Transformation de modèles d'agents dans la méthode ADELFE

Des stéréotypes de conception à l'implémentation

Kévin Ottens – Gauthier Picard – Valérie Camps

IRIT - Université Paul Sabatier
118, route de Narbonne
F-31062 Toulouse Cedex
{ottens,picard,camps}@irit.fr

RÉSUMÉ. *Dans cet article, nous présentons un modèle d'agent coopératif utilisé dans la méthode* ADELFE *pour la conception de systèmes multi-agents adaptatifs. Cette méthode repose sur un processus (*Rational Unified Process*) ainsi que des notations (*UML *et* AUML*) issus de l'analyse et de la conception objet et agent. Concernant l'aspect statique de conception, les agents sont décrits en tant que classes dont la structure est contrainte par l'usage de stéréotypes ayant une sémantique forte d'accessibilité sur les différents modules composant un agent coopératif. Les aspects dynamiques, et plus particulièrement les protocoles de communication entre agents, sont spécifiés grâce aux diagrammes de protocoles AUML. Le modèle fonctionnel et structurel ainsi obtenu peut aisément conduire à la génération de code, et ainsi permettre de passer de la conception à l'implémentation, comme proposé dans le cadre de l'ingénierie des modèles, ou MDA (*Model Driven Architecture*).*

ABSTRACT. *In this paper, we present a cooperative agent model used by the* ADELFE *method to design adaptive multi-agent systems. This method is based on object-oriented and agent-oriented process (*Rational Unified Process*) and notations (*UML *and* AUML*). From the static viewpoint, agents are described as classes, the structure of which is constrained by using stereotypes with a strong semantic on the accessibility to the different modules composing a cooperative agent. From the dynamic viewpoint, communications between agents are specified using AUML protocol diagrams. The functional and structural agent model can therefore easily lead to code generation, and then fill the gap between design and implementation phases, as proposed in the MDA (*Model Driven Architecture*) framework.*

MOTS-CLÉS : *Méthode orientée agent, modèle d'agent, stéréotypes, génération de code*

KEYWORDS: *Agent-oriented methodology, agent model, stereotypes, code generation*

2e soumission à *L'objet*, le 27 juin 2006

1. Introduction

La communauté SMA (*Systèmes Multi-Agents*) a beaucoup œuvré dans le domaine