation. Comme pour AMAS-ML ces principes ont été modélisés grâce au langage Ecore et sont présentés dans le paragraphe suivant.

8.3.1 Méta-modèle

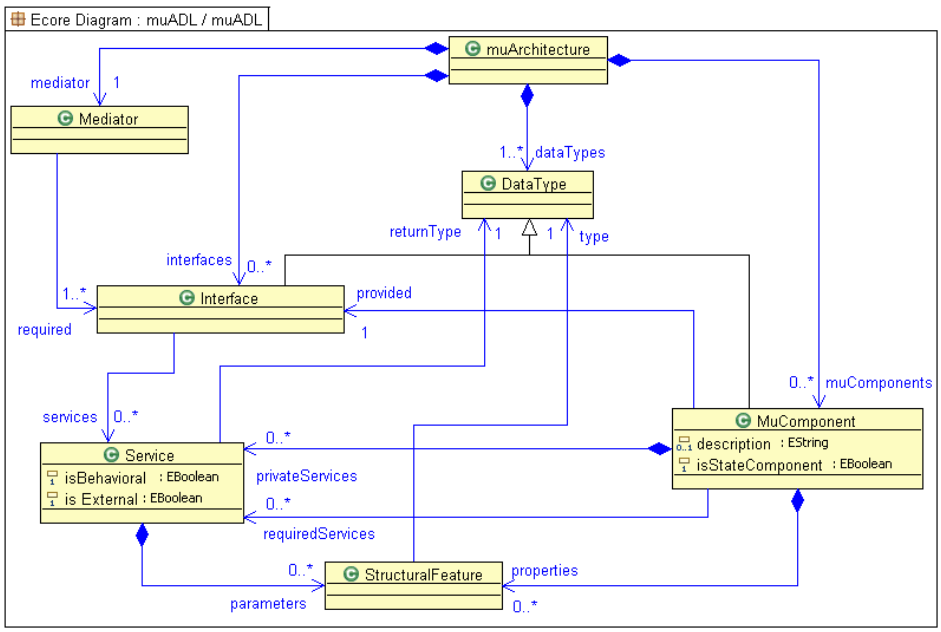

Figure 8.11 — Méta-modèle de μADL.

La notion d'agent flexible est basée sur un style architecturale « en étoile » organisé autour d'un médiateur qui implémente le principe de délégation (voir figure 8.11). Chaque micro-composant (*MuComponent*) se définit par :

- son nom (hérité de *NamedElement*),
- le fait qu'il possède un état rémanent (*withState*),

- le fait qu'il soit remplaçable à l'exécution (*isChangeable*),
- des propriétés structurelles (*properties*),
- un ensemble de traitements qu'il est capable de réaliser (*privateServices*),
- un ensemble de services qui lui sont nécessaire pour effectuer ses propres traitements (*requiredServices*).

Les services décrit au sein d'un micro-composant peuvent êtres fournis aux autres micro-composants ou au niveau comportemental par le biais d'une interface (*Interface*) elle-même liée au médiateur (*Mediator*). Chaque *Service* est caractérisé par :

- son nom,
- une courte description informelle (*Description*),
- sa visibilité permet de définir s'il est utilisable par le niveau comportemental ou uniquement au niveau opératoire (*isBehavioural*)
- son accessibilité représente le fait qu'il est accessible depuis l'extérieur (*isExternal*),
- un type de retour (*returnType*),
- des paramètres typés (*parameters*).

Les micro-composants peuvent être liés par le biais des services qu'ils requièrent de la micro-architecture et ceux qu'ils fournissent. Un exemple de ces relations entre micro-composants peut-être pris dans la micro-architecture des agents JavAct. Le composant de réception de message (*ReceiveCt*) nécessite le service de dépôt de message du composant boîte aux lettres (*MailBoxCt*). Ce dernier, illustre une autre propriété des micro-composants, l'état rémanent (*withState*). En effet, ces composants étant voués à être remplacés dynamiquement il est important de fournir au concepteur un marqueur, un moyen d'exprimer que l'état d'un micro-composant doit être gérer en cas de branchement ou de débranchement « à chaud ».

8.3.2 Syntaxe du langage

Chaque concept du méta-modèle que l'on a souhaité rendre éditable est ici associé à un élément graphique. Un exemple de modèle μADL est donné dans la figure 8.12. Un micro-composant est représenté sous la forme d'un rectangle contenant trois compartiments distincts : un pour son nom (*« μComponent »*), un pour ses caractéristiques structurelle (*propertiesCompartment*) et un dernier pour ses services (*servicesCompartment*). Chaque service est illustré par une icône dépendant de sa visibilité. S'il est utilisable au niveau comportemental ou décisionnel (*isBehavioural* positionné à vrai) une ampoule est affichée, sinon il est considéré comme opératoire et un engrenage est affiché. De même, l'accessibilité depuis l'extérieur de l'agent (*isExternal* positionné à vrai) est représenté par un arrobas. Une interface est représentée par un cercle bleu lorsqu'elle est connectée par un lien *provided* au micro-composant qui la fournit, rouge sinon. Toutes les interfaces sont liées au médiateur (*Mediator*). Ce dernier est toujours présent dans une micro-architecture, il est l'élément principal permettant la séparation des préoccupations ainsi que la délégation.

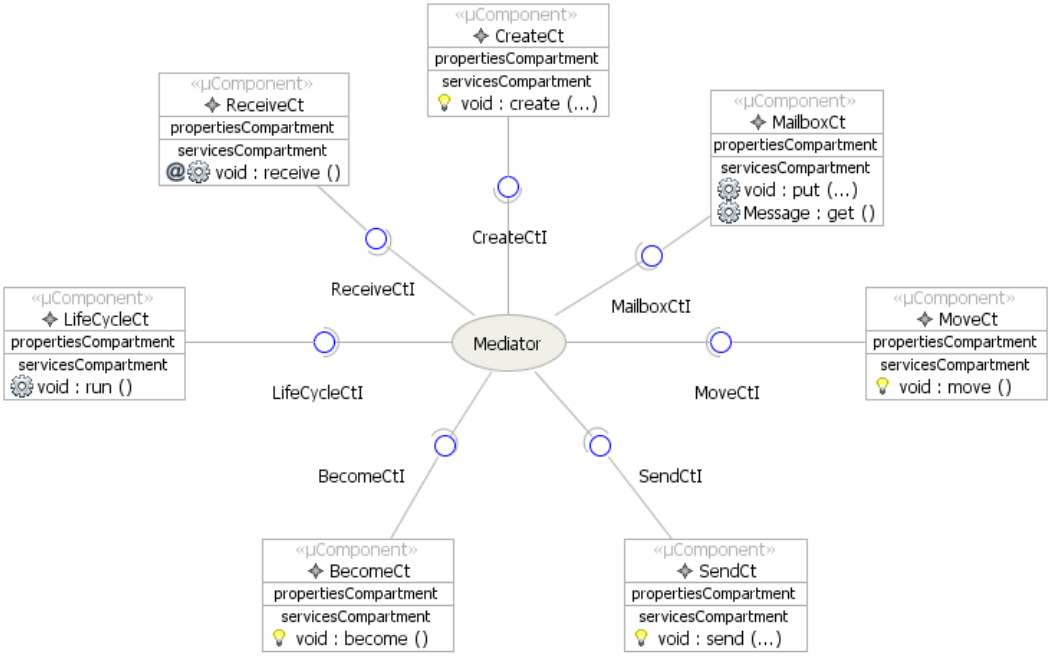

Figure 8.12 — Un exemple d'utilisation de μADL : la micro-architecture d'un agent JavAct.

8.3.3 Mise en œuvre de l'éditeur graphique

Le langage μADL est supporté par un éditeur graphique intégré à Eclipse (*plugin*). Ceci permet de bénéficier des capacités d'interopérabilité du format standard fournit par Ecore (XMI). Le développement de cet éditeur a également suivi un processus dirigé par les modèles. Les aspects graphiques (formes géométriques 2D, canevas, boutons, actions, etc.) sont gérés par le *plugin* GEF (Graphical Editing Framework) [1]. Ce dernier met en pratique le patron de conception MVC (Modèle Vue Contrôleur) afin de maintenir un couplage faible du modèle (s'appuyant sur Ecore et le *plugin* EMF) vis-à-vis de sa représentation graphique. Les deux *plugins* EMF et GEF sont mis en relation grâce à un troisième outil appelé GMF (Graphical Modeling Framework). Il permet de générer de manière plus ou moins automatique un *plugin* d'édition graphique se basant sur GEF et utilisant les résultats de la méta-modélisation en Ecore. L'approche GMF est intégralement dirigée par les modèles. Il s'agit de décrire tous les aspects de l'éditeur souhaité sous la forme de modèles.

Ainsi, le modèle du domaine (ou méta-modèle), décrit en Ecore, est associé au modèle d'outillage (décrivant la palette graphique et les actions de création, suppression possibles) et au modèle de représentation graphique (qui définit les différentes figures et leur propriétés de formes, tailles, etc.) par un quatrième modèle de mapping. Ce dernier, met en relation un concept du méta-modèle avec sa représentation graphique et les outils qui lui sont associés et permet la création d'un modèle de génération paramétrable. Ce modèle permet de personnaliser les paramètres de génération du code du *plugin*. Une fois ceux-ci complétés, GMF

1. `http://www.eclipse.org/gef/`

génère le code du *plugin* correspondant aux modèles définis (domaine, graphique, outillage et mapping). La figure 8.13 décrit le déroulement du processus de génération d'un éditeur grâce à GMF. Le *plugin* généré ne propose cependant que des fonctionnalités de base. GMF est encore en cours de développement et propose régulièrement de nouvelles capacités de génération. De ce fait, nous avons dû étendre cette première version du *plugin* pour offrir des fonctionnalités supplémentaires et améliorer la présentation des divers éléments de modélisation.

Ainsi, nous avons ajouté à l'éditeur μADL un outil de réutilisation de micro-composants, en autorisant la sauvegarde et le chargement individuels des micro-composants. De même, nous avons amélioré la spécification des paramètres des services des micro-composants ainsi que leurs représentations.

En termes de résultats, ce travail nous a permis de distinguer les limites d'une approche générative dans le cas des éditeurs de diagrammes. Elle permet de mettre en place rapidement (pour une personne rompue à la notion de modèle et à l'environnement Eclipse) un éditeur graphique fonctionnel, mais nécessite un travail d'extension dès lors que des fonctionnalités non-triviales sont requises. Quoi qu'il en soit nous avons obtenu un *plugin* répondant à nos attentes et permettant de décrire graphiquement des modèles de micro-architecture conformes à notre méta-modèle.

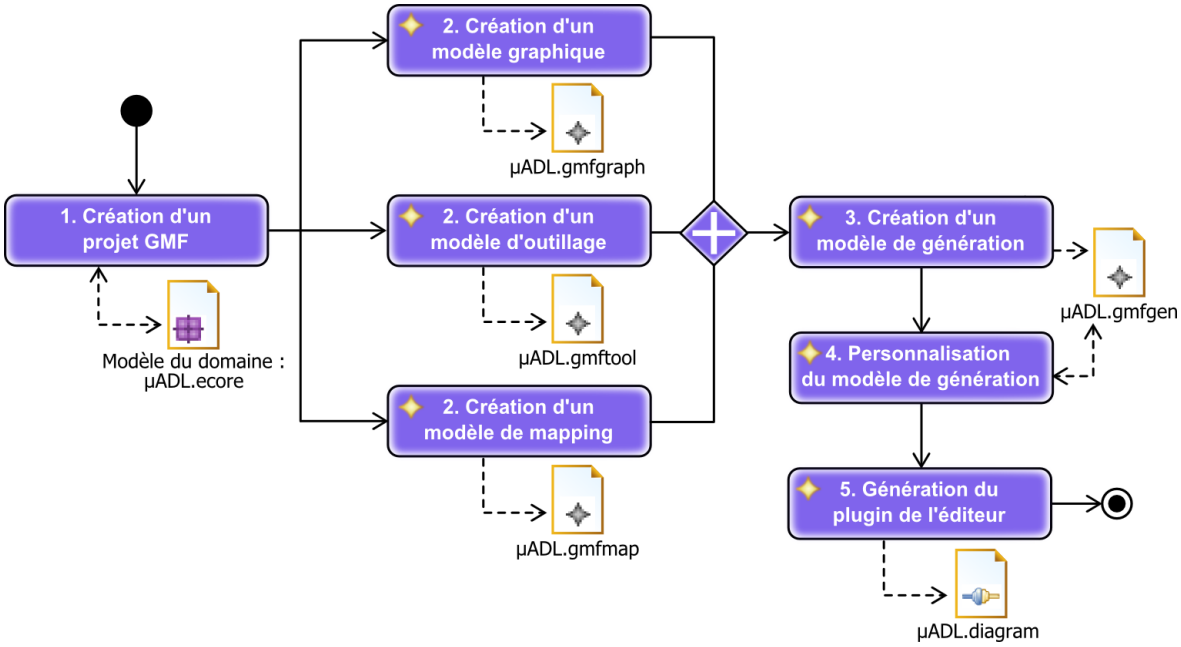

Figure 8.13 — Processus de génération d'un éditeur avec GMF (exemple de μADL).

ADELFE utilise l'UML pour décrire les aspects génériques du système en devenir. De plus, nous avons défini deux langages spécifiques afin de permettre la prise en compte de la théorie des AMAS et du modèle d'agent flexible dès la conception. Ces trois langages sont intégrés au sein d'un même espace technique, celui d'Eclipse et de son *plugin* EMF. En effet,

Eclipe fournit dans le cadre du projet MDT (Model Driven Tool) un *plugin* UML2 basé sur EMF et son langage de méta-modélisation Ecore [2]. Nous souhaitons lier ces trois langages par transformation, afin de permettre la description automatique d'une micro-architecture d'agent coopératif adaptée aux besoins de l'application. Cette micro-architecture, de surcroît, sert de base à la génération de la partie opératoire du code des agents.

Le chapitre suivant présente l'ensemble des traitements automatiques que nous souhaitons mettre en place afin d'intégrer notre vision de l'ingénierie des modèles à la méthodologie ADELFE.

2. `http://www.Eclipse.org/modeling/mdt/?project=uml2`

9 Transformations et ponts technologiques

L'INGÉNIERIE des modèles met particulièrement l'accent sur l'automatisation comme moyen de faciliter le développement de logiciel. Nous avons enrichi la phase de conception d'ADELFE par l'utilisation du langage spécifique AMAS-ML. De plus, les phases préliminaires du processus s'appuient désormais sur l'UML dans sa version 2.0. Le premier paragraphe de ce chapitre présente les transformations que nous proposons afin de lier ces deux langages.

Une phase d'implémentation a également été ajoutée à la méthode ADELFE. Elle s'inscrit dans une démarche dirigée par les modèles. De ce fait, elle utilise plusieurs phases de transformation et de génération permettant de produire le code final de l'application. La phase d'implémentation d'ADELFE a pour objectif de produire, à partir du modèle AMAS-ML, le code comportemental des agents (en se basant sur la théorie des AMAS) et le code opératoire (en respectant le modèle d'agent flexible). L'ensemble des transformations que nous avons implantées est présenté dans les paragraphes suivants.

ADELFE permet au concepteur d'utiliser une approche AMAS pour implanter son système. Cependant, ce choix est conditionné par l'adéquation du problème à la théorie. Ainsi, l'UML est utilisé dans les phases préliminaires de la méthode pour permettre, de déterminer cette adéquation sans imposer le modèle AMAS. Le modèle issu de ces premières phases peut tout à fait être utilisé pour concevoir et implanter le système dans une approche orientée objet classique. Si l'adéquation est avérée, nous souhaitons récupérer les informations pertinentes pour la conception du systèmes et de ses agents au sein de modèles spécifiques (AMAS-ML). Nous avons défini plusieurs stéréotypes UML afin de faciliter la tâche du concepteur en repérant dès les premières phases de conception les éléments pouvant être considérés comme des entités (actives ou passives) ou des agents coopératifs. En ce qui concerne l'étude des besoins et notamment l'élaboration des diagrammes de cas d'utilisation, nous proposons un stéréotype permettant d'identifier les interactions pouvant provoquer des situations non-souhaitées (Défaut de coopération). Ces stéréotypes permettent également d'identifier différents éléments du méta-modèle UML afin de les transformer de manière adéquate.

```
rule interaction2protocol {
   from i : UML2!Interaction (
      not(i.oclIsTypeOf(UML2!Class))
   )
   to p : AMAS!CooperativeInteractionProtocol(
      name <- i.name,
      definedMessages <- i.message->collect(m|thisModule.resolveTemp(m,'mA'))
   )
}

rule message2message {
   from m :UML2!Message
   using {
      sourceLifeline : UML2!Lifeline = m.sendEvent.covered->first();
      targetLifeline : UML2!Lifeline = m.receiveEvent.covered->first();
   }
   to mA : AMAS!Message (
      name <- m.name,
      protocol <- m.interaction,
      from <- thisModule.resolveTemp(sourceLifeline.represents.type,'agent'),
      to <- thisModule.resolveTemp(targetLifeline.represents.type,'agent')
   )
}
```

Figure 9.1 — Exemple de code ATL : transformation d'une interaction UML en un protocole AMAS-ML.

9.1 Transformation d'UML à AMAS-ML

Pour constituer la version préliminaire du modèle AMAS-ML à partir du modèle UML nous avons développé une transformation de modèle grâce au langage ATL. Cette récupération permet d'extraire des informations provenant des phases d'établissement des besoins et de l'étude des interactions. Nous présentons ci-dessous les principes de cette transformation :

1. **Entités** Il s'agit ici de parcourir le modèle UML issu des phases préliminaires d'ADELFE afin de récupérer les éléments ayant été identifiés (stéréotypés) comme des entités actives ou passives. Nous pouvons, pour chaque stéréotype trouvé, créer au sein du modèle AMAS-ML l'entité correspondante en récupérant le nom de l'élément du modèle sur lequel a été appliqué le stéréotype.
2. **Interactions** Nous avons choisi d'utiliser les diagrammes de séquence UML2.0 afin d'exprimer les interactions agents et agents/entités. Les messages échangés par les agents font partie intégrante de la conception détaillée. La notion de protocole d'interaction apparaît comme un élément indispensable au système dans le langage AMAS-ML. Notre objectif est d'utiliser UML 2.0 comme un support graphique afin d'éditer ces pro-

tocoles et AMAS-ML pour organiser les différents messages au sein des modules des agents coopératifs constituant le système. Le lien entre ces deux langages est assuré par une transformation de modèle. La figure 9.1 présente deux règles de transformation :

– *Interaction2protocol* : permet de transformer une *Interaction* UML en un protocole de coopération AMAS-ML (*CooperativeInteractionProtocol*). Il s'agit de récupérer le nom donné à ce protocole dans le modèle UML ainsi que l'ensemble des messages qui y sont définis.

– *message2message* : cette règle transforme un message UML (quelque soit ça nature *Assynchronous Call*, *Synchronous Call*, *Assynchronous Signal*, etc.) en un message AMAS-ML. Elle permet de récupérer la source et la cible du message (*from*, *to*) en retrouvant la classe représentée par les lignes de vie (*Lifeline*) entre lesquelles le message a été défini.

Cette transformation est également utilisée lors de la phase de conception d'ADELFE 2.0 pour définir les protocoles d'interactions entre agents coopératifs.

9.2 Chaîne de transformations de la phase d'implémentation

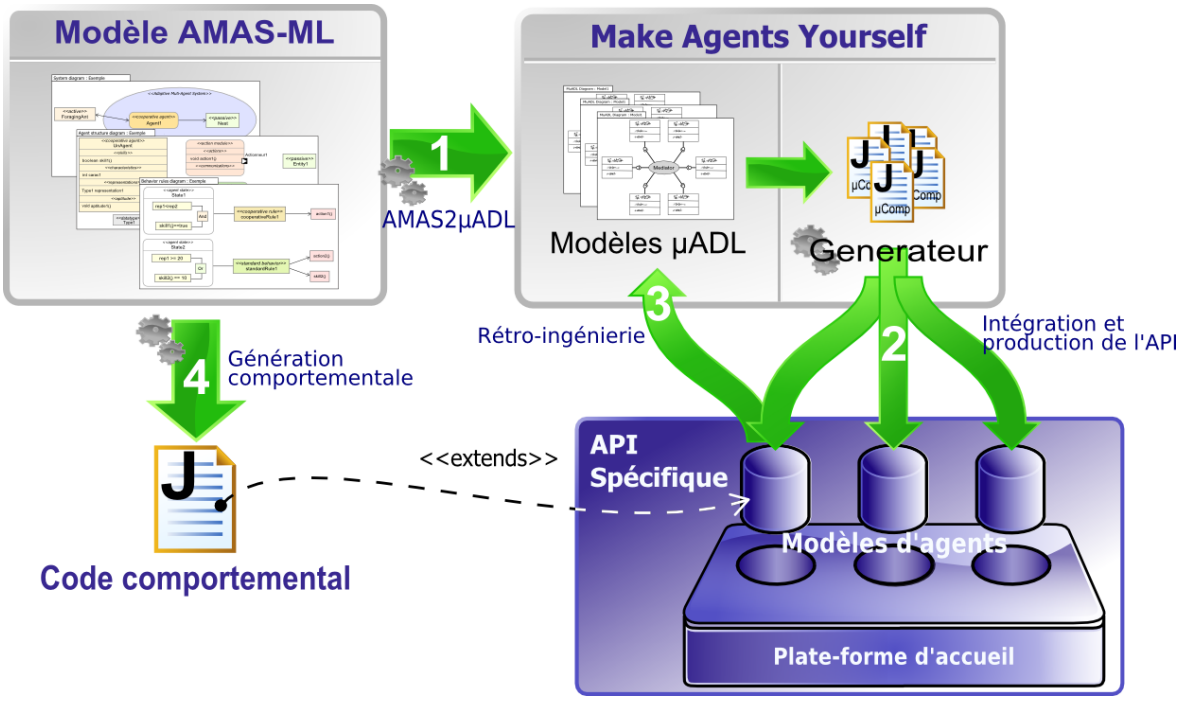

Figure 9.2 — Présentation générale des transformations.

Ce paragraphe présente brièvement la suite de transformation et de génération qui est utilisée dans la phase d'implémentation d'ADELFE. La production d'une API orientée agent dédiée est réalisée en plusieurs étapes de transformations et de génération de code

grâce, entre autre, à MAY (cf. paragraphe 9.4). Cette succession de transformations est associée à des tâches d'implémentation ou de réutilisation. En effet, la première étape de transformation (flèche numérotée 1 dans la figure 9.2) produit un modèle μADL pour chaque type d'agent coopératif identifié et décrit dans AMAS-ML (voir paragraphe 9.3). Ce modèle de micro-architecture peut-être enrichi ou modifié par réutilisation de micro-composants. Une fois la cohérence de ce modèle établie, il s'en suit une première étape de génération de code, celle de la micro-architecture abstraite de l'agent (génération interne à l'outil MAY, cf. figure 9.2). A ce niveau, il est possible de réutiliser du code déjà produit, en sélectionnant des implémentations existantes. Pour les nouveaux micro-composants un squelette de code est généré, il ne reste plus qu'à compléter le code des services. Leur intégration au sein de la micro-architecture, les liens de délégation entre micro-composants, les liens éventuels avec le niveau comportemental ou l'extérieur de l'agent ont été automatiquement générés au sein du *médiateur* et du *QuasiBehavior* (cf. paragraphe 9.4). Une fois cette tâche d'implantation réalisée, MAY produit une bibliothèque Java contenant l'API permettant la création, la manipulation et la programmation du comportement des agents (flèche numérotée 2 dans la figure 9.2). Cette API spécifique, répondant aux besoins établis lors des phases préliminaires d'ADELFE, est directement utilisable dans n'importe qu'elle application Java. Nous proposons, en outre, un outil de rétro-ingénierie permettant de récupérer un modèle μADL à partir du code d'une API (flèche numérotée 4 dans la figure 9.2).

La dernière étape de génération de code provient du modèle AMAS-ML et permet de générer des parties de code et des aides pour le développeur AMAS en charge du code comportemental des agents du système (flèche numérotée 4 dans la figure 9.2). Cette transformation utilise les règles décrites dans le modèle AMAS-ML et décrit un ensemble de conditions booléennes correspondant aux états identifiés de l'agent. Elles sont ensuite intégré à des expressions conditionnelles commentées et elle précise quelles sont les actions attendues lorsqu'un état particulier de l'agent est atteint.

Tous les détails de mise en œuvre des transformations présentées dans la figure 9.2 sont détaillés dans les prochains paragraphes.

9.3 Transformation AMAS-ML vers μADL

L'objectif de cette transformation est de permettre la génération d'une API (Application Program Interface) dédiée aux agents tels qu'ils sont exprimés dans un modèle AMAS-ML. L'idée est de traduire chaque agent coopératif du modèle AMAS-ML en un nouveau type d'agent respectant le style, c'est-à-dire une nouvelle micro-architecture. Il s'agit donc de définir une transformation entre le méta-modèle AMAS-ML et celui de μADL.

Cette transformation implique de déterminer la frontière entre ce qui doit être du ressort du comportement de l'agent et ce qui peut en être abstrait. Cependant, cette frontière entre opératoire et comportemental n'est pas nécessairement nette. Quoi qu'il en soit, nous considérons que, pour un agent coopératif, le niveau comportemental (ou niveau décision-

nel, cf. section 6.2.4) est constitué de tout ce qui est lié à la sélection d'une action à mener (*DecisionModule*). De ce fait, tout ce qui permet à un agent de percevoir, d'agir sur son environnement (*ActionModule* et *PerceptionModule*), de mémoriser, maintenir ses connaissances (*Knowledge*) et de mettre en œuvre ses capacités essentiels (*Aptitude*) est considéré comme mécanisme opératoire. La figure 9.3 résume ce choix d'implantation et représente la séparation des préoccupations telle que nous l'envisageons pour la micro-architecture d'un agent coopératif. Il s'agit là d'une simplification de l'architecture de l'agent coopératif, cependant il est important de noter que le niveau de définition d'un élément (opératoire ou comportemental) ne correspond pas nécessairement au niveau auquel il est visible. Par exemple, la gestion de connaissances et des perceptions est décrite au niveau opératoire, mais les connaissances sont utilisées par le module de décision du niveau comportemental et doivent donc y être accessibles. Les éléments du niveau opératoire seront transformés en micro-composants.

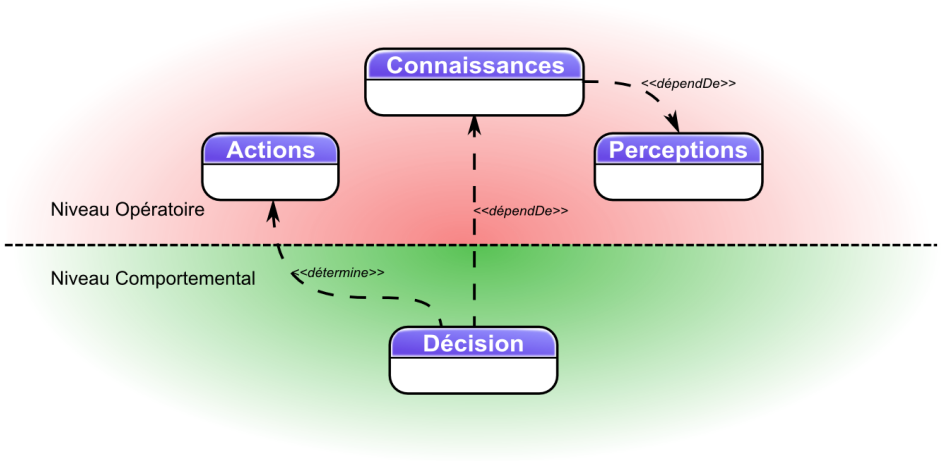

Figure 9.3 — Séparation des préoccupations au sein de la micro-architecture d'un agent coopératif.

Cette traduction est définie sous la forme d'une transformation ATL (voir l'exemple de la figure 9.4). Cette transformation : *AMAS2MuADL*, est composée d'un ensemble de douze règles (*MatchedRules*) qui permettent de transformer dans un modèle AMAS-ML tous les éléments instances d'une méta-classe donnée. Par exemple, dans la figure 9.4, la règle *Actuators2muComponent* est exécutée par le moteur de transformations ATL pour chaque instance de la méta-classe *Actuator* définit dans le modèle source (*AMAS*). Cette règle crée un *muComponent* homonyme dans le modèle cible (*muADL*) et définit ses caractéristiques en fonction du modèle AMAS-ML (propriétés et services). Chaque type d'agent défini dans le modèle AMAS-ML donnera lieu à un modèle de micro-architecture μADL. Pour chaque micro-architecture, le *Mediator* est automatiquement généré, puisqu'indissociable de notre style architectural (cf. paragraphe 6.2.4). En effet, le *Mediator* est connecté à toutes les interfaces fournies par les composants de l'architecture. Les services fournis par le biais de ces interfaces sont ainsi regroupés par le *Mediator* celui-ci rend accessible les services définis comme comportemental (*isBehavioural* à vrai) depuis le code fonctionnel.

```
rule Actuators2muComponent{
  from actuator : AMAS!Actuator
  to actuatorCt:muADL!muComponent(
    name <- actuator.name,
    description<- 'MuComponent dedicated to the performing of action over
                   environmental Entities',
    provided <- thisModule.resolveTemp
                (actuator,'providedActuatorInterface'),
    isChangeable <- true,
    withState <- false,
    properties <- actuator.properties->
                  collect(prop|thisModule.resolveTemp(prop,'struct')),
    privateServices <- actuator.operations->
                  collect(op|thisModule.resolveTemp(op,'service'))
  ),
  providedActuatorInterface:muADL!Interface(
    name <- actuator.name+'I',
    services <- actuator.operations->
                  collect(op|thisModule.resolveTemp(op,'service'))
  )
}
```

Figure 9.4 — Exemple de code ATL : transformation d'un actuateur AMAS-ML en un micro-composant μADL.

Les modules d'interaction, *ActionModule* et *PerceptionModule* de l'agent AMAS sont transformés en différents micro-composants. Chaque *Actuator* et *Sensor* est transformé en micro-composant. Chaque *Action* et *Perception* est transformée en un service et intégrée au micro-composant correspondant à l'actionneur, respectivement au capteur, qui le met en œuvre. Toutes les actions sont définies comme des services accessibles au niveau comportemental. Quant aux perceptions elles restent uniquement visible au niveau opératoire.
Les actions de communication sont transformées en service de niveau comportemental et ajouté à un micro-composant nommé *Send*. Les perceptions sont elles aussi transformées en service, toutefois, elles sont considérées comme des services visible au niveau opératoire et externes (accessibles par d'autres agents). Ces services sont ensuite ajoutés à un micro-composant appelé *Receive*.
Chaque micro-composant créé est associé à une interface qui regroupe l'ensemble des services qu'il fournit à la micro-architecture. Cette interface est automatiquement connecté au médiateur. Les propriétés typées du modèle AMAS sont transformées en caractéristiques structurelles du côté μADL et ajoutées au micro-composant correspondant ; les opérations sont traduites en termes de service dont la visibilité dépend de l'élément de l'agent coopératif auquel elles appartiennent, par exemple pour une perception, le service correspondant sera visible au niveau opératoire uniquement.

En résumé, chaque agent coopératif AMAS-ML est traduit en une micro-architecture, contenant un micro-composant pour :

- chacun de ses capteurs,
- chacun de ses actionneurs,
- ses représentations,
- ses caractéristiques,
- ses capacités,
- ses aptitudes,
- son cycle de vie (perception, decision, action),
- les messages qu'il peut recevoir,
- les messages qu'il peut émettre.

De plus, un certain nombre de liens de dépendances doivent être définis au sein du modèle μADL. Par exemple, les micro-composants de perception (les capteurs) requièrent les services du micro-composant de gestion des représentations afin de les mettre à jour. Toutes ces informations architecturales permettent de générer une première ébauche du code de l'architecture ; ce point fait l'objet du paragraphe suivant.

9.4 Un générateur d'API : MAY

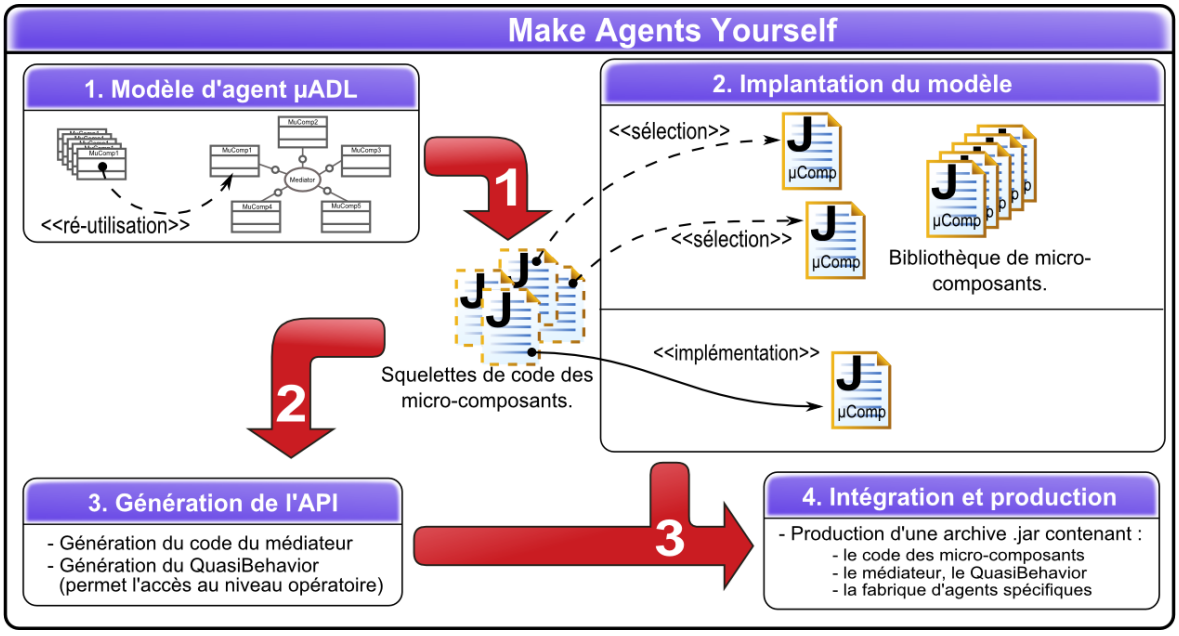

Figure 9.5 — Différentes étapes de génération de l'outil MAY.

Afin de satisfaire le besoin de personnalisation de la couche intergicielle (mise en adéquation du support de développement avec le besoin applicatif), nous proposons l'outil MAY (Make Agents Yourself) qui permet de construire les types d'agents adaptés au SMA qui doit

être développé et de générer une API dédiée à leur programmation. Nous montrons dans cette section comment, à partir d'un modèle de l'application on peut dériver les types d'agents et leur implantation au sein de l'intergiciel dédié. Ces agents, configurables statiquement et auto-reconfigurables dynamiquement, sont construits conformément au style d'architecture des agents flexibles que nous avons présenté dans le paragraphe 6.2. Nous précisons cette notion de style dans le paragraphe suivant. Nous montrons également comment ces principes architecturaux sont pris en compte au sein d'un langage de modélisation dédié et nous présentons l'outil de génération de code.
La figure 9.5 présente les différentes étapes du processus de génération d'une API orientée agent spécifique à l'aide de MAY. Dans cette figure chaque flèche constitue une phase de transformation ou de génération de code. L'idée est de construire une API utilisant des agents spécifiques décrit à partir d'un modèle abstrait respectant le style architectural (μADL).
MAY permet de générer l'API dédiée à la programmation des agents (programmation du niveau comportemental) pour les types dérivés du modèle de l'application conformément aux concepts architecturaux décrits précédemment. JavAct constitue, en quelque sorte, un exemple de ce que peut être un produit de l'outil MAY.

9.4.1 Modèle d'agent μADL

Cette première étape permet de modéliser les caractéristiques opératoires des agents qui seront la base de l'API générée par MAY. Elle utilise le langage μADL et produit des modèles respectant le style architectural des agents flexibles. A ce niveau il est possible de récupérer des spécifications de micro-composants pré-définies pour les intégrer au modèle et bénéficier de leur implémentation pour la suite du processus de génération. Dans le cas de la phase d'implémentation ADELFE 2.0 cette étape est automatiquement accomplie par transformation de modèle (cf. paragraphe 9.3). Toutefois, le résultat de la transformation peut tout à fait être modifiée, affinée pour réutiliser des micro-composants par exemple.

9.4.2 Génération de l'architecture abstraite

Cette étape de génération et représentée dans la figure 9.5 par la flèche numérotée par un un. A partir du modèle de l'architecture, une première version abstraite de ce type d'agent est générée. Pour chaque micro-composant une classe Java est générée. Celle-ci déclare une méthode pour chaque service défini dans le modèle ainsi qu'un attribut par propriété. Les service sont également annotés afin de rendre compte de leur niveau de visibilité ainsi que du fait qu'ils sont visibles de l'extérieur ou non. Ces annotations sont utilisées dans la phase de génération suivante.

9.4.3 Implantation du modèle

Les micro-composants abstraits issus de l'étape de génération précédente doivent ensuite être implantés. Un outil permet de sélectionner parmi les micro-composants pré-existants ceux qui peuvent correspondre à une spécification contenue dans le modèle. Pour les nouveaux composants il est nécessaire d'implémenter le corps de chacune des méthodes qu'ils déclarent. Une fois cette implémentation réalisée leur code est ajouté à la bibliothèque, ils deviennent ainsi ré-utilisables pour une autre architecture d'agent.

9.4.4 Génération de l'API

Cette phase est symbolisée par la flèche numéro deux dans la figure 9.5. Elle génère le *Mediator* et le *QuasiBehavior* qui garantissent l'accès aux services de niveau de comportemental et opératoire. Le *Mediator* regroupe les différentes méthodes issues de la traduction des services μADL tandis que le *QuasiBehavior* déclare l'ensemble des méthodes correspondant à des services de niveau comportemental. Le code du comportement de l'agent (module de décision en ce qui concerne les agents coopératifs) doit étendre cette classe abstraite *QuasiBehavior* pour accéder, par le biais du mécanisme d'héritage, aux services décrits dans les micro-composants comme comportementaux. Cette phase de génération utilise les annotations ajoutée aux classes des micro-composants pour répartir les méthodes au sein du *Mediator* et du *QuasiBehavior*. En plus de ces deux classe MAY génère une fabrique d'agent spécifique au modèle développé et une classe jouant le rôle de proxy (adresse visible de l'extérieur et retournée par la fabrique) qui gère l'accès aux services externes de l'agent.

9.4.5 Intégration et production

A l'issue des différentes transformations, MAY intègre l'ensemble des classes générées (*Mediator*, *QuasiBehavior* et)et implantées dans une archive Java. Elle contient les types d'agents modélisés par μADL et leurs mécanismes opératoires. Grâce à ces étapes de transformations et de génération, nous pouvons mettre en application des agents adaptés aux besoins spécifiques du projet dans une API dédiée. Depuis le code fonctionnel, on peut créer de nouvelles instances des types d'agents générés par MAY et utiliser les services de base des micro-composants de l'architecture. Un de nos objectifs est de générer depuis le modèle AMAS-ML une partie de ce code fonctionnel afin de faciliter encore la tâche du développeur.

9.4.6 Rétro-ingénierie

L'un des objectif de l'IDM est de maintenir au maximum la cohérence entre le modèle et le code généré. Les modifications éventuelles apportées au code lors de l'intégration des agents dans leurs environnement peuvent faire diverger le code de sa spécification. Dans ce but nous avons décrit une transformation permettant de redéfinir un modèle à partir des

classes Java produites par MAY. Ces dernières sont regroupées par paquetage, chaque paquetage correspondant à un modèle d'agent contenant une classe par micro-composants. Nous avons utiliser les capacités de la plate-forme Eclipse et de ses plugins JDT (Java Development Tool) et EMF (Eclipse Modeling Framework) pour développer un outil de rétro-ingénierie. Le principe de cet outil consiste en un parcourt de l'ensemble des unités de compilation d'un paquetage JAVA (fichiers sources) permettant de détecter des implantations de micro-composants (classe étendant *MuComponent*) afin de reconstruire un modèle μADL. La figure 9.6 présente l'algorithme de traduction d'un paquetage en une micro-architecture. Une fois le modèle créé celui-ci peut-être utilisé pour y introduire de nouveaux composants, réutiliser des composants prédéfinis et générer une nouvelle API intégrant les mises à jour décrites dans le modèle.

9.5 Génération comportementale

Cette étape de transformation permet de générer une ébauche de code et un ensemble de guide pour mener à bien le processus de décision. Ce code est voué à utiliser le résultat de la génération précédente. Il s'agit donc d'une portion de code Java décrivant le comportement des agents en utilisant l'API générée. Le comportement défini pour chaque agent doit étendre la classe abstraite *QuasiBehavior* afin d'autoriser l'accès au niveau opératoire. A partir du modèle AMAS-ML et de son module de décision décrit par le diagramme de règles comportementales, il est possible d'identifier les états de l'agent devant être pris en compte. Ces états peuvent être traduits en conditions booléennes dont les valeurs dépendront des différentes connaissances impliquées dans leur description et mises à jour automatiquement par la phase précédente de perception. Le processus de décision (méthode *decide()*) consiste ensuite, à déterminer en fonction des valeurs des états quelles sont les actions à mener. Cette génération a été définie sous la forme de deux requêtes ATL (*queries*), une permettant de générer des classes et des énumérations pour les différents type de données définis dans le modèle AMAS-ML et l'autre dédié à la génération de des règles comportementales. La figure 9.7 présente la fonction ATL (*helper*) permettant de générer le bloc conditionnel Java correspondant à une règle. Cette fonction est appelée pour toutes les règles contenues dans le module de décision de chaque agent.

9.6 Conclusion

Dans ce chapitre nous avons présenté l'ensemble des transformations de modèles que nous avons développé pour assurer la génération d'une application respectant les principes des AMAS. Le code produit par cette chaîne de transformation s'appuie sur une API spécifique basée sur les principes architecturaux des agents flexibles. Ce processus est inscrit dans une démarche dirigée par les modèles et met en œuvre deux langages spécifiques AMAS-

```
Créer un modèle muADL;
Créer une micro-architecture;
Initialiser la micro-architecture (Ajouter un médiateur et les types de données JAVA);
Pour chaque élément contenu par le paquetage
   Si l'élément courant est une unité de compilation alors
      Pour chaque classe définie dans l'unité de compilation
         Si la classe courante est un micro-composant alors
            Créer un micro-composant homonyme de la classe courante;
            Créer une interface;
            Ajouter l'interface à la micro-architecture;
            Ajouter l'interface comme interface fournie par le micro-composant;
            Pour chaque attribut définit par la classe courante
               Créer une propriété homonyme de l'attribut courant;
               Rechercher ou créer le type de l'attribut dans le modèle;
               Ajouter la propriété au micro-composant courant;
            fin pour
            Pour chaque méthode définit par la classe courante
               Créer un service homonyme de la méthode courante;
               Rechercher ou créer le type de l'attribut dans le modèle;
               Ajouter les paramètres de la méthode au service courant;
               Ajouter le service au micro-composant courant;
               Si la méthode courante est publique alors
                  Ajouter le service à l'interface fournie par le micro-composant;
               fin si
            fin pour
         Sinon si la classe courante est une interface de micro-composant alors
            Rechercher une interface dans le modèle;
            Si l'interface n'est pas présente dans le modèle alors
               Créer une nouvelle interface;
               Récupérer les services de l'interface (idem micro-composant);
            fin si
            Connecter l'interface au médiateur;
         Sinon
            Créer un nouveau type de données homonyme la classe courante;
         fin si
         Ajouter l'élément créer à la micro-architecture;
      fin pour
   fin si
fin pour
Ajouter la micro-architecture au le modèle;
```

Figure 9.6 — Algorithme de traduction d'un paquetage JAVA en un modèle μADL.

ML et μADL qui ont été liés par le biais d'une transformation (cf. paragraphe 9.3). Dans un deuxième temps les résultats de cette transformation (des modèles μADL) sont utilisés par l'outil de génération MAY (cf. paragraphe 9.4) pour produire une API spécifique. Le processus ADELFE grâce à ces transformations est ainsi entièrement dirigée par les modèles, des phases préliminaires jusqu'à l'implantation.

```
-- Transforming AMAS Actuator into homonymic muADL MuComponents.
helper
context AMAS!Rule
def : generateIfThenElse(): String =
'\ t**\ n\ t* Generated '+
  if self.oclIsTypeOf(AMAS!CooperativeRule) then
  'cooperative rule : '
    +self.name+' handles '+
    self.handledNCSName()+' situation :\ n\ t'+
    self.description
  else
    'standard rule : '+self.name
  endif
  +'\ n\ t*\ n'+
  '\ tif ('+ self.trigger.condition.generateCondition()+
  ')\ n'+
  self.impliedActions
   ->iterate(a; accA: String=''|accA+'\ t\ t'+
  a.generateAction()+'\ n\ t\ t'
);
```

Figure 9.7 — Génération d'une conditionnelle Java à partir d'une règle comportementale AMAS-ML.

Cependant, nous devons valider l'utilisation de nos langages et des transformations associées sur des exemples concrets. La partie suivante expose une mise en pratique de la version 2.0 d'ADELFE, c'est à dire l'utilisation des langages spécifiques AMAS-ML et μADL ainsi que des différentes transformations de modèles pour la réalisation de deux applications.

Quatrième partie

Applications

Introduction

Cette partie présente deux expérimentations que nous avons menées pour valider notre proposition. Elles ont permis de tester le nouveau processus de développement ADELFE 2.0, les langages et les transformations que nous avons décrits dans les chapitres précédents.

La première application développée propose un environnement de simulation de fourmis fourrageuses. Cette application avait été développée précédemment dans notre équipe mais sans utiliser d'approche dirigée par les modèles. Nous l'avons choisie comme exemple car les comportements des agents étaient bien définis et connus. Cela élimine de fait la difficulté de la conception et nous permet de nous focaliser sur l'efficacité à la fois du langage de modélisation AMAS-ML et des transformations mises en place dans la phase d'implémentation d'ADELFE. ADELFE a permis d'implanter une nouvelle version des agents coopératifs respectant le style architectural d'agent flexible à partir d'un modèle μADL. Ces agents ont été intégrés dans un environnement de simulation entièrement réimplanté en Java.

La deuxième application présentée est un exemple de système de gestion de production. AMAS-ML et ADELFE 2.0 ont été utilisés pour modéliser le système de contrôle manufacturier DAmasCop et l'intégrer à la plate-forme de simulation MASC [1]. Nous avons ainsi soumis notre approche aux tests de nouveaux concepteurs dans des conditions d'utilisation *« réelle »* pour un système n'ayant jamais été développé.

Enfin, nous tirons des conclusions et évaluons les apports de notre approche à la lumière des résultats obtenus dans ces deux exemples d'application.

1. http://www.irit.fr/COLLINE/

10 Ants : simulation de fourmis fourrageuses

LE but de cette expérience est de tester les transformations de modèles que nous venons de décrire et de valider notre démarche dirigée par les modèles dans son ensemble (langages et automatisations). Pour ce faire, nous avons choisi de prendre comme exemple une application bien maîtrisée. Ainsi, nous avons pu nous concentrer sur la phase d'implémentation pour tester les apports et les résultats fournis par les transformations de modèles.

Le système développé et présenté dans ce chapitre est un environnement de simulation spécifique permettant de visualiser le comportement d'une colonie de fourmis fourrageuses. Cet exemple a déjà été programmé en 1999 selon la théorie des AMAS mais sans bénéficier de méthode de conception (Topin et al., 1999). Nous souhaitons, avant tout, examiner l'apport de notre démarche de modélisation et de traitements automatiques. Cet exemple a été choisi, en outre, pour la relative simplicité du comportement de fourragement des fourmis. Nous présentons brièvement ci-dessous les spécifications du système en termes d'éléments à simuler, puis nous présentons le déroulement du processus d'ADELFE 2.0 tel qu'il a été présenté dans le chapitre 7.

10.1 Cahier des Charges

Il s'agit d'un problème qui fût posé d'une part, pour fournir un outil aux éthologues et d'autre part, pour établir l'apport de la coopération en comparant des fourmis suivant l'approche AMAS aux fourmis non coopératives. Le monde dans lequel évoluent nos laborieux arthropodes est composé d'un nid, d'un certains nombres d'obstacles, de phéromones et bien entendu de nourriture. La phéromone s'évapore au cours du temps. Elle peut être accumulée lorsque plusieurs fourmis en déposent au même endroit. Les fourmis possèdent plusieurs caractéristiques. Elles ont différents degrés de perception du monde qui les entoure (un champs de perception). Elles connaissent la position de leur nid. Elles sont capables de transporter une quantité donnée d'aliments. Elles restent hors du nid pour une durée déterminée, à la fin de cette durée elles y retournent.
Le comportement de fourragement de ces fourmis consiste tout d'abord à explorer leur monde. Lorsqu'elles y rencontrent des obstacles, elles les évitent. Si elles y découvrent de la nourriture, elles la récoltent et retournent au nid en déposant une certaine quantité de phéromone sur leur passage. En conséquence, les pistes de phéromones apparaissent et disparaissent dynamiquement dans l'environnement. Au cours de son exploration, une fourmi est particulièrement attirée par les phéromones qu'elle détecte, ainsi elle est conduite à suivre

la voie empruntée par ses congénères. S'il s'avère que cette piste conduit effectivement à de la nourriture, elle déposera à son tour de la phéromone. Ce comportement induit ainsi un renforcement des pistes existantes.

10.2 Conception à l'aide d'ADELFE 2.0

Ce paragraphe décrit la manière dont l'environnement de simulation présenté ci-dessus peut être mis en œuvre grâce à ADELFE et sa démarche dirigée par les modèles. Ainsi, nous nous concentrons principalement sur les dernières étapes de la méthodologie, c'est-à-dire celles qui ont été modifiées ou ajoutées pour permettre l'intégration des modèles spécifiques AMAS-ML et μADL ainsi que la génération de code. Les premières phases sont, quant à elles, résumées dans les deux prochains paragraphes.

10.2.1 Etudes des besoins

Ces étapes sont consacrées à l'établissement des exigences, elles sont habituelles dans les méthodologies de développement logiciel. Elles consistent en une description du domaine du problème tel qu'il nous est proposé de le résoudre et une spécification des besoins de l'utilisateur final. La première phase, à savoir la phase d'établissement des *exigences préliminaires* a dores et déjà été menée à bien. Bien qu'elle ne soit pas formalisée, la brève description du paragraphe 10.1 peut être considérée comme son résultat. Il constitue une vue d'ensemble des exigences (les besoins des utilisateurs, mots-clés et les limites).

Concernant la phase suivante, celles des *exigences finales*, elle est impliquée dans la description du système dans son environnement et permet l'identification des différents éléments qui le peuplent. A partir du cahier des charges établi précédemment, nous déterminons les entités suivantes :

- *Entités passives* (ressources pour le système) : nous pouvons identifier la phéromone, les obstacles, la nourriture et le nid.
- *Entités actives* (qui peuvent agir de manière autonome) : il n'y en a qu'une il s'agit de la fourmi.

De plus, en fonction de la description du problème, nous pouvons caractériser l'environnement comme étant :

- Accessible : son état est connu par le système, ce qui est bien le cas dans le cadre d'une simulation où l'on veut pouvoir observer les résultats de l'exécution ;
- Non-déterministe : les actions mené par les entités actives (les fourmis) peuvent produire des effets différents et imprévisibles ;
- Discret : il s'agit d'un environnement simulé, il est considéré comme une grille ;
- Dynamique : les actions des entités actives le modifient en permanence, de plus les phéromones s'évaporent avec le temps.

Si l'on considère le point de vue moins spécifique aux agents des cas d'utilisation, les fonctionnalités requises pour le système sont liées à la gestion d'un outil de simulation : la confi-

guration des paramètres de simulation, l'observation, etc. Nous ne les présenterons pas précisément ici car elles n'impactent directement pas la suite de la modélisation spécifique aux AMAS.

10.2.2 Analyse

Après avoir décrit les exigences, nous pouvons proposer une première analyse qui vise à nous permettre de déterminer si une approche AMAS est adaptée au problème ou non. Cette tâche d'évaluation de l'adéquation, permet d'étudier du niveau global au local l'activité du système et de ces entités. Dans l'exemple qui nous intéresse ici, l'activité du système, du point de vue global au local peut-être caractérisée de la manière suivante :

- il n'existe visiblement pas d'algorithme pour décrire la tâche que doit accomplir la colonie de fourmis,
- cette tâche globale (la récolte de nourriture) est menée à bien par un grand nombre d'entités en interaction (les fourmis),
- ces entités sont logiquement distribuées, il n'y a pas de contrôle centralisé,
- leur environnement évolue au cours du temps,
- chacune de ces entités possède une perception limitée de son environnement ainsi qu'un ensemble relativement restreint d'actions à mener pour gérer un contexte en perpétuel évolution.

A partir de cette caractérisation de l'activité du système il apparaît qu'une approche AMAS est particulièrement bien adaptée. En outre, la seule entité capable d'être considérée comme un agent coopératif, n'est autre que la fourmi car :

- il s'agit de la seule entité possédant une activité autonome et essayant d'atteindre un objectif qui lui est propre : celui de récolter de la nourriture,
- elle possède une vision partielle de son environnement qui de surcroît est en évolution constante,
- elle doit gérer des interactions avec d'autre entités (fourmis, nourriture, etc.) et peut donc potentiellement être confronté à des défauts de coopération.

Pour résumer, nous avons déterminé que l'approche AMAS est appropriée au problème que nous voulons résoudre. Grâce aux exigences, nous avons aussi pu identifier les différentes entités du système ainsi que les agents : les fourmis. A partir de cette étape, nous mettons l'accent sur la conception et la mise en œuvre de notre agent coopératif fourmi. Pour ce faire, nous avons adopté une approche axée sur le modèle plutôt que sur le code en nous appuyant sur la version 2.0 d'ADELFE.

10.2.3 Conception

Les premières phases de la méthodologie ADELFE sont basées sur le processus unifié qui est intrinsèquement lié à la notation UML. Les informations contenues par le modèle résultant des phases préliminaire sont récupérées par transformation. La section suivante

présente les modèles qui ont été définis pour concevoir l'outil de simulation des fourmis fourrageuses.

10.2.3.1 Modélisation de l'agent ForagingAnt

Pour la conception précise des agents, nous utilisons le diagramme agent d'AMAS-ML (voir la figure 10.1 pour plus de détails). Dans notre application, une fourmi n'utilise pas de communication directe, elle dépose de la phéromone qui peut être détectée par d'autres fourmis (Stigmergie), c'est pourquoi aucune action de communication n'est présentée dans la figure 10.1. La phase de perception consiste à renseigner les différentes *représentations* avec de nouvelles valeurs. Par exemple, le vecteur de positions *food* correspond aux endroits où de la nourriture a été perçue par la fourmi. Grâce à l'ensemble de ses *représentations* (*food*, *ants*, *obstacles*, etc.) et ses compétences (*favor()*, *freeFood()*, etc.) l'agent ForagingAnt doit déterminer sa prochaine action ou plus précisément sa prochaine position. Pour ce faire, il remplit la grille *interpretedSurroundings* avec des valeurs entières correspondant à son niveau d'intérêt pour chacune des positions sur laquelle elle est susceptible de se rendre. Le processus de décision consiste à déterminer ces valeurs en fonction des positions préférées (*favor()*). Cette décision est exprimée sous forme de règles et est décrite dans la section suivante.

Comme il a été présenté dans le paragraphe 10.1, les fourmis tracent des pistes de phéromone qui permettent de recruter leurs congénères afin de récolter la nourriture découverte. Cette tâche nécessite l'utilisation d'une action spécifique *dropPheromone()* qui est menée à bien par l'actionneur spécifique *ExocrynGland*. Cette information est utilisée lors de l'extraction de la micro-architecture de l'agent, elle indique quelle partie de l'agent est responsable de la réalisation d'une action (cf. paragraphe 8.3). Les autres spécificités de l'agent coopératif *ForagingAnt* sont représentées dans la figure 10.1, l'ensemble de ces caractéristiques permettent de réaliser la prochaine étape de conception.

10.2.3.2 Comportement de l'agent ForagingAnt

Nous distinguons deux types de règles, celles liées au *comportement standard* qui constitue la fonction nominale ou le moyen d'atteindre le but assigné à un agent et celles représentant le *comportement coopératif* qui sont destinées à gérer les *situations de non coopération*. À cette étape de la méthodologie, nous concevons ces règles comme étant déclenchées par un état de l'agent. les différents états de l'agent sont caractérisés par une expression logique permettant de mettre en relation les valeurs des *représentations, caractéristiques* et *compétences* de l'agent. Une règle peut avoir pour conséquence un ensemble d'actions et de compétences qui doivent être mise en œuvre pour atteindre un objectif ou pour recouvrer un état de coopération. La figure 10.2 montre un exemple de ces règles. La partie gauche de la figure représente des états particuliers qui provoque la réalisation des actions représentées sur la partie droite. Une règle comportementale lie un état à un ensemble d'actions et son nom est indiqué dans un rectangle (au milieu de la figure), qui est également utilisée pour spécifier le type de

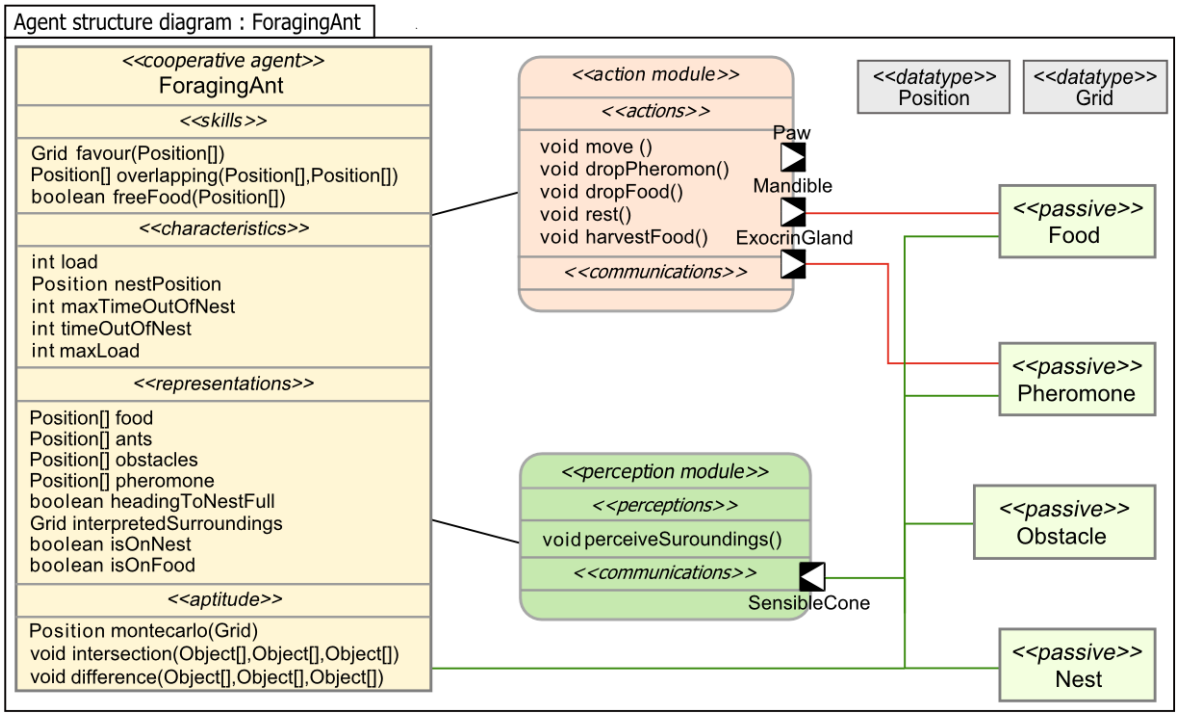

Figure 10.1 — Diagramme agent AMAS-ML : conception détaillée de l'agent coopératif fourmi.

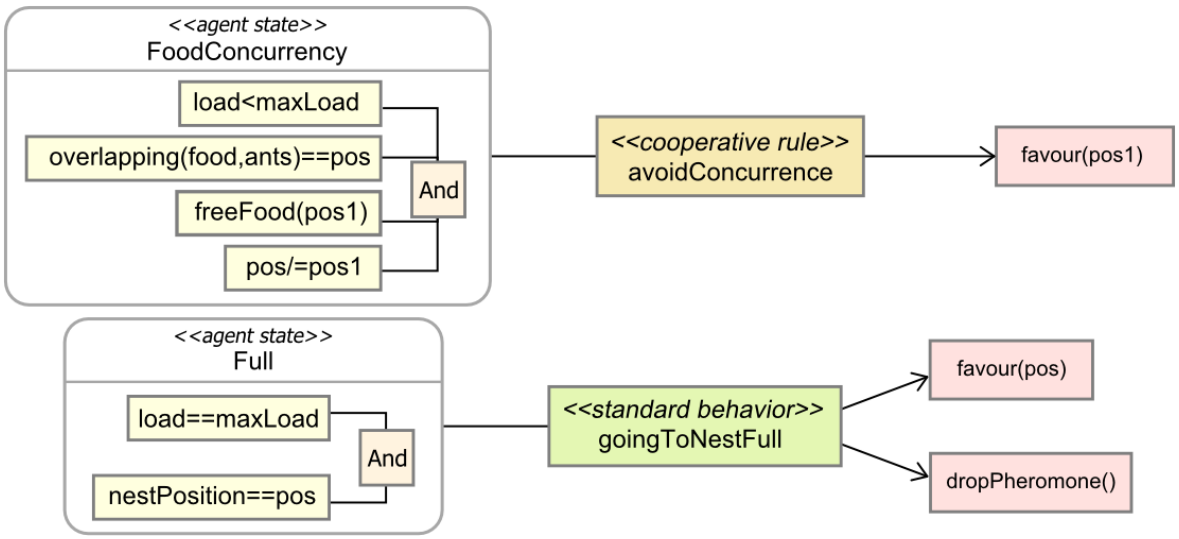

Figure 10.2 — Exemple du diagramme de règles comportementales AMAS-ML : règles de coopération et de comportement nominal.

comportement à laquelle cette règle est liée (coopératif ou standard). La figure 10.2 présente les deux types de règles. Pour la première, il s'agit de prévenir la concurrence qui peut apparaître lorsque des fourmis convoitent la même source de nourriture. Si une fourmi détecte de la nourriture et des fourmis sur des positions communes (*overlapping(food,ants)==pos*) mais qu'il existe de la nourriture qui n'est pas encore exploitée (*freeFood(pos1)*) alors elle choisira cette position. La seconde représente un comportement standard de la fourmi qui

consiste à retourner au nid en déposant une piste de phéromone lorsqu'elle a récolté de la nourriture.

10.2.4 Implantation

Cette phase à pour objectif de fournir l'implémentation de l'application telle qu'elle a été modélisée dans les phases précédentes. Notre objectif est de combiner l'approche AMAS-ML, prise en compte par des modèles spécifiques dans la phase de conception, au modèle d'agent flexible, dont le style architecturale est incarné par le langage de modélisation spécifique μADL. Cette phase utilise plusieurs étapes de transformation et de génération de code qui permettent d'obtenir le code de l'application. Ces différentes étapes ont été réalisées dans le cadre de cette application et sont présentées dans les prochains paragraphes.

10.2.4.1 Modèle μADL

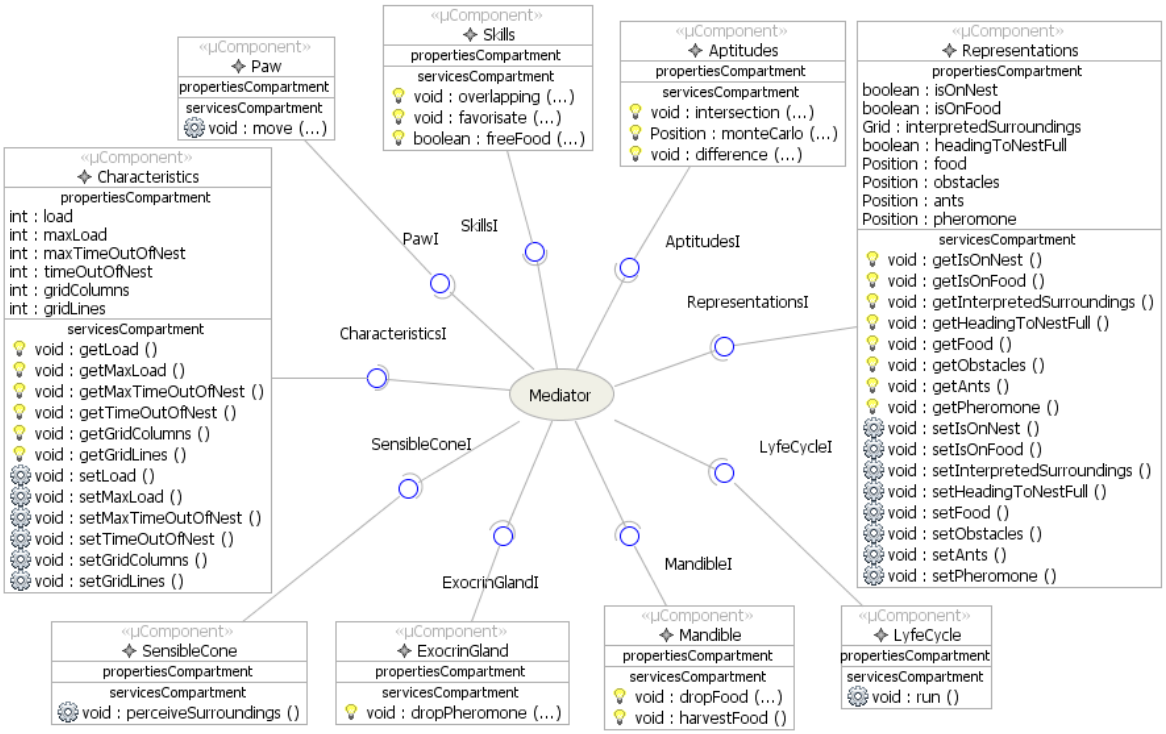

Figure 10.3 — Diagramme μADL de l'agent coopératif fourmi

A partir de la phase de conception et du diagramme agent d'AMAS-ML, nous de générons automatiquement une architecture abstraite d'agent. C'est-à-dire une spécification de l'architecture de l'agent constituée de micro-composants au sein d'un modèle μADL. La figure 10.3 représente l'agent coopératif fourmi résultat de la transformation modèle à modèle (cf. paragraphe 9.3) réalisée à partir du modèle AMAS-ML de la figure 10.1. Ce modèle peut-être complété ou modifié pour ré-utiliser des micro-composants prédéfinis par exemple.

Le modèle d'agent issu de cette étape de modélisation est par la suite utilisé pour générer une API spécifique aux agents fourmis coopératifs grâce au générateur de MAY (cf. paragraphe 9.4). Cette API est utilisée par le développeur afin de compléter les étapes suivantes de cette phase.

10.2.4.2 Code généré

MAY génère une API dédiée à partir d'un modèle d'agent μADL. Dans notre cas, ce modèle a été extrait à partir des résultats de la conception AMAS-ML (voir 10.2.3.1). L'architecture abstraite d'agent doit être implémenté soit par réutilisation du code de micro-composants, soit par l'implémentation de nouveaux. Pour ce qui concerne notre projet nous avons implémenté l'ensemble des micro-composants présents dans le diagramme de la figure 10.2.3.1. Un exemple d'une partie du code d'un micro-composant est donné dans la figure 10.4. Il s'agit du composant implémentant le capteur de l'agent fourmi lui permettant de percevoir les différentes entités qui l'entoure et de mettre à jour ses représentations en conséquence. Une fois cette tâche terminée, MAY peut générer l'API spécifique qui sera utilisée pour le développement de l'agent comportement.

```
...
/* Perception re-feeding */
ArrayList eltSeen = environment.elementsArroundPosition(pos, max_cl, max_ln);
for (Object o : eltSeen){
        DomainElement de =(DomainElement) o;
        Position posDe = de.getPosition();
        if(de.containsAnt())
                ants.add(posDe);
        if(de.containsFood())
                food.add(posDe);
        if(de.containsObstacle())
                obstacles.add(posDe);
        if(de.containsPheromone())
                pheromone.add(posDe);
        if(de.free())
                freeP.add(posDe);
}
myControler.setIsOnFood(food.contains(pos));
myControler.setIsOnNest(environment.getElementAt(pos).containsColony());
...
```

Figure 10.4 — Exemple de code d'un micro-composant : SensibleCone.

10.2.5 Code de l'application

A ce stade du processus, il reste au développeur AMAS à implémenter le comportement de l'agent fourmi coopératif. Le développeur AMAS peut mettre en pratique l'API qui a été

généré afin de décrire ce comportement, en utilisant par exemple le service de tirage aléatoire (*monteCarlo()*) fourni par le micro-composant d'aptitude. En outre, le modèle AMAS-ML contient une description du comportement de l'agent sous forme de règles (voir la section 10.2.3.2). Cette partie du modèle AMAS-ML peut servir de base à une autre étape de génération de code, qui permet de générer une partie du code comportemental ou de produire des guides pour la mise en œuvre du processus de décision. Concernant notre exemple, le module de décision de l'agent ForagingAnt doit lui permettre de déterminer sa prochaine position. La mise en pratique de cette décision se fait par la valuation d'une grille représentant les positions potentiellement éligibles (*interpretedSurroundings*). Cette grille est donnée à l'aptitude *montecarlo* qui choisit la position suivante, grâce à un algorithme aléatoire pondéré par les valeurs de la grille. Ainsi, la décision de la fourmi peut se résumer comme la valuation de cette grille ainsi que le positionnement de la représentation *headingToNestFull* qui correspond au dépôt de phéromones si nécessaire. Nous avons implanté au niveau comportemental deux méthodes *decide()* et *act()* qui sont appelées au sein du micro-composant *Lifecycle*. Nous considérons que la gestion des perceptions et des représentations est un mécanisme d'ordre opératoire ; en revanche, la prise de décision et sa réalisation en fonction des résultats de la perception, doivent être gérées au niveau comportemental de l'agent.

10.3 Conclusion

Nous avons développé un environnement de simulation sous la forme d'une grille contenant différents éléments : un nid, des fourmis, des obstacles, de la nourriture et de la phéromone. Chacun de ces éléments est représenté au sein d'une interface graphique qui permet, entre autres, d'évaluer visuellement les résultats de la simulation. Cette interface est également un moyen pour l'utilisateur de définir de nouveaux environnements de simulation, de les sauvegarder et de les recharger. Un ensemble de menus et d'éléments d'interfaces permettent de retirer ou d'ajouter dynamiquement des agents fourmi, des obstacles et des plots de nourriture en cours de simulation dans l'environnement. La figure 10.5 correspond à l'exécution de l'expérience des deux ponts (Bonabeau et al., 2001). Cette expérience permet de mettre en évidence le fait que l'utilisation de phéromone pour le recrutement de fourmis sur une piste de nourriture, est un moyen de déterminer le plus court chemin entre une source de nourriture et le nid de la colonie. Sur cette image, il est possible de distinguer des points de couleur rouge représentant la phéromone déposée par les fourmis. Plus la couleur est intense, plus la quantité de phéromone déposée à l'endroit indiqué par le point est importante. Le résultat visible de cette expérience, est que la piste de phéromone la plus marquée est celle correspondant au chemin le plus court entre la nourriture (située en haut à gauche de l'image) et le nid (situé en bas à droite). Ce phénomène est qualifié d'émergent, il n'est nullement spécifié dans le code de l'application.

Nous avons pu nous rendre compte de l'apport des transformations dans le développement de l'application. En considérant le temps de développement global de cette application (3 jours/hommes), nous avons pu nous rendre compte que la plus grosse partie du travail

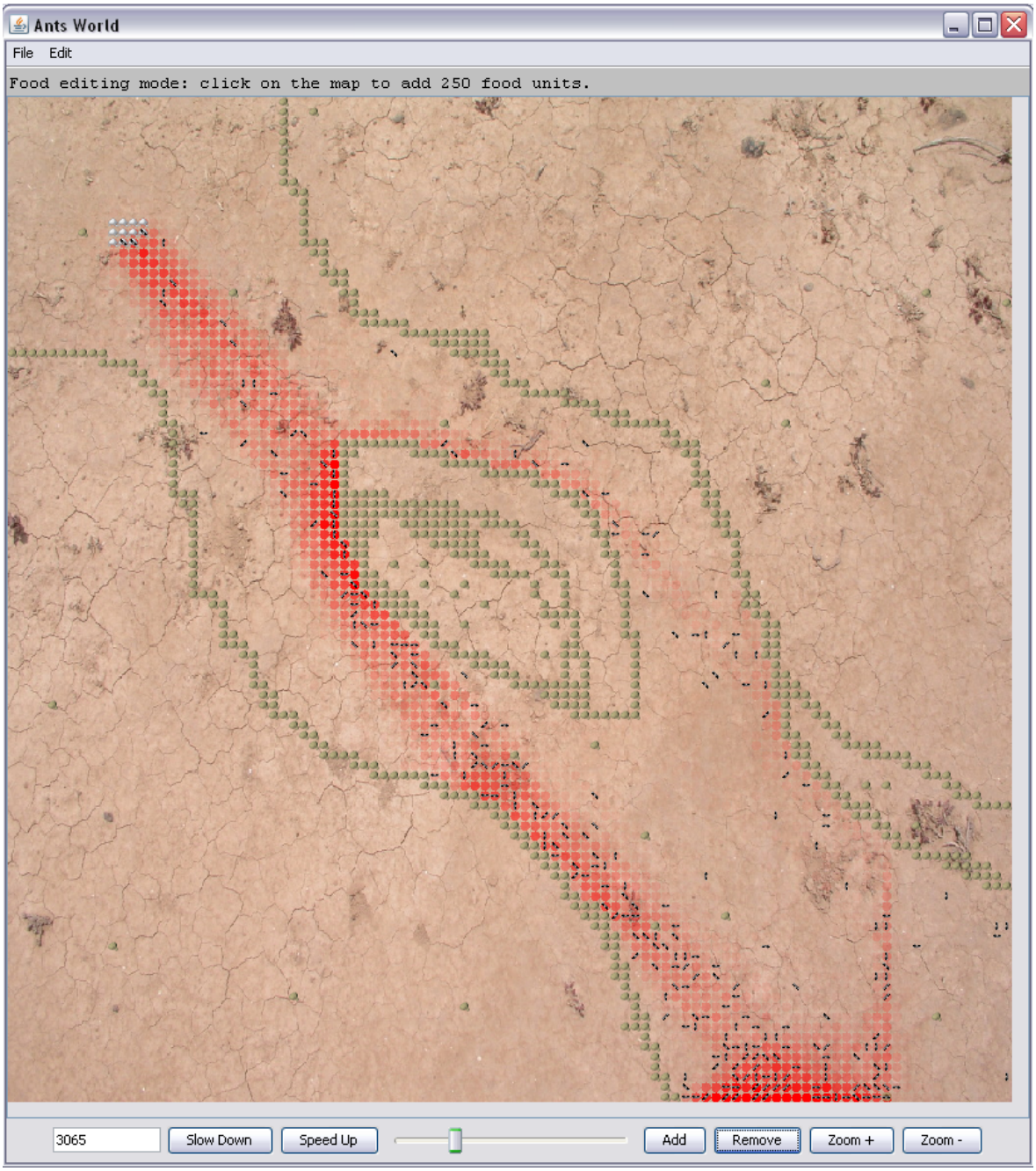

Figure 10.5 — Un exemple de mise en œuvre de l'environnement de simulation Ants : l'expérience des deux ponts.

s'est portée sur l'environnement de simulation et sa mise au point. Le développement des services opératoires de l'agent et de son comportement a quant à lui, été relativement rapide (de l'ordre d'une demi-journée/homme).

11 DAmasCop : système de contrôle manufacturier

DANS ce chapitre, nous détaillons la conception d'une application de contrôle manufacturier appelée DAmasCop. Ce système est conçu pour s'exécuter sur une plate-forme de simulation de gestion de production (MASC) qui est présentée dans les paragraphes qui suivent. Le développement et la modélisation ont été réalisés par une étudiante de master 2 recherche, Elsy Kaddoum, dans le cadre de son stage. Nous avons supervisé les tâches liées à la modélisation spécifique AMAS-ML et avons de cette manière pu tester l'ergonomie et l'adéquation des concepts avec les activités de modélisation et de transformation.

11.1 Contexte

L'informatique et les technologies qui y sont liées sont d'ores et déjà très présentes dans le monde industriel. L'intégration de nouvelles technologies à la chaîne de production peut permettre de mettre en œuvre des approches nouvelles comme l'informatique ubiquitaire, l'intelligence ambiante afin de produire d'une meilleure manière, plus rapidement et plus efficacement. Le problème du contrôle manufacturier ou de gestion de production propose de fournir des moyens de planifier l'utilisation de ressources de production. Cette planification est soumise à un ensemble de contraintes prédéfinies (délais de livraisons, emploi du temps des employés, etc.). La difficulté de cette tâche de planification réside essentiellement dans l'apparition de perturbations imprévisibles produisant de nouvelles contraintes à satisfaire en cours de production (pannes de stations, absence d'un opérateur, etc.). Le système de gestion de production doit ainsi faire face à un environnement dynamique et doit être capable de s'y adapter. Ce type de problèmes peut être rapproché de la satisfaction de contraintes et de la résolution multi-critères présentées dans le paragraphe 1.2.4. Dans un contexte réparti au sein des unités de production et dynamique, une approche par AMAS paraît tout à fait adéquate.

La plate-forme MASC est un outil de simulation, défini dans notre équipe [1], dans le cadre du groupe de travail COLLINE (*« COLlectif, INteraction, Emergence »*) de l'AFIA (Association Française d'Intelligence Artificielle). Ce groupe de travail s'implique dans l'étude et la caractérisation de la notion d'émergence, notamment dans les systèmes multi-agents. Dans ce cadre, divers domaines sont abordés et plusieurs cas d'études ont été définis afin d'étudier l'intérêt des approches multi-agents et de l'émergence. Un des domaines abordés est celui du

1. Cette plate-forme a été développée et est maintenue par André Machonin (SMAC-IRIT), elle est disponible sur le site du groupe COLLINE (`http://www.irit.fr/COLLINE/`)

contrôle manufacturier. L'objet de la plate-forme est de fournir un banc d'essais pour comparer différentes approches sur des scénarios communs dédiés à la simulation de système de gestion de production. Elle permet qui plus est, de proposer de nouveaux algorithmes et de nouveaux scénarios afin de mettre au point des métriques et des critères pour affiner la comparaison des différentes approches. Nous présentons ici l'application d'ADELFE 2.0 pour produire une solution à ce problème de gestion de production et son intégration à la plate-forme MASC.

11.2 Cahier des charges

La plate-forme de simulation MASC permet de simuler une unité de production répondant aux besoins exprimés dans le cahier des charges défini par le groupe COLLINE [2]. MASC propose deux volets : l'un permettant l'édition et la visualisation de scénarios et l'autre permettant le contrôle et l'affichage de la simulation. Chaque simulation permet de dérouler un scénario faisant intervenir trois types d'entités :

- **Opérateur** : une personne possédant un certain nombre de qualifications lui permettant de réaliser différents usinages. Un *opérateur* peut subir des perturbations et être rendu indisponible pour un laps de temps déterminé par le scénario.
- **Conteneur** : contient un ensemble de pièces devant être usinées.
- **Station** : poste de travail permettant à un *opérateur* de réaliser différents types d'usinages sur les pièces d'un *conteneur*. Les stations peuvent subir des pannes dont la durée est spécifiée dans le scénario.

MASC met en œuvre un moteur de simulation qui déroule le scénario pas à pas. A chaque pas de temps, le scénario décrit la répartition des perturbations sur les entités. L'ordonnanceur a pour objectif d'interfacer le simulateur avec les algorithmes de répartition en récupérant la liste des entités disponibles et des perturbations à partir des données du scénario et en les transmettant au système de résolution. Le rôle des algorithmes de répartition est alors d'affecter à chaque pas de temps un ensemble d'opérateurs et de conteneurs sur les stations disponibles. Cette affectation doit tenir compte des qualifications des opérateurs, des stations et des usinages à réaliser sur les pièces du conteneur. L'objectif est d'obtenir une planification tenant compte des différentes priorités et des contraintes temporelles afin que toutes les pièces soient traitées dans le temps imparti. Le résultat du travail du système de résolution est un ensemble de triplets (opérateur, station, conteneur) qui représentent l'affectation d'un opérateur à l'usinage d'un ensemble de pièces sur un poste de travail. C'est donc grâce à l'ordonnanceur qu'il est possible d'intégrer une nouvelle approche de résolution dans la plate-forme, ces relations entre simulateur, ordonnanceur et système de résolution sont résumées dans la figure 11.1. L'implémentation de l'ordonnanceur est à la charge du concepteur du système de résolution. Néanmoins, il doit répondre à une interface précise permettant de le connecter au simulateur déjà implémenté.

2. `http://www.irit.fr/COLLINE/DOCUMENTS/CaseStudy_ManufacturingControl.pdf`

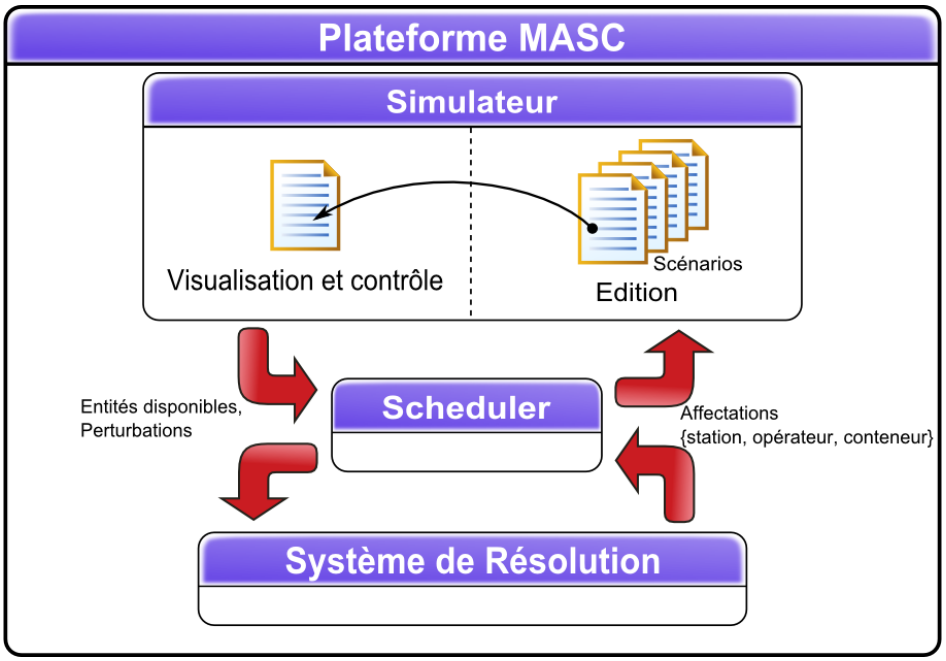

Figure 11.1 — Les différents composants de la plate-forme MASC.

11.3 Conception à l'aide d'ADELFE 2.0

La méthodologie ADELFE 2.0 a été utilisée pour tenter d'apporter une réponse au problème de planification proposé par la plate-forme de simulation MASC. L'approche par AMAS peut être envisagée pour fournir une résolution par auto-organisation des entités constituant le domaine. Nous envisageons ce problème sous l'angle des systèmes auto-organisateurs. En effet, l'utilisation de cette approche s'est avérée particulièrement efficace pour une application comparable de gestion d'emploi du temps (ETTO Emergent TimeTabling Organization) (Picard et al., 2005).

11.3.1 Etudes des besoins

Les deux premières phases d'ADELFE permettent d'établir un cahier des charges précis pour le système en devenir. Cette étude autorise entre autre la détermination des différentes parties, entités du système, qui pourront être intégrés ou non à l'AMAS (agentifiées) délimitant ainsi ses frontières.

11.3.1.1 Préliminaires

Cette phase a pour objet la définition d'un cahier des charges préliminaire en listant un ensemble de mots-clefs et de contraintes. Dans le cadre de la plate-forme MASC le cahier des charges a été défini par le groupe de travail COLLINE. Nous nous en tiendrons à la définition du vocabulaire spécifique au domaine de l'application. Pour ce faire, une liste de

mots-clefs a été décrite :

- **Qualifications** : tâches qu'une station ou un opérateur est capable de réaliser et nécessaires pour le traitement des pièces d'un conteneur ;
- **Conteneurs** : ensemble de pièces à usiner correspondant aux commandes arrivant dans le système ;
- **Stations** : postes de travail permettant de réaliser des tâches précises (Qualifications) ;
- **Opérateurs** : réalisent des tâches sur des stations correspondant à leur qualifications ;
- **Pièces** : appartenant à un conteneur, elles nécessitent des usinages spécifiques (qualifications) ;
- **Rails** : permettent de déplacer les conteneurs entre les différentes stations ;
- **Planning** : résultat attendu du système de résolution respectant les délais et les coûts de production ;
- **Perturbations/Pannes** : interruptions de l'activité d'une machine ou d'un opérateur pendant un certain laps de temps ;
- **Ressources** : besoins matériels et humains nécessaires pour la réalisation des tâches ;
- **Responsable de production** : personne qui surveille le système et veille à son bon fonctionnement.
- **Responsable des ressources** : personne gérant les ressources matérielles (matières premières, outils, etc.) et humaines (opérateurs)

L'ensemble de ces éléments constitue la base de sur laquelle s'organise l'étude du système à développer. En plus de ces termes spécifiques au problème, nous pouvons déterminer un ensemble de contraintes et de limitations à prendre en compte. Ces contraintes sont réparties au sein des différents éléments précédemment identifiés et particulièrement au sein des entités de production :

1. **Contraintes des opérateurs** :
   - ils ne peuvent traiter que des conteneurs pour lesquels ils sont qualifiés ;
   - ils ne peuvent travailler qu'à une seule station à la fois ;
   - ils doivent limiter leurs déplacements au sein de l'usine.
2. **Contraintes des conteneurs** :
   - les pièces qu'ils contiennent doivent être traitées dans un ordre précis (contraintes de précédences entre les qualifications) ;
   - leur traitement est contraint par leur date d'arrivée et doit être terminé avant une date limite (contraintes temporelles) ;
   - ils ne peuvent être traités que par une machine et un opérateur disponibles et possédant la qualification adéquate (contraintes sur les ressources) ;
   - leurs déplacements doivent être minimisés dans l'usine ;
   - ils ne peuvent quitter l'usine qu'après le traitement de la totalité de leurs pièces.
3. **Contraintes des stations** :
   - une station ne peut traiter un conteneur que si elle possède la qualification adéquate ;
   - une station ne peut traiter qu'un conteneur à un instant donné ;
   - une station ne peut commencer le traitement d'un conteneur que si l'opérateur adé-

quat est disponible.

Le respect de ces contraintes doit être maintenu tout au long de la simulation en toutes circonstances (pannes des stations ou absences des opérateurs).

11.3.1.2 Finals

Nous devons déterminer dans cette phase, l'ensemble des entités du système et caractériser précisément son environnement. Il s'agit de modéliser le système dans son environnement et de spécifier les interactions entre celui-ci et les entités extérieures. Cette activité permet également de préciser la nature de chacune des entités identifiées en fonction de leurs interactions avec le système. Nous pouvons identifier les entités actives suivantes :

- **Conteneur** : il définit les tâches à réaliser et peut ainsi permettre de déterminer la meilleure station pour l'accueillir ;
- **Station** : elle possède un comportement autonome, elle peut choisir les conteneurs et les opérateurs pour les traiter ;
- **Opérateur** : peut choisir la station sur laquelle travailler en fonction de ses perceptions ;
- **Responsable de production** : il gère la production et veille au fonctionnement du système, il intervient sur les contraintes temporelles en modifiant les dates de fin ou de début d'usinage des conteneurs ;
- **Responsable des ressources** : il gère l'ensemble des ressources matérielles et humaines et peut modifier les contraintes de disponibilité des stations et des opérateurs.

Nous pouvons, de même, distinguer les entités dites passives, c'est-à-dire ne possédant pas de comportements autonomes : elles subissent les actions des entités actives. Dans notre exemple, nous considérons les entités suivantes comme passives :

- **Pièce** : elle appartient à un conteneur et est usinée sur une station par un opérateur ;
- **Rails** : dans cet exemple, nous ne gérons pas le déplacement des conteneurs, les rails sont utilisés par les conteneurs comme un moyen de relier deux stations distantes ;
- **Ressources matérielles** : elles sont manipulées par le responsable des ressources ;
- **Atelier** : il s'agit du lieu de production, il fournit les cycles d'usinages à l'ensemble des entités.

En ce qui concerne les interactions dans le cadre de la plate-forme MASC, l'ensemble des interactions entre les différents responsables et les entités sont inscrites dans le scénario. Ces interactions sont effectuées lors du déroulement du scénario, elles sont à la charge du simulateur et non du système de résolution qui nous intéresse. C'est pourquoi nous ne les décrivons pas dans cet exemple.

Nous pouvons caractériser l'environnement suivant les critères suivants :

- **Dynamique** : les différents événements pouvant affecter le système comme l'arrivée des conteneurs, la modification de leur délai d'usinage, les perturbations affectant les stations et les opérateurs sont imprévisibles pour le système.
- **Non accessible** : seul le responsable de production chargé de veiller à la cohérence du système possède une vue d'ensemble de l'état du système. Les autres entités n'ont,

quant à elles, qu'une connaissance partielle ou locale de cet environnement.

– **Non déterministe** : l'effet d'une action d'une des entités du système dépend fortement de l'état dans lequel se trouve le système. Par exemple, une station qui reçoit la demande d'utilisation d'un conteneur peut l'accepter ou le refuser en fonction de son propre état. Les actions n'ont pas un effet unique déterminé.
– **Discret** : les événements introduits dans le système ne sont pris en compte qu'au début de chaque cycle d'usinage. Cette discrétisation du temps est liée aux besoins spécifiques de la simulation et de son contrôle. L'environnement est composé d'un nombre dénombrable d'entités, bien que l'association de ces entités entres elles produit un résultat combinatoire. L'environnement peut par conséquent être considéré comme discret.

A l'issue de cette phase, nous avons déterminé l'ensemble des entités actives et passives du système et caractérisé son environnement. Ces informations préliminaires sont essentielles pour analyser la solution en devenir. L'étude des besoins fonctionnels liés à la gestion de la chaîne de production est représentée par un diagramme de cas d'utilisation (voir figure 11.2). Ce diagramme fait intervenir plusieurs stéréotypes permettant d'identifier les entités (actives ou passives) et les interactions pouvant conduire à des situations de non coopération (*«CooperationFailure»*). Ici il s'agit des cas d'utilisation déterminés par le cahier des charges du groupe de travail. Cependant, pour ce qui concerne notre système de résolution voué à être simulé sur la plate-forme MASC, les rôles des différents gestionnaires sont tenus par une seule entité active : l'ordonnanceur (*Sheduler*).

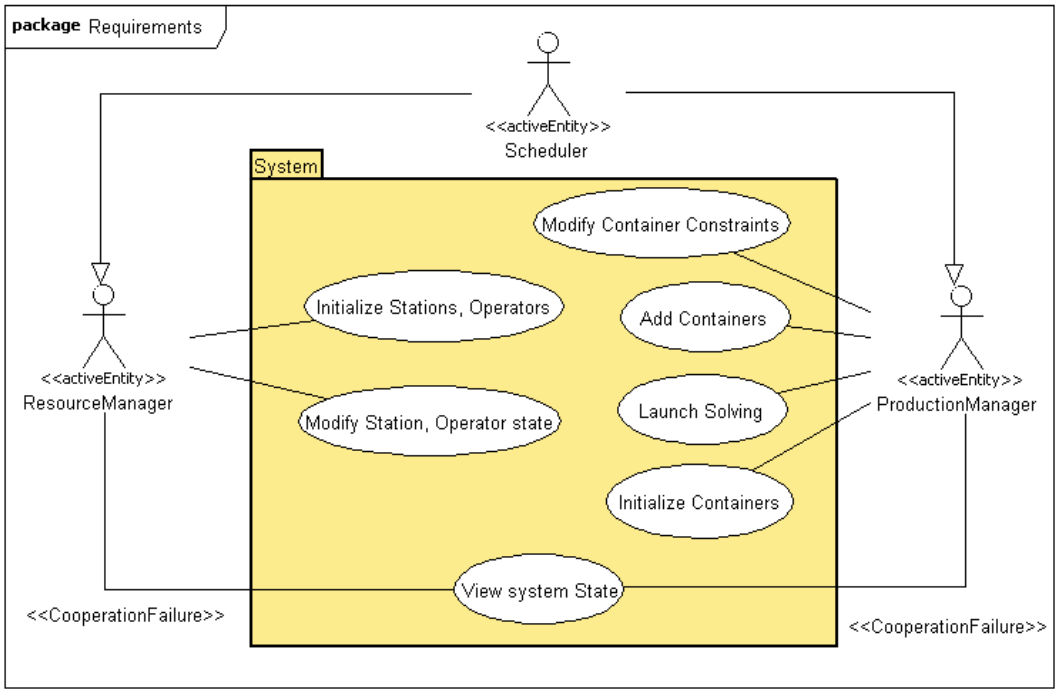

Figure 11.2 — Diagramme de cas d'utilisation du système.

11.3.2 Analyse

Cette étape est cruciale au sein de la méthodologie ADELFE ; elle permet en premier lieu de déterminer l'adéquation du problème avec l'approche AMAS (Activité 11, cf. paragraphe 6.3). Pour cette première activité, nous devons répondre à un ensemble de questions dont les réponses caractérisent cette adéquation au niveau global (ensemble du système) ou au niveau local (au sein des entités) :

- **Niveau global** : il n'existe pas d'algorithme permettant de décrire la résolution du problème posé bien que le résultat attendu soit connu : un ordonnancement des tâches d'usinage tenant compte des contraintes temporelles et des disponibilités. Le fonctionnement du système est physiquement et logiquement distribué au sein des différentes entités (stations, opérateurs et conteneurs). En outre, l'environnement du système a été caractérisé comme dynamique en raison de l'apparition de perturbations imprévisibles. Ces propriétés du système constituent des raisons pour utiliser l'approche distribuée et auto-organisatrice proposé par les AMAS. Il reste cependant à étudier le système au niveau de chacun de ses constituants.
- **Niveau local** : les entités représentant le niveau local possèdent un comportement autonome, une perception et une connaissance limitée de leur environnement. Leur activité est supposée stable et relativement simple (négociation), elle n'est pas susceptible d'évoluer. Une approche AMAS ne semble donc pas nécessaire au niveau local, les entités n'ont pas besoin d'être elles-mêmes décomposées en systèmes auto-organisateurs.

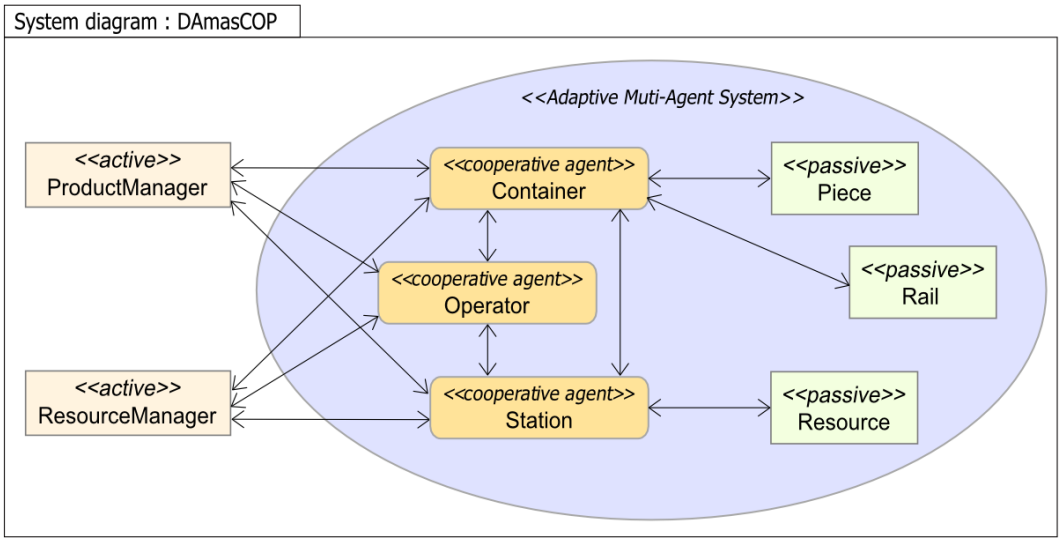

Figure 11.3 — Diagramme environnement/système pour la plate-forme MASC.

Le système sera par conséquent spécifié comme un AMAS. Les activités suivantes d'ADELFE ont pour objet la détermination de l'architecture du système et notamment l'identification des agents. L'activité 12 permet de choisir parmi les entités du système celles qui doivent être considérées comme des agents coopératifs (cf. paragraphe 6.3). Pour identifier les agents, nous examinons les critères de sélection des entités suivants : leur autonomie de décision, la poursuite d'un but local, la capacité de négocier, des capacités d'interaction et

une perception locale de l'environnement. L'un des principes clefs de l'approche AMAS est la coopération. Celle-ci définit la manière de résoudre des situations exceptionnelles appelées SNC (cf. paragraphe 6.1.3). Cette notion caractérise particulièrement les agents coopératifs. Nous avons donc considéré les entités remplissant les critères précédents et susceptibles de faire face à ces situations pouvant mettre en défaut la résolution. Pour ces raisons, nous considérons les opérateurs, les stations et les opérateurs comme des agents coopératifs. Par exemple, la coopération d'un conteneur dont l'ensemble des pièces n'a pas été traité et n'étant affecté à aucune station, est mise en défaut. Il doit tenter de résoudre cette état de fait en trouvant une station adéquate pour l'accueillir.

Le diagramme présenté dans la figure 11.3 regroupe les résultats de cette phase d'analyse. Il représente à un haut niveau d'abstraction les agents coopératifs et leurs interactions avec les différentes entités dynamiques et statiques qui composent l'environnement. Dans notre cas et pour la suite de la conception les entités passives reliées aux agents seront directement intégrées au sein de ces agents. En effet, les pièces d'un conteneur peuvent être considérées comme un ensemble de qualifications et n'ont pas besoin de constituer un élément autonome du système. De la même manière, les responsables (ressources et production) bien qu'appartenant au cahier des charges [3] n'apparaissent dans la plate-forme de simulation MASC qu'au travers des événements du scénario (perturbations, pannes, etc.).

11.3.3 Conception

Le but de cette phase est de fournir une architecture logicielle suffisamment précise pour permettre l'implantation du système. Dans le cas de la méthodologie ADELFE 2.0 elle permet de constituer un modèle spécifique aux AMAS qui sera utilisé comme une entrée de la phase d'implémentation.

11.3.3.1 Modélisation des agents : structure et comportement

Nous présentons ici le résultat de l'étude détaillée de l'architecture des agents coopératifs du système de résolution pour la plate-forme MASC. La phase de conception nécessite également l'étude de leurs interactions et de leurs comportements.

1. **Container**
   Au lancement de la simulation, le conteneur est initialisé avec un ensemble de stations. Au fur et à mesure de son parcours dans l'atelier, il enregistre les stations qu'il rencontre (mise à jour de la représentation *informationAboutStations*). À chaque cycle d'usinage, les conteneurs n'étant pas encore affectés à un poste de travail choisissent parmi les stations qu'ils connaissent (*chooseStation()*) celle qui peut leur être utile. Ce choix s'opère en privilégiant les stations les plus proches de leur position courante (*currentPosition*). Dès lors qu'une station a été sélectionnée, le conteneur en demande l'accès en précisant la qualification nécessaire ainsi que sa priorité (*askToUseStation(Work)*). Il se met

3. Celui proposé par le groupe COLLINE.

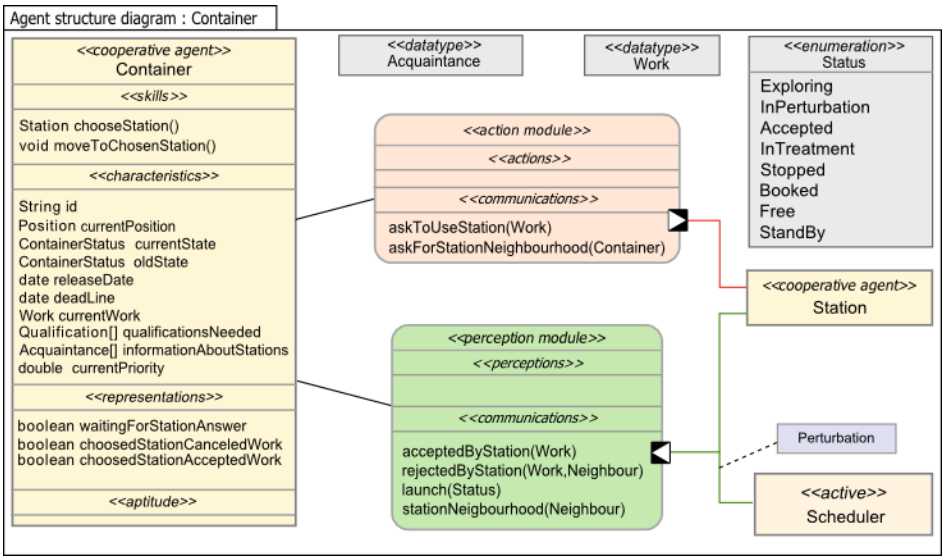

Figure 11.4 — Conception détaillé de l'agent container.

ensuite en attente d'une réponse de la part de la station choisie (*waitingForStationAnswer*). Si aucune station n'a été sélectionnée, il interroge une station connue afin de récupérer son voisinage (*askForStationNeighbourhood()*). Une fois ce voisinage récupéré (*stationNeighbourhood(neighbourhood)*), le conteneur actualise ses représentations (*informationAboutStations*) et peut à nouveau tenter de trouver une station susceptible de l'accueillir. Lorsqu'une station a été choisie, les réponses que celle-ci fournit peuvent être de natures différentes :

– **Acceptation** (*acceptedByStation(Work)*) : dans ce cas le conteneur modifie son état courant (passage de l'état *Exploring* à l'état *Accepted*), met à jour sa position (*currentposition*) et arrête son exploration ;
– **Rejet** (*rejectedByStation(Work, Neighbourhood)*) : le conteneur récupère le voisinage de cette station et l'intègre à ses connaissances. Il choisit alors une autre station et continue son exploration.

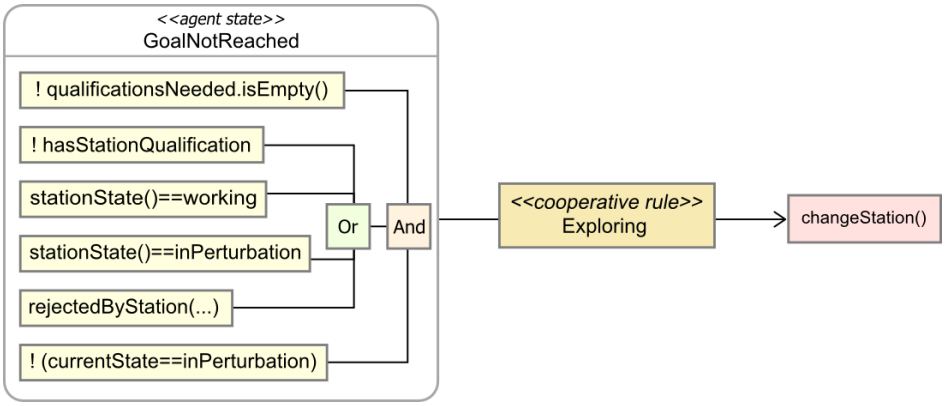

Figure 11.5 — Comportement de l'agent conteneur : exemple de règles coopératives *Exploring* pallie l'improductivité

En outre, une fois le conteneur accepté par une station, il peut tout de même être rejeté (*rejectedByStation(Work, Neighbourhood)*) si celle-ci a dû accepter la demande d'un conteneur plus urgent. Le conteneur est alors contraint de reprendre son exploration. Le comportement de l'agent décrit en langage naturel ci-dessus, est spécifié sous forme de règles (coopératives ou standards). La figure 11.5 en présente un exemple. Il s'agit de la règle comportementale décrivant la recherche d'une station adéquate, les conditions exprimées en prémisse de cette règle caractérise l'exploration de l'atelier (*Eploring*).

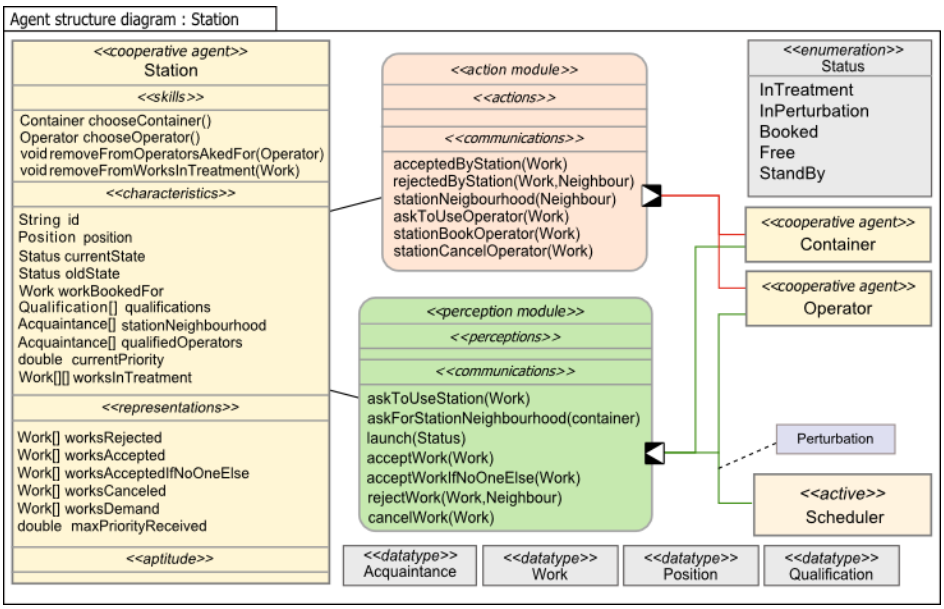

Figure 11.6 — Conception détaillé de l'agent station.

2. **Station**
   La figure 11.6 représente l'architecture détaillée de l'agent station sous forme d'un diagramme agent AMAS-ML (cf. paragraphe 8.2). L'intérêt des différents éléments qu'il contient est précisé au travers d'une description informelle du comportement de l'agent :
   (a) Chaque station lors de son initialisation reçoit l'ensemble des opérateurs qualifiés qu'elle peut accueillir (*qualifiedOperator*) et une liste de stations (avec leur qualification) constituant son voisinage (*neighbourhood*).
   (b) À chaque cycle d'usinage, la station sélectionne parmi les demandes reçues des conteneurs (*worksDemand*) celles apparaissant comme les plus urgentes (*chooseContainer()*).
   (c) Elle envoie alors des requêtes aux opérateurs qualifiés pour réaliser les travaux sélectionnés (*askToUseOperator(Work)*). Les conteneurs les moins prioritaires reçoivent une notification de rejet (*rejectedByStation(Work, neighbourhood)*).
   (d) Dès la réception des réponses des opérateurs, la station sélectionne l'opérateur le moins sollicité par les autres stations (*chooseOperator()*) et le réserve (*stationBookOperator(Work)*). Elle accepte ensuite le conteneur correspondant (*acceptedByStation(Work)*) et rejette l'ensemble des conteneurs correspondant aux demandes

non satisfaites (pas d'opérateur) en leur fournissant des informations concernant son voisinage (rejectedByStation(Work, neighbourhood)).

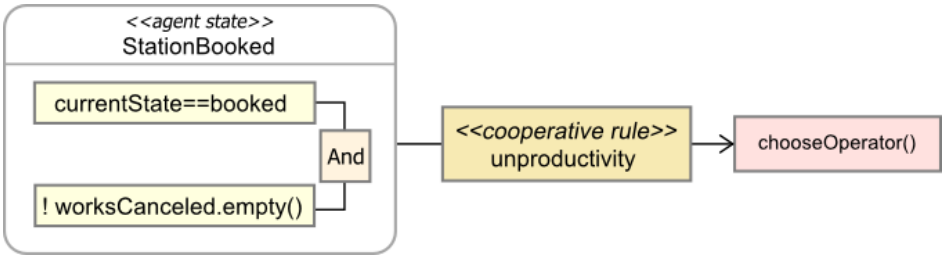

Figure 11.7 — Comportement de l'agent station.

Une fois le conteneur accepté, la station est réservée *Booked*. Si en attendant la réponse des opérateurs, des demandes plus urgentes sont perçues, la station rejette les demandes en cours et tente de satisfaire ces nouvelles requêtes. Une station réservée peut toutefois recevoir de nouvelles demandes. Dans ce cas, deux choix s'offrent à elle :

– si les demandes reçues sont plus urgentes que le travail en cours, la station cherche un opérateur qualifié pour réaliser ces travaux plus urgents. Dans le cas où un opérateur est trouvé, la station annule la réservation courante (conteneur et opérateur *stationCancelOperator()*), rejette les autres demandes et accepte le nouveau conteneur. Le cas échéant, elle rejette le conteneur le plus urgent et réitère le processus avec les conteneurs suivants. La station n'annule la réservation que si elle est capable de satisfaire une demande plus urgente.
– si les demandes sont moins prioritaires, la station les rejette.

Lors de la sélection d'un opérateur, la station favorise les opérateurs libres (ayant répondu *acceptWork(Work)*) sur les opérateurs qui peuvent se libérer (ayant répondu *acceptWorkIfNoOneElse(Work)*). Si plusieurs opérateurs possèdent la même priorité, elle privilégie l'opérateur réservé au cycle d'usinage précédent, à condition qu'il soit qualifié et disponible pour le nouveau travail à réaliser. Elle permet de cette façon aux opérateurs de minimiser leurs déplacements dans l'usine. De plus, un opérateur sélectionné peut refuser la demande d'une station ; dans ce cas la station cherche un nouvel opérateur pour réaliser l'usinage nécessaire au conteneur accepté. Il s'agit d'un comportement coopératif ayant pour objectif d'éviter l'improductivité de la station. Elle a accepté un conteneur et doit donc essayer de le traiter (cf. figure 11.7).

3. **Opérateur**

Nous présentons le résultat de l'étude détaillée de l'agent opérateur au sein d'un diagramme d'agent (cf. figure 11.8). Celui-ci regroupe les propriétés que nous avons considérées comme essentielles pour que l'agent opérateur accomplisse ses tâches. En effet, pour chaque cycle d'usinage, un opérateur détermine la station la plus prioritaire (*chooseStation()*) parmi les demandes reçues (*worksDemand*). Il accepte alors cette proposition de travail en précisant son propre degré de priorité (*acceptWork(Work)*). Si la station confirme sa demande (*stationBookOperator(Work)*) l'opérateur est alors réservé, son état devient *booked*. Il rejette toutes les autres demandes reçues (*rejectWork(Work)*). En

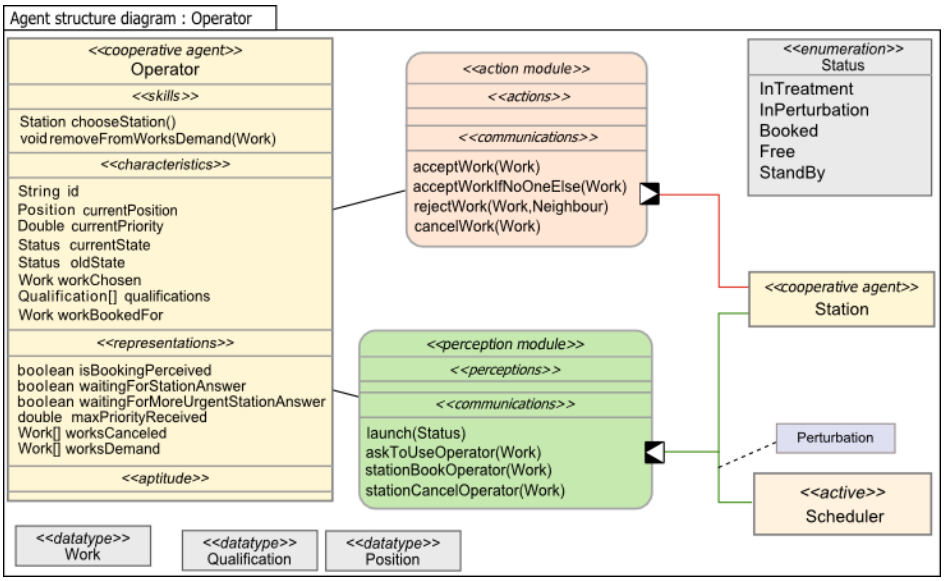

Figure 11.8 — Conception détaillé de l'agent opérateur.

revanche, si une station annule sa demande (*stationCancelOperator(Work)*), l'opérateur actualise la liste des demandes et accepte celles provenant d'une autre station. Une fois l'opérateur réservé par une station, de nouvelles demandes peuvent tout de même lui parvenir, deux cas se présentent :

– si les demandes perçues sont plus urgentes, l'opérateur informe la station la plus urgente qu'il peut se libérer si aucun autre opérateur n'est disponible (*acceptIfNoOneElse(Work)*, cf. règle comportementale présentée dans la figure 11.9). Il attend ensuite la confirmation de cette station avant d'annuler (cancelWork(Work)) la réservation actuelle. Dans le cas d'un rejet, il réitère l'opération avec les autres demandes urgentes. La réservation courante ne sera annulée que dans le cas de l'acceptation d'une demande plus urgente ;
– si les demandes perçues ne sont pas prioritaires, l'opérateur les rejette (rejectWork(Work)).

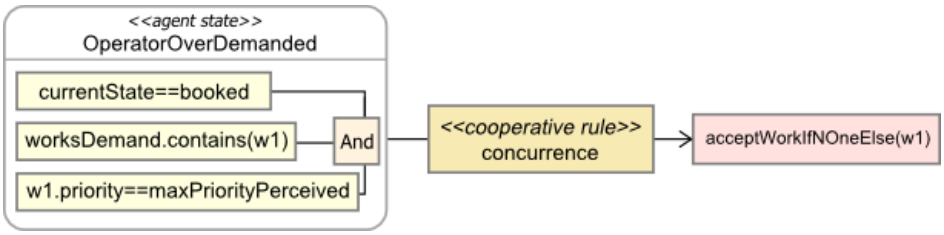

Figure 11.9 — Comportement de l'agent opérateur.

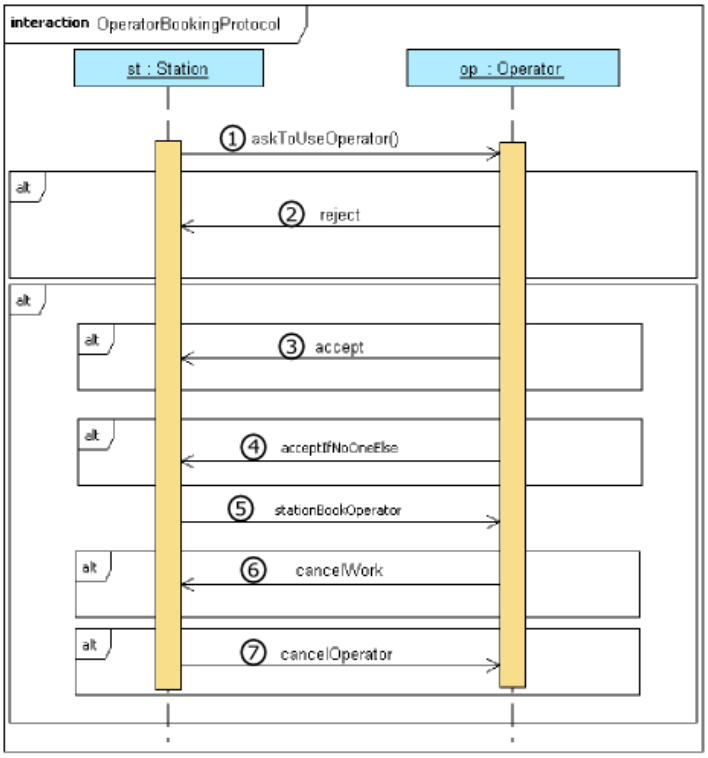

Figure 11.10 — Protocole d'interactions station / opérateur.

11.3.3.2 Interactions

Les agents de DAmasCop, coopèrent pour satisfaire les différentes contraintes qui leurs sont imposées dans le cadre du traitement d'un ensemble de pièces (temporelles, ressources, etc.). Cette coopération se caractérise particulièrement par une négociation entre les différents agents. Cette négociation fait intervenir un ensemble de messages que nous avons regroupés au sein de protocoles (*CooperatveInteractionProtocole* cf. paragraphe 8.2). Ces protocoles ont été représentés grâce à des diagrammes de séquences UML 2.0 et intégrés dans le modèle AMAS-ML par transformation (voir paragraphe 1). La figure 11.10 représente le protocole de négociation mis en place lors de la réservation d'un opérateur par une station :

1. Une station demande à un opérateur de sa connaissance s'il est disponible pour réaliser un usinage (*askToUseOperator(Work)*).
2. L'opérateur peut rejeter la demande s'il est d'ores et déjà occupé sur une autre station (*reject(Work)*),
3. il peut également accepter sans condition la demande s'il est libre (*accept(Work)*),
4. ou accepter la demande s'il n'y a personne d'autre pour la satisfaire dans le cas où son travail actuel est moins prioritaire (*acceptIfNoOneElse(Work)*)
5. La station traite les différentes réponses des opérateurs et en réserve un ayant accepté la demande (*stationBookOperator(Work)*).

6. Une station (*cancelWork(Work)*), si un travail plus prioritaire a été proposé à la station ou à l'opérateur,
7. il en est de même pour l'opérateur peut remettre en question la réservation (*cancelOperator(Work)*).

11.3.4 Implantation

La première tâche de la phase d'implantation consiste à extraire du modèle AMAS-ML de conception les caractéristiques opératoires de chaque type d'agent coopératif. Cette étape est automatisée au travers d'une transformation de modèle et produit pour chaque agent coopératif un modèle de micro-architecture. La transformation AMAS-ML2μADL ((cf. paragraphe 9.3)) produit dans notre cas, trois modèles représentants les agents coopératifs conteneur, station et opérateur que nous souhaitons voir apparaître dans le système.

11.3.4.1 Modèle μADL

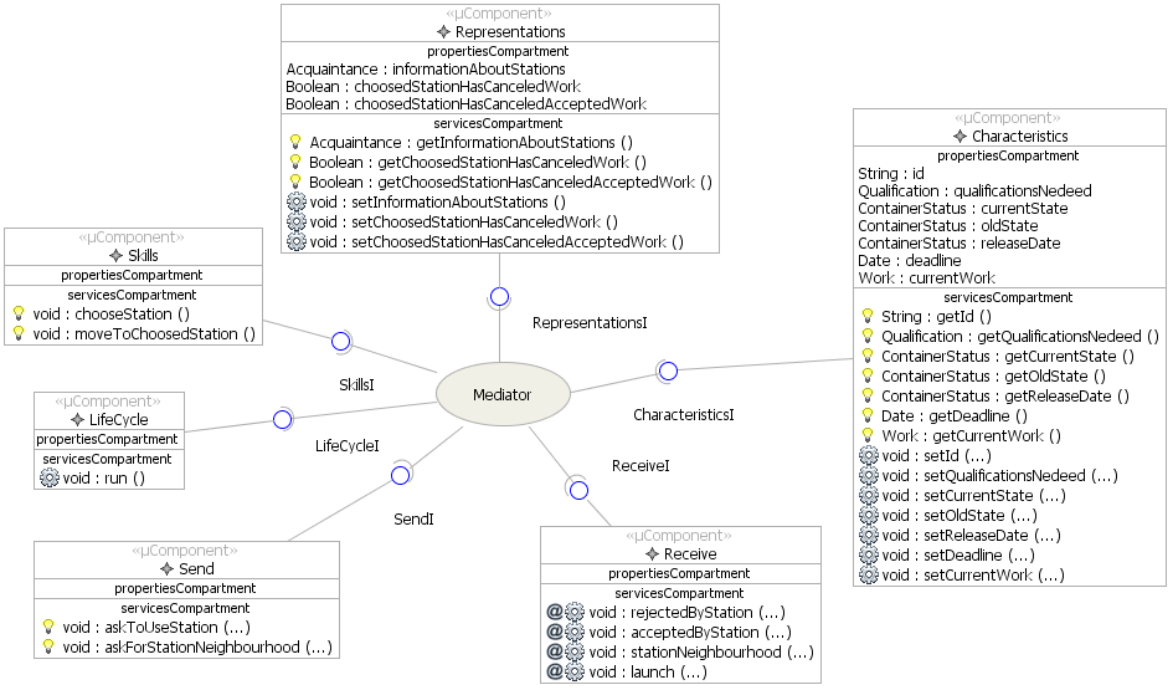

Figure 11.11 — Micro-architecture d'un agent coopératif conteneur.

La figure 11.11 représente le résultat de la transformation du modèle AMAS-ML de l'agent coopératif conteneur. La gestion de ses représentations et de ses caractéristiques est répartie dans deux micro-composants distincts. Ces deux types de connaissance sont représentés par des propriétés structurelles dont l'accès est limité par une ensemble de service. Elle sont accessibles en lecture au niveau comportemental puisqu'utilisées pour déterminer les actions à mener (processus de décision). En revanche, leur mise à jour reste transparente

au niveau décisionnel, elle est à la charge de la phase de perception du cycle de vie et dépend uniquement (dans le cas du conteneur) des messages reçus. Les composants *Send* et *Receive* gèrent ces messages comme des appels de méthodes (éventuellement distant). Les messages qu'est capable de traiter un conteneur sont contenus dans le composant *Receive* et sont accessibles depuis l'extérieur de l'agent (*isExternal* positionné à vrai). Cette accessibilité est symbolisée par un arobas. A l'inverse, les messages qui peuvent être envoyés à un autre agent correspondent à des services de niveau de base. Ils représentent les moyens que possède un conteneur pour agir sur son environnement (les stations dans le cas du conteneur). Les compétences du conteneur sont spécifiées comme des services comportementaux (accessible par le niveau comportemental) offerts par le micro-composant *Skill*.

Cette répartition en termes de services et composants est la même pour les agents station et conteneur. Néanmoins, chaque modèle d'agent définit ses propres connaissances et messages. Bien évidemment, l'implémentation des mécanismes de gestion de ces micro-composants différera en fonction du modèle d'agent auquel ils sont associés (conteneur, station et opérateur).

11.3.4.2 Code comportemental

Le comportement des agents consiste essentiellement a traité les messages reçus et à fournir une réponse aux demandes en fonction de leur état. La station joue un rôle prépondérant dans le processus d'allocation puisqu'elle reçoit les demande des conteneurs et doit réserver en conséquence les opérateurs adéquats. En fonction de leurs réponses elle accepte ou rejette les demandes des conteneurs en attente. La portion de code de la figure 11.12 présente une partie du processus de décision du cycle de vie de l'agent coopératif station. Il s'agit du traitement des messages de refus reçus de la part des opérateurs.

11.4 Conclusion

Le code et l'approche de coopération mise en place pour résoudre ce problème de contrôle manufacturier dans DAmasCop donne des résultats très satisfaisants. Le temps d'exécution s'avère très faible malgré une coopération basée essentiellement sur la négociation et un grand nombre de messages échangés. Ceci est dû essentiellement à la légèreté de l'intergiciel qui supporte le code de l'application. Les résultats des expérimentations sur la plate-forme MASC et les comparaisons avec d'autres approches de résolution sont présentés dans (Clair et al., 2008).

Du point de vue de la conception nous avons pu réaliser l'intérêt de la mise en place de la démarche de modélisation spécifique. Cela a permis effectivement de faire remonter au niveau de la conception toutes les questions liées à la détection et au traitement de situations de non-coopération. De surcroît, cela a permis d'exprimer clairement les règles de coopération en fonction des propriétés de l'agent chose qui jusqu'à lors n'était pas

```
private void decideWorkRejected(StationStatus currentStatus) {
   int nbOfRejectedWork = getWorksRejected().size();
   switch (currentStatus){
      case Free:
      case Booked:
         //La station supprime les travaux pour lesquels elle a
         //demandé un opérateur qui a refusé le travail
         for(int i=0; i<nbOfRejectedWork; i++){
            Work work = getWorksRejected().get(i);
            if(worksContain(work.getContainer(),
                  getWorksInTreatment())){
               if(otherOperatorsAskedFor(work)){
                  removeWork(work, getWorksInTreatment());
                  removeOperatorAskedFor(work.getOperator());
               }else{
               //Elle annule le conteneur si tous les opérateurs
               //demandés ont refusé le travail
                  removeContainerWorks(work.getContainer(),
                                 getWorksInTreatment());
                  addWorkToRejectContainer(work);
                  removeOperatorAskedFor(work.getOperator());
               }
            }
         }
         setWorksRejected(new Vector<Work>());
         break;
         //En traitment ou en perturbation, la station ne demande
         //pas d'opérateurs ne reçoit donc pas de messages de rejet
      default:
         break;
   }
}
```

Figure 11.12 — Phase de décision de l'agent station : traitement des messages de refus reçus de la part des opérateurs.

possible. D'un point de vue moins pragmatique, la modélisation graphique des règles comportementales et des agents a également facilité la présentation et la communication autour de leur mise au point.

Cette application nous a servi de test de validation de l'approche et a pu faire apparaître les points qui doivent encore être améliorés. Ces aspects sont discutés dans le chapitre qui suit.

12 Synthèse et analyse

LES exemples d'utilisation d'ADELFE 2.0 et de son approche dirigée par les modèles nous permettent d'envisager leurs résultats selon différents axes. Tout d'abord en étudiant quels sont les apports de notre démarche pour les concepteurs de SMA adaptatif. Nous pouvons, en outre, étudier les bénéfices d'une approche dirigée par les modèles pour faciliter l'implantation de ces systèmes. Enfin, nous pouvons interpréter ces résultats d'un point de vue plus général et considérer cette nouvelle phase d'implémentation comme un élément méthodologique réutilisable.

Nous nuancerons ces résultats en présentant les limitations et les améliorations qui peuvent être amenées à notre démarche.

12.1 Point de vue du concepteur

L'utilisation des diagrammes AMAS-ML a montré son intérêt dans la conception des agents coopératifs. Tout d'abord, en proposant au concepteur un niveau d'abstraction et des concepts directement représentatifs de son domaine d'expertise. C'est l'intérêt principal d'une démarche spécifique au domaine, elle permet de porter l'attention du concepteur sur les aspects qui sont particulièrement important pour le système en devenir (par exemple, la notion de coopération). Au delà de ces considérations, qui valent pour l'ensemble des approches spécifiques à un domaine, nous pouvons constater des progrès significatifs dans la conception détaillée des agents coopératifs. Grâce au méta-modèle AMAS-ML nous fournissons aux concepteurs des concepts précis, un langage pour décrire dans un même modèle l'agent, ses propriétés, ses capacités et son comportement. Jusqu'à lors, la méthodologie ADELFE préconisait l'utilisation d'une représentation tabulaire informelle pour l'expression des situations de non coopération (Picard, 2004). Cette représentation n'est cependant pas utilisable pour générer du code.

En outre, l'expression des règles comportementales en fonction des connaissances et des caractéristiques des agents a naturellement induit un processus progressif et itératif dans la conception précise des propriétés et du comportement des agents. La définition des règles comportementales permet en effet de mettre en avant des besoins autant en termes de représentations ou de connaissances que d'actions. A l'inverse, l'ajout d'éléments structurels au diagramme d'agent peut entraîner l'apparition de nouvelles situations (états) qui doivent être traitées par de nouvelles règles comportementales (coopératives ou non). Ainsi, cette tâche de conception du comportement permet de mettre au point le détail des caractéristiques de

l'agent. Grâce au modèle spécifique et aux diagrammes qui y sont associés, ces deux tâches de conception bénéficient l'une de l'autre et permettent d'accélérer et de faciliter cette phase.

12.2 Point de vue du développeur

L'introduction d'une phase d'implémentation orientée modèle dans ADELFE (cf. paragraphe 7.2.4) a apporté au développement de SMA adaptatif un degré plus élevé d'automatisation. Les développeurs profitent d'outil qui les aident dans la production d'application orientée agent, domaine dont ils ne sont pas nécessairement spécialistes. En effet, MAY offre aux développeurs un moyen de produire leur propre API orientée agent avec un minimum d'effort, c'est-à-dire en se concentrant sur l'implantation de micro-composants. De plus, les principes architecturaux, le langage μADL et l'environnement de génération proposés par MAY autorisent la réutilisation à plusieurs niveaux. Tout d'abord, au niveau du diagramme d'architecture μADL, en intégrant des micro-composants prédéfinis et finalement dans le choix des classes Java implantant ces micro-composants.

Grâce au principe d'agent flexible et à l'outil MAY, nous avons introduit un niveau d'abstraction supplémentaire au sein même de l'API agent utilisé pour développer le code de l'application finale. Ce qui permet de réduire le fossé qui sépare la conception de l'implémentation. Les concepts utilisés dans les modèles du système se retrouvent directement au niveau de l'implémentation du comportement des agents. Ainsi, la responsabilité de ce développement peut être donnée à un expert MAS lui permettant de maîtriser tous les aspects du processus décisionnel de l'agent sans avoir à s'impliquer dans les mécanismes mis en place au sein des actionneurs ou des capteurs.

Si l'on considère un point de vue plus pragmatique, l'implantation et la mise au point du modèle d'agent ForagingAnt présentée dans le chapitre 10 (mécanismes opératoires et comportement) n'a nécessité que trois jours-homme. Bien entendu, il s'agit d'un exemple relativement simple et bien connu, cependant, cela démontre tout de même l'intérêt d'une approche dirigée par les modèles. De cette manière, l'implantation est facilitée puisque l'effort est porté sur la conception plutôt que sur le développement.

Un autre intérêt de l'approche que nous proposons est le gain de performance essentiellement dû à la légèreté des applicatifs produits. La plate-forme d'accueil des modèles d'agent est constitué de quatre classes, d'une interface de marquage (*Agent*), auxquelles s'ajoutent celles issues du processus de génération des micro-composants (une classe par micro-composant). Dans le cas de la simulation de fourmi l'API Agent spécifique est constituée de 17 classes et 9 interfaces pour un poids relatif compilé de 53 ko (contre plus de 2 Mo pour la plateforme Jade par exemple), l'environnement du système représente un ensemble de 29 classes (69 ko) le comportement de l'agent et le main de l'application, quant à eux ne représente que 2 classes (6 ko). La légèreté des applicatifs produits, est due aux dernières phases de génération de code (utilisation de MAY) permettant de produire une plate-forme agent dédiée aux problèmes spécifiques rencontrés par le système en devenir. Elle ne contient, de cette façon, que ce qui a été modélisé et qui apparaît comme nécessaire

pour le système. L'exemple présenté dans le chapitre 11 a démontré son efficacité en termes de temps d'exécution par rapport à des approches implémentées sur des plate-formes agent comme Jade. En effet, pour un nombre de messages échangés bien supérieur aux approches implémentées sous Jade, DAmasCOP établit une solution en un temps extrêmement court (de l'ordre de 700 ms) (Clair et al., 2008).

12.3 Point de vue de l'ingénieur méthode

Cette section présente le point de vue du concepteur de processus de développement et particulièrement dans le cadre de la réutilisation d'élément méthodologique : la notion de fragment. Le principe de fragment est soutenu par la FIPA et son groupe de travail (fip, 2003) et partage ses principes avec l'*ingénierie méthodologique de situation* SME (Situational Method Engineering) comme le présente (Seidita et al., 2006). Les fragments méthodologiques de la FIPA sont étroitement liés au méta-modèle SPEM 1.0 de l'OMG, ils en utilisent les principaux concepts : *Activités, Artefacts, Rôle*, etc. Depuis, cette norme a évolué, nous avons proposé dans (Rougemaille et al., 2008) d'utiliser les capacités de la version 2.0 de SPEM (omg, 2007a) pour implémenter la notion de fragment. Nous avons proposé une traduction du processus ADELFE grâce à l'outil de définition de processus du projet EPF (Eclipse Process Framework) [1] basé sur la dernière version de SPEM. Nous avons proposé une nouvelle phase d'implémentation et l'avons décrite grâce aux concepts de SPEM 2.0 (cf. paragraphe 7.2.4), ce qui nous permet de bénéficier de ses capacités de réutilisation et de séparation des préoccupations. Nous considérons, ainsi, que cette phase d'implémentation constitue un fragment réutilisable. Notamment, grâce à son approche dirigée par les modèles ne dépendant que du modèle du domaine qui constitue son entrée (le modèle AMAS-ML dans notre cas). Par conséquent, elle pourrait être considérée comme un fragment méthodologique (Cossentino et al., 2007a) paramétrée par le modèle d'entrée et la transformation associée. C'est-à-dire un modèle spécifique et la traduction de ces mécanismes opératoires en une micro-architecture μADL. On peut cependant objecter que cette transformation pourrait poser un problème, mais nous supposons que l'ingénierie dirigée par les modèles offre déjà des moyens d'assister sa définition. Il existe par exemple, des langages comme AMW (Atlas Model Weaver) (Fabro and Valduriez, 2007) pour exprimer des relations qui peuvent permettre de générer des modèles de transformations [2]. En outre, le méta-modèle cible, à savoir celui de μADL, offre un ensemble relativement réduit de concepts qui peuvent constituer des cibles plus évidentes. Toutefois, la génération de transformations constitue toujours un challenge dans le monde de l'ingénierie des modèles.

1. `http://www.eclipse.org/epf/`
2. `http://www.eclipse.org/gmt/AMW/`

Conclusion

TOUT au long de ce document, nous nous sommes attachés à présenter le travail mené pour introduire l'ingénierie dirigée par les modèles dans le développement des systèmes multi-agents adaptatifs. Nous considérons que cette approche associée à un support applicatif léger et adaptable peut permettre de maîtriser les défis que représentent la conception et l'implantation de systèmes informatiques complexes.

Synthèse

Les travaux présentés dans ce document ont permis d'atteindre une bonne partie des objectifs que nous nous étions fixés (voir chapitre 3) :

– *Prise en compte de l'adaptation*

Un des axes majeurs qui ont motivé ces travaux est le besoin récurrent en adaptation au sein des systèmes artificiels considérés comme complexes. Notre intérêt porte tout particulièrement sur la notion d'auto-adaptation dans laquelle le système modifie sa fonctionnalité ou sa structure et maintient ainsi, son intégrité face aux aléas du monde qui l'entoure. Ce principe est particulièrement incarné par les systèmes auto-organisateurs et notamment les AMAS qui mènent à bien cette tâche en suivant des critères de coopération. Nous avons souhaité ajouter une dimension à ces systèmes en proposant d'offrir la possibilité d'adapter à la fois la fonction du système (par auto-organisation grâce à la coopération) et la manière dont un agent s'exécute (adaptation opératoire des agents flexibles). Pour permettre une telle mise en commun de capacités nous nous sommes appuyés sur les principes de l'Ingénierie Dirigée par les Modèles. Ceci nous a permis un gain non négligeable en abstraction grâce notamment aux langages de modélisation spécifiques (DSML) que nous avons définis pour chacune de ces dimensions (AMAS-ML et μADL).

– *Formalisation de l'approche SMA adaptatif*

Nous avons spécifié sous la forme d'un méta-modèle les concepts essentiels d'un SMA Adaptatif ainsi que les relations qui les lient. À partir de ce méta-modèle, nous avons défini un langage de modélisation spécifique AMAS-ML qui nous permet de décrire

des modèles structurellement conformes à notre vision des AMAS. Il s'agit de plus, d'un moyen fourni au concepteur pour décrire avec précision des agents coopératifs en garantissant que ceux-ci intègrent bien les concepts nécessaires à leur coopération. De plus, nous permettons grâce à une chaîne de transformations de modèles de garantir que tous ces éléments se retrouveront effectivement dans le code de l'application et en permettront l'auto-adaptation. Il s'agit d'un moyen de valider en amont (à la conception) le fait que le système produit possédera des capacités d'auto-adaptation.

– *Formalisation des comportements coopératifs*

Dans le cadre du langage AMAS-ML, nous avons spécifié le contenu du module de décision d'un agent coopératif. En ouvrant ce module, qui était auparavant une boîte noire, nous permettons au concepteur de définir des règles comportementales en se basant sur les propriétés modélisées de l'agent. En effet, ce dernier est amené à décrire les états de non coopération sous la forme d'une condition impliquant les représentations, caractéristiques et capacités de l'agent coopératif qu'il a précédemment modélisées. De cette manière, il peut se rendre compte de manques ou de besoins spécifiques en termes de nouvelles propriétés ou bien de situations. En outre, ces règles constituent la source d'une génération de code qui fournit une base pour le développement du comportement des agents.

– *Automatisation du développement*

Celle-ci intervient sous la forme de transformations de modèles à plusieurs moments dans le processus de développement. Tout d'abord, pour récupérer les informations du modèle UML issu des phases d'établissement des besoins puis, pour lier AMAS-ML au diagramme de séquences UML afin de permettre la définition d'interactions. Nous avons également automatisé l'extraction des mécanismes opératoires des agents coopératifs du modèle AMAS-ML afin de construire les architectures d'agents flexibles μADL correspondantes. En outre, nous utilisons l'outil de génération MAY (cf. paragraphe 9.4) pour produire une API dédiée aux modèles d'agents décrits grâce à μADL. Enfin, nous avons implémenté une transformation de rétro-ingénierie produisant un modèle μADL à partir du code Java de micro-composants pour conserver la cohérence entre code et modèle.

– *Séparation des préoccupations*

Grâce à l'architecture d'agent flexible, nous avons pu proposer une séparation des préoccupations au sein des agents coopératifs AMAS. Les aspects comportementaux de ces agents, c'est-à-dire le résultat de la modélisation de leurs modules de décision, sert de base au code comportemental. En revanche, les modules d'interactions sont considérés comme des mécanismes opératoires dont la gestion est indépendante du

processus de décision. Cette séparation des préoccupations est implantée sous la forme d'une transformation de modèles AMAS-ML vers μADL.

D'un point de vue méthodologique, l'adoption d'une approche dirigée par les modèles nous a permis d'intégrer au sein d'un même processus de développement, celui d'ADELFE, plusieurs langages de modélisation. L'UML est utilisé dans les phases préliminaires de la méthodologie puisque celles-ci sont directement basées sur le processus unifié, il y est donc particulièrement adapté. Les transformations de modèles permettent, en créant des ponts, d'utiliser à chaque instant le langage le plus adéquat. L'UML est utilisé pour la description des besoins et des interactions, AMAS-ML pour la conception détaillée des agents et μADL pour la modélisation de l'architecture de ces agents. Nous avons pu compléter le processus ADELFE d'une phase d'implémentation qui utilise ces principes de l'IDM pour générer le code de l'application (le comportement des agents qui la composent) et le code de la plate-forme qui les accueille (les capacités spécifiques des agents, le support d'exécution et de déploiement).

Perspectives

Notre travail se situe à la frontière de plusieurs domaines, celui des systèmes multi-agents, de l'ingénierie dirigée par les modèles ou plus largement du génie logiciel. Aussi, nous pouvons organiser les différentes perspectives selon le domaine auquel elles se rapportent.

– **Techniques liées à l'IDM**

L'utilisation d'une démarche de modélisation spécifique nous a permis d'évaluer la maturité de l'approche et des outils qui y sont associés, spécifiquement dans le cadre de la plate-forme Eclipse. Si l'on considère la notion de méta-modèle et son utilisation pratique pour l'implantation de langages de modélisation, nous pouvons constater que de nombreuses améliorations pourraient y être apportées. L'un des premiers inconvénients auxquels il faut faire face est le problème de la modification des méta-modèles. Le développement d'un langage de modélisation et de ses éditeurs impliquent de nombreuses itérations. Chaque modification du méta-modèle liée à l'intégration d'un nouveau concept ou d'une nouvelle relation, nécessite la régénération ou la reprise de l'ensemble des outils s'y référant. De même, les modèles précédemment conformes au méta-modèle ne peuvent plus être pris en compte par les outils mis à jour pour respecter le nouveau méta-modèle. Il existe des solutions ad-hoc définies par les fournisseurs d'outils pour permettre la migration des modèles d'une version à une autre. Cependant, les capacités génératives de l'IDM pourraient être mises à profit afin de produire automatiquement une transformation réalisant cette migration à partir de deux versions distinctes d'un méta-modèle.

La notion même de modèle et de méta-modèle pourrait bénéficier d'une approche plus modulaire, en composant un modèle à partir de plusieurs autres. Il s'agirait de

s'approcher de la notion d'aspect mais en l'appliquant à un niveau d'abstraction plus élevé. Des travaux existent dors et déjà pour mettre ces principes en œuvre (Fabro and Valduriez, 2007). Nous pourrions définir un méta-modèle en plusieurs modèles distincts dont chacun représenterait un aspect différent, organisé autour d'un modèle central représentatif du domaine (comparable à une ontologie). C'est à peu de chose près, l'approche utilisée par l'outil de génération GMF (cf. paragraphe 8.3.3). Cependant, la mise en relation des différents aspects reste manuelle et relativement fastidieuse. De plus, elle ne concerne que l'édition graphique des modèles.

Tout ceci constitue une partie des défis identifiables dans le domaine de l'IDM. Apporter des réponses à ces problèmes ne constitue pas directement un de nos objectifs. Cependant, il nous semble important de relever les imperfections que nous avons pu noter tout au long de la mise en place de notre processus dirigé par les modèles afin de faire bénéficier de nôtre expérience en la matière.

– **Méthodologiques**

Le domaine de l'AOSE (Agent Oriented Software Engineering) fournit depuis quelques années déjà des outils permettant de modéliser et pour certains d'automatiser l'implantation des systèmes multi-agents. Cependant, ces outils nécessitent pour la plupart une bonne connaissance du paradigme agent utilisé et du processus proposé par la méthodologie associée. Pour étendre l'utilisation des SMA à une plus large audience, nous souhaiterions que ces outils intègrent un guide méthodologique qui puisse répondre aux questions d'un utilisateur novice. Dans ce cadre, nous envisageons de proposer un environnement de développement complet permettant d'assister la réalisation des différentes tâches du processus. A plus longue échéance, cette activité d'assistance pourrait être dévolue à un système auto-adaptatif. Il garantirait dynamiquement la cohérence des modèles, proposerait les tâches à venir et anticiperait sur les besoins de l'utilisateur en se basant sur le modèle du système en devenir et la description du processus.

Un tel système d'assistance s'avèrerait particulièrement utile pour indiquer au concepteur d'AMAS les interactions pouvant éventuellement provoquer des situations de non coopération et lui proposer les outils (AMAS-ML) lui permettant de les décrire. Une première étape vers cet objectif a été réalisée en décrivant dans un cadre unifié, celui d'Eclipse et de son plugin EMF, la méthodologie, son processus, les langages de modélisation et les traitements automatiques de modèles (transformations et génération de code). ADELFE 2.0 a également été décrite à l'aide du plugin EPF (Eclipse Process Framework) qui produit un modèle de processus conforme au SPEM 2.0 en respectant le standard XMI (Ecore). Ce modèle de processus peut ainsi être utilisé pour conduire le ou les utilisateurs de la méthodologie en interagissant avec les outils de modélisation et de transformations eux-mêmes définis dans Eclipse.

Exprimer des processus sous forme de modèles SPEM peut également faciliter la description et l'utilisation de fragments méthodologiques (Rougemaille et al., 2008).

Il s'agirait alors de récupérer des briques, des composants méthodologiques pour les assembler et former en cours de développement un nouveau processus. Cette composition de fragment « à la volée » s'appuierait sur les tâches déjà réalisées par les acteurs du processus et sur une spécification précise du fragment. SPEM permet en effet, de décrire des éléments méthodologiques réutilisables en précisant leurs entrées et sorties en termes de produits. Cette capacité pourrait être étendue pour exprimer des compositions de fragments. Une utilisation envisageable de ces fragments serait de décrire un outil de développement adaptatif capable de proposer à ses utilisateurs les tâches les plus adaptées au moment ou ils en ont besoin. En sélectionnant parmi une banque de fragments le plus approprié vis-à-vis de l'état d'avancement courant (fragments déjà utilisés).

– **Liées aux AMAS**

Dans le domaine qui nous intéresse plus spécifiquement, celui des AMAS, nous avons pu constaté l'intérêt d'une approche utilisant des modèles spécifiques. Cela nous a permis de formaliser précisément des aspects primordiaux de la conception des agents notamment, l'expression de la coopération. Nous envisageons toutefois, de fournir une modélisation plus fine encore du comportement. En effet, l'expression du comportement des agents coopératifs sous forme de règles pose un problème de niveau d'abstraction. Nous exprimons les règles comportementales au niveau du modèle mais nous aimerions qu'elles se réfèrent à des valeurs qui dépendent de l'exécution du système. Il faudrait donc disposer de concepts de modélisation supplémentaires afin de permettre l'expression plus précise de types et d'instances de ces types. Par exemple, pour donner les valeurs d'instances des paramètres d'une capacité ou d'une action et non la chaîne de caractères qui les décrit. Néanmoins, la description d'un méta-modèle de type complet (instances et valeurs) ne constituait pas à l'origine un objectif primordial pour nos travaux aussi nous n'avons pas poursuivi davantage cette piste, il s'agit d'une amélioration à apporter pour la suite d'AMAS-ML.

Nous souhaitons également permettre la simulation et la mise au point du système au niveau des modèles. Ceci implique une extension du méta-modèle pour prendre en compte des concepts liés à l'exécution (notion de temps principalement). Concrètement, cette exécution pourrait être simulée grâce à des langages de transformations, ou de méta-programmation tel que Kermeta (Muller et al., 2005a). Ceci permettrait de vérifier la cohérence du comportement des agents avant leur implantation.

Toutes ces perspectives entrent dans le cadre d'un projet de pré-industrialisation d'ADELFE et de son processus dirigée par les modèles. Le travail que nous avons présenté a permis entre autres de valider l'approche IDM spécifique au domaine des AMAS et d'établir ainsi la faisabilité d'un tel projet. Nos résultats constituent une première étape qui pourra être poursuivie par le développement d'un atelier complet alliant langages spécifiques, génériques et transformations de modèles.

Bibliographie

(2003). *MDA Guide Version 1.0.1, OMG document ad/2002-04-10 (2002)*. Object Management Group, Framingham, Massachusetts.

(2003a). *Meta Object Facility (MOF) 2.0 Core Specification*. Object Management Group, Inc., OMG edition.

(2003). *Method fragment definition*. FIPA, fipa document edition.

(2003b). *UML Object Constraint Language (OCL) 2.0 Specification*. Object Management Group, Inc., OMG edition. Final Adopted Specification.

(2003c). *Unified Modeling Language (UML) 2.0 Infrastructure Specification*. Object Management Group, Inc., OMG edition. Final Adopted Specification.

(2003d). *Unified Modeling Language (UML) 2.0 Superstructure Specification*. Object Management Group, Inc., OMG edition. Final Adopted Specification.

(2005a). *MOF QVT Final Adopted Specification*. Object Management Group Inc., OMG edition.

(2005b). *Software Process Engineering Metamodel (SPEM) 1.1 Specification*. Object Management Group, Inc., OMG edition. formal/05-01-06.

(2007a). *Software & Systems Process Engineering Metamodel Specification v2.0*. Object Management Group, Inc., OMG edition.

(2007b). *Unified Modeling Language (UML) 2.1.1 Superstructure Specification*. Object Management Group, Inc., OMG edition. Final Adopted Specification.

Aarts, Emile (2005). Ambient intelligence drives open innovation. *interactions*, 12(4) :66–68.

Aarts, Emile (2006). New Interfaces - How To Interact with a Connected World : Ambient Intelligence. In *PICNIC'06 Cross media week*.

Abbott, R. (2006). Complex Systems + Systems Engineering = Complex Systems Engineering. Position paper presented at Conference on Systems Engineering Research.

Abbott, R. (2007). Putting complex systems to work. *Complexity*, 13(2) :30–49.

Appel, K. and Haken, W. (1977). Every planar map is four-colorable. *Ill. J. Math.*, 21 :429–567.

Arcangeli, J.-P. and Leriche, S. (2007). Construction d'agents auto-adaptatifs à base de micro-composants opératoires. In Camps, V. and Mathieu, P., editors, *Journées Francophones des Systèmes Multi-Agents, Carcassonne, 17/10/2007-19/10/2007*, pages 75–84, http ://www.cepadues.com/. Cépaduès Editions.

ATLAS (2005). KM3 : Kernel metametamodel. Technical report, LINA & INRIA, Nantes.

Beni, G. (2004). From swarm intelligence to swarm robotics. In Sahin, E. and Spears, W. M., editors, *Swarm Robotics*, volume 3342 of *Lecture Notes in Computer Science*, pages 1–9. Springer.

Bernon, C., Camps, V., Gleizes, M.-P., and Picard, G. (2005). Engineering Adaptive Multi-Agent Systems : The ADELFE Methodology . In Henderson-Sellers, B. and Giorgini, P., editors, *Agent-Oriented Methodologies*, volume ISBN1-59140-581-5, pages 172–202. Idea Group Pub, NY, USA.

Bertolini, D., Delpero, L., Mylopoulos, J., Novikau, A., Orler, A., Penserini, L., Perini, A., Susi, A., and Tomasi, B. (2006). A tropos model-driven development environment. In Boudjlida, N., Cheng, D., and Guelfi, N., editors, *CAiSE Forum*, volume 231 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Bézivin, J. (2006). Model driven engineering : An emerging technical space. In Lämmel, R., Saraiva, J., and Visser, J., editors, *GTTSE*, volume 4143 of *Lecture Notes in Computer Science*, pages 36–64. Springer.

Bézivin, J. and Gerbé, O. (2001). Towards a precise definition of the OMG/MDA framework. In *ASE*, pages 273–280. IEEE Computer Society.

Bézivin, J. and Jouault, F. (2005). Using atl for checking models. In *GraMoT*.

Bonabeau, E., Dorigo, M., and Theraulaz, G. (2001). Swarm Intelligence : From Natural to Artificial Systems. *J. Artificial Societies and Social Simulation*, 4(1).

Booch, G., Rumbaugh, J. E., and Jacobson, I. (1999). The unified modeling language user guide. *J. Database Manag*, 10(4) :51–52.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. (2004). Tropos : An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3) :203–236.

Budinsky, F., Steinberg, D., and Ellersick, R. (2003). *Eclipse Modeling Framework : A Developer's Guide*. Addison-Wesley Professional.

Busetta, P., Rönnquist, R., Hodgson, A., and Lucas, A. (1999). JACK intelligent agents - components for intelligent agents in java.

Camazine, S., Franks, N. R., Sneyd, J., Bonabeau, E., Deneubourg, J.-L., and Theraula, G. (2001). *Self-Organization in Biological Systems*. Princeton University Press, Princeton, NJ, USA.

Capera, D., Georgé, J.-P., Gleizes, M.-P., and Glize, P. (2003). The AMAS Theory for Complex Problem Solving Based on Self-organizing Cooperative Agents . In *TAPOCS 2003 at WETICE 2003, Linz, Austria, 09/06/03-11/06/03*. IEEE CS.

Chella, A., Cossentino, M., Sabatucci, L., and Seidita, V. (2006). Agile PASSI : An agile process for designing agents. *Comput. Syst. Sci. Eng*, 21(2).

Clair, G., Kaddoum, E., Gleizes, M.-P., and Picard, G. (2008). Approches multi-agents auto-organisatrices pour un contrôle manufacturier intelligent et adaptatif. In *Journées Francophones sur les Systèmes Multi-Agents (JFSMA), Brest, 15/10/2008-17/10/2008*.

Clark, T., Sammut, P., and Willans, J. (2008). *Applied Metamodelling, a Foundation for Language Driven Development*, volume Second Edition. Ceteva. `http://www.ceteva.com/docs/Applied+Metamodelling+(Second+Edition).pdf`.

Colorni, A., Dorigo, M., and Maniezzo, V. (1991). Distributed optimization by ant colonies. In *Proceedings of ECAL91 - European Conference on Artificial Life,*. Elsevier Publishing.

Combemale, B., Rougemaille, S., Crégut, X., Migeon, F., Pantel, M., and Maurel, C. (2006). Sémantique dans la méta-modélisation. In *2ieme journées sur l'Ingénierie Dirigée par les Modèles (IDM'06), Lille, France, 26/06/06-28/06/06*, pages 17–33. Université de Lille. ISBN 2-7261-1290-8.

Cossentino, M., Fortino, G., Garro, A., Mascillaro, S., and Russo, W. (2008). PASSIM : a simulation-based process for the development of multi-agent systems. 2 :132–170.

Cossentino, M., Gaglio, S., Garro, A., and Seidita, V. (2007a). Method fragments for agent design methodologies : from standardisation to research. *Int. J. of Agent-Oriented Software Engineering*, 1 :91–121.

Cossentino, M., Gaud, N., Galland, S., Hilaire, V., and Koukam, A. (2007b). A holonic metamodel for agent-oriented analysis and design. In Mavrík, V., Vyatkin, V., and Colombo, A. W., editors, *HoloMAS*, volume 4659 of *Lecture Notes in Computer Science*, pages 237–246. Springer.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection*. John Murray, London.

Ehrig, K., Ermel, C., Hänsgen, S., and Taentzer, G. (2005). Towards graph transformation based generation of visual editors using eclipse. *Electr. Notes Theor. Comput. Sci*, 127(4).

Fabro, M. D. D. and Valduriez, P. (2007). Semi-automatic model integration using matching transformations and weaving models. In Cho, Y., Wainwright, R. L., Haddad, H., Shin, S. Y., and Koo, Y. W., editors, *SAC*, pages 963–970. ACM.

Farail, P., Gaufillet, P., Canals, A., Camus, C. L., Sciamma, D., Michel, P., Crégut, X., and Pantel, M. (2006). the TOPCASED project : a toolkit in open source for critical aeronautic systems design. In *Embedded Real Time Software (ERTS)*, Toulouse.

Favre, J.-M. (2004). Towards a basic theory to model model driven engineering. In *Workshop on Software Model Engineering, WISME 2004, joint event with UML2004*.

Favre, J.-M., Estublier, J., and Blay, M. (2006). *L'ingénierie dirigée par les modèles : au-delà du MDA*. Hermes-Lavoisier. ISBN : 2-7462-12-12-7.

Ferber, J. (1999). *Multi-Agent Systems—An Introduction to Distributed Artifical Intelligence*. Addison-Wesley.

Ferber, J. and Gutknecht, O. (1998). A meta-model for the analysis and design of organizations in multi-agent systems.

Forrest, S. (1990). Emergent computation : self-organizing, collective, and cooperative phenomena in natural and artificial computing networks. *Physica D*, 42 :1–11.

France, R. B. and Rumpe, B. (2005). Domain specific modeling. *Software and System Modeling*, 4(1) :1–3.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison Wesley, Massachusetts.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison Wesley, Reading, Mass.

Georgé, J.-P., Edmonds, B., and Glize, P. (2004). Making self-organising adaptive multiagent systems work. In Bergenti, F., Gleizes, M.-P., and Zombonelli, F., editors, *Methodologies and Software Engineering for Agent Systems*, pages 319–338. Kluwer Academic Publishers.

Georgé, J.-P., Gleizes, M.-P., and Glize, P. (2003). Conception de systèmes adaptatifs à fonctionnalité émergente : la théorie des AMAS. *Revue d'Intelligence Artificielle*, 17(4/2003) :591–626.

Gleizes, M.-P. (2004). *Vers la résolution de problèmes par émergence*. Habilitation à diriger des recherches, Université Paul Sabatier, Toulouse, France.

Gleizes, M.-P., Camps, V., and Glize, P. (1999). A Theory of Emergent Computation based on Cooperative Self-organization for Adaptive Artificial Systems. In *Fourth European Congress of Systems Science , Valencia Spain, 20/09/99-24/09/99*. Pages de la publication : ..

Glize, P. (2001). *L'Adaptation des Systemes a Fonctionnalite Emergente par Auto-Organisation Cooperative*. Habilitation à diriger des recherches, Université Paul Sabatier, Toulouse, France.

Goldstein, J. (1999). Emergence as a Construct : History and Issues. *Emergence : Complexity and Organization*, 1(1) :49–72.

Gomez-Sanz, J. and Pavon, J. (2003). *Agent Oriented Software Engineering with INGENIAS*, volume Multi-Agent Systems and Applications III. Springer.

Heylighen, F. (2001). The science of self-organization and adaptivity. In *The Encyclopedia of Life Support Systems*. EOLSS Publishers, Paris, France. in print.

Holland, J. H. (1998). Emergence : From chaos to order. *J. Artificial Societies and Social Simulation*, 1(4).

Jacobson, I. (1995). Formalizing use-case modeling. *JOOP*, 8(3) :10–14.

Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process*. Addison-Wesley.

Jarraya, T. and Guessoum, Z. (2007). Towards a model driven process for multi-agent system. In Burkhard, H.-D., Lindemann, G., Verbrugge, R., and Varga, L. Z., editors, *CEEMAS*, volume 4696 of *Lecture Notes in Computer Science*, pages 256–265. Springer.

Jouault, F., Bézivin, J., and Kurtev, I. (2006). TCS : a DSL for the specification of textual concrete syntaxes in model engineering. In Jarzabek, S., Schmidt, D. C., and Veldhuizen, T. L., editors, *GPCE*, pages 249–254. ACM.

Jouault, F. and Bézivin, J. (2006). Km3 : a dsl for metamodel specification. In *Proceedings of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, LNCS 4037*, pages 171–185, Bologna, Italy.

Jouault, F. and Kurtev, I. (2005). Transforming Models with ATL. In *Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005*, Montego Bay, Jamaica.

Kahn, G. (1987). Natural semantics. Report no. 601, INRIA.

Keller, R. (1994). *On Language Change : The Invisible Hand in Language*. Number ISBN 0-415-07671-4. Routledge.

Kinny, D., Georgeff, M. P., and Rao, A. S. (1996). A methodology and modelling technique for systems of BDI agents. In *Proceedings of MAAMAW 1996*, pages 56–71.

Kubera, Y., Mathieu, P., and Picault, S. (2008). Interaction-oriented agent simulations : From theory to implementation. In et Al ed, M. G., editor, *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, pages 383–387. IOS Press.

Laddaga, R. (2001). Active software. *Lecture Notes in Computer Science*, 1936 :11– ? ?

Langton, C. G. (1990). Computation at the edge of chaos. *Physica D*, 42.

Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., IV, C. T., Nordstrom, G., Sprinkle, J., and Volgyesi, P. (2001). The generic modeling environment. In *Workshop on Intelligent Signal Processing*, Budapest, Hungary.

Leriche, S. (2006). *Architectures à composants et agents pour la conception d'applications réparties adaptables*. PhD thesis, Paul Sabatier University, Toulouse, France.

Leriche, S. and Arcangeli, J.-P. (2007). Adaptive Autonomous Agent Models for Open Distributed Systems. In *International Multi-Conference on Computing in the Global Information Technology (ICCGI), Guadeloupe, 04/03/2007-09/03/2007*, pages 19–24, http ://www.computer.org. IEEE Computer Society.

Luck, M., Ashri, R., and d'Inverno, M. (2004). *Agent-Based Software Development*. Artech House, Inc., Norwood, MA, USA.

Mathieu, P. and Picault, S. (2006). Vers une représentation des comportements centrée interactions. In *Actes du 15e congrès francophone AFRIF-AFIA Reconnaissance des Formes et Intelligence Artificielle (RFIA'2006)*.

Maturana, H. R. and Varela, F. J. (1994). *L'arbre de la connaissance (éd. or. 1992)*. Addisson-Wesley.

Medvidovic, N. and Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1) :70–93.

Mens, T. and Gorp, P. V. (2006). A taxonomy of model transformation. *Electr. Notes Theor. Comput. Sci.*, 152 :125–142.

Monperrus, M., Jezequel, J.-M., Champeau, J., and Hoeltzener, B. (2008). *Model-Driven Software Development : Integrating Quality Assurance*, chapter Measuring Models. IDEA Group Inc., Hershey, PA, USA. ISBN : 978-1-60566-006-6.

Muller, P.-A., Fleurey, F., Fondement, F., Hassenforder, M., Schneckenburger, R., Gérard, S., and Jézéquel, J.-M. (2006). Model-driven analysis and synthesis of concrete syntax. In Nierstrasz, O., Whittle, J., Harel, D., and Reggio, G., editors, *MoDELS*, volume 4199 of *Lecture Notes in Computer Science*, pages 98–110. Springer.

Muller, P.-A., Fleurey, F., and Jézéquel, J.-M. (2005a). Weaving executability into object-oriented meta-languages. In *LNCS*, Montego Bay, Jamaica. MODELS/UML'2005, Springer.

Muller, P.-A., Fleurey, F., Vojtisek, D., Drey, Z., Pollet, D., Fondement, F., Studer, P., and Jézéquel, J.-M. (2005b). On executable meta-languages applied to model transformations. Model Transformations In Practice Workshop.

Nicolis, G. and Prigogine, I. (1977). *Self-organization nonequilibrium systems*. Wiley-Interscience, New York.

Odell, J., Parunak, H., and Bauer, B. (2000). *Representing Agent Interaction Protocols in UML*. Springer Verlag.

Padgham, L., Thangarajah, J., and Winikoff, M. (2005). Tool support for agent development using the prometheus methodology. In *Evolvable Hardware*, pages 383–388. IEEE Computer Society.

Padgham, L. and Winikoff, M. (2002). Prometheus : A methodology for developing intelligent agents. In *Proceedings of the Third International Workshop on Agent Oriented Software Engineering at AAMAS*.

Penserini, L., Perini, A., Susi, A., Morandini, M., and Mylopoulos, J. (2007). A design framework for generating BDI-agents from goal models. In Durfee, E. H., Yokoo, M., Huhns, M. N., and Shehory, O., editors, *AAMAS*, page 149. IFAAMAS.

Perini, A. and Susi, A. (2005). Automating model transformations in agent-oriented modelling. In Müller, J. P. and Zambonelli, F., editors, *AOSE*, volume 3950 of *Lecture Notes in Computer Science*, pages 167–178. Springer.

Picard, G. (2003). UML stereotypes definition and AUML notations for ADELFE methodology with opentool.

Picard, G. (2004). *Méthodologie de développement de systèmes multi-agents adaptatifs et conception de logiciels à fonctionnalité émergente*. Thèse de doctorat, Université Paul Sabatier, Toulouse, France.

Picard, G., Bernon, C., and Gleizes, M.-P. (2005). ETTO : Emergent Timetabling Organization. In *International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS), Budapest, Hungary, 15/09/2005-17/09/2005*, pages 440–449, http ://www.springerlink.com/. Springer-Verlag.

Picard, G. and Gleizes, M.-P. (2005). Cooperative Self-Organization to Design Robust and Adaptive Collectives. In *International Conference on Informatics in Control, Automation and Robotics (ICINCO), Barcelona, Spain, 14/09/2005-17/09/2005*, pages 236–241, http ://www.insticc.net/. INSTICC Press.

Plotkin, G. D. (1981). A Structural Approach to Operational Semantics. Technical Report FN-19, DAIMI, University of Aarhus, Denmark.

Reynolds, C. W. (1987). Flocks, Herds, and Schools : A Distributed Behavioral Model, in Computer Graphics. In *SIGGRAPH '87 Conference Proceedings*, pages 25–34.

Robertson, P., Shrobe, H. E., and Laddaga, R., editors (2001). *Self-adaptive software : First International Workshop, IWSAS 2000, Oxford, UK, April 17–19, 2000 : Revised Papers*, volume 1936 of *Lecture Notes in Computer Science*, pub-SV :adr. Springer-Verlag Inc.

Rochfeld, A. (1986). Merise, an information system design and development methodology, tutorial. In Spaccapietra, S., editor, *Entity-Relationship Approach : Ten Years of Experience in Information Modeling, Proceedings of the Fifth International Conference on Entity-Relationship Approach, Dijon, France, November 17-19, 1986*, pages 489–528. North-Holland.

Rouff, C. A., Hinchey, M. G., Truszkowski, W., and Rash, J. L. (2006). Experiences applying formal approaches in the development of swarm-based space exploration systems. *STTT*, 8(6) :587–603.

Rougemaille, S., Migeon, F., and Maurel, C. (2007). Conception d'applications adaptatives basées sur l'idm. In *NOTERE (NOuvelle TEchnologie de la REpartition)*, Marakesh. Accepted.

Rougemaille, S., Migeon, F., Millan, T., and Gleizes, M.-P. (2008). Methodology Fragments Definition in SPEM for Designing Adaptive Methodology : a First Step. In *AOSE Workshop on Agent Oriented Software Engineering, Estoril, Portugal, 12/05/08-13/05/08*, pages 213–224, http ://www.springerlink.com. Springer.

Rumbaugh, J., Jacobson, I., and Booch, G. (1999). *The Unified Modeling Language Reference Manual*. Object Technology Series. Addison Wesley Longman, Reading, Mass.

Rumbaugh, J. E. (1995). OMT : The development process. *JOOP*, 8(2) :8–16, 76.

Russel, S. and Norvig, P. (1995). *Artificial Intelligence : a Modern Approach*. Prentice-Hall.

Schilit, B., Adams, N., and Want, R. (1994). Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US.

Seidita, V., Cossentino, M., and Gaglio, S. (2006). A repository of fragments for agent system design. In Paoli, F. D., Stefano, A. D., Omicini, A., and Santoro, C., editors, *Proceedings of the 7th WOA 2006 Workshop From Objects to Agents*, volume 204 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Selic, B. (2007). The embarrassing truth about software automation and what should be done about it. In Stirewalt, R. E. K., Egyed, A., and Fischer, B., editors, *ASE*, page 3. ACM.

Selic, B. (2008). Filling in the Whitespace : Research Opportunities in Model-Driven Software Development and Model-Driven Engineering. In Ameur, Y. A., editor, *CAL*, volume RNTI-L-2 of *Revue des Nouvelles Technologies de l'Information*, page 3. Cépaduès-Éditions.

Shannon, R. E. (1975). *Systems simulation : the art and science*. Prentice-Hall, Englewood Cliffs.

Shannon, R. E. (1998). Introduction to the art and science of simulation. In *WSC '98 : Proceedings of the 30th conference on Winter simulation*, pages 7–14, Los Alamitos, CA, USA. IEEE Computer Society Press.

Shoham, Y. (1996). Communication and coordination in multi-agent systems : Agent-oriented programming and computational social laws.

Sudeikat, J., Braubach, L., Pokahr, A., and Lamersdorf, W. (2004). Evaluation of agent-oriented software methodologies - examination of the gap between modeling and platform. In Odell, J., Giorgini, P., and Müller, J. P., editors, *AOSE*, volume 3382 of *Lecture Notes in Computer Science*, pages 126–141. Springer.

Szyperski, C., Gruntz, D., and Murer, S. (2002). *Component Software : Beyond Object-Oriented Programming*. ACM Press and Addison-Wesley, New York, NY, second edition edition.

Topin, X., Fourcassié, V., Gleizes, M.-P., Theraulaz, G., and Régis, C. (1999). Theories and experiments on emergent behaviour : From natural to artificial systems and back. In *European Conference on Cognitive Science, Siena.*

Van De Vijver, G. (1997). Emergence et explication. *Intellectica : Emergence and explanation*, 2(25) :185–194.

Weiser, M. (1999). The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3) :3–11.

Welcomme, J.-B., Gleizes, M.-P., and Redon, R. (2007). A Self-Organising Multi-Agent System Managing Complex System Design Application to Conceptual Aircraft Design. In *International Conference on Complex Open Distributed Systems (CODS), Chengdu, 22/07/2007-24/07/2007*.

Weyns, D., Parunak, H. V. D., Michel, F., Holvoet, T., and Ferber, J. (2004). Environments for multiagent systems state-of-the-art and research challenges. In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *E4MAS*, volume 3374 of *Lecture Notes in Computer Science*, pages 1–47. Springer.

Weyns, D., Vizzari, G., and Holvoet, T. (2005). Environments for situated multi-agent systems : Beyond infrastructure. In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *E4MAS*, volume 3830 of *Lecture Notes in Computer Science*, pages 1–17. Springer.

Wolfram, S. (1986). *Theory and Applications of Cellular Automata*, volume 1 of *Advanced series on complex systems*. World Scientific.

Wooldridge, M. (1999). Intelligent agents. In Weiss, G., editor, *Multiagent Systems : A Modern Approach to Distributed Artificial Intelligence*, chapter 1, pages 27–78. The MIT Press, Cambridge, MA, USA.

Liste des figures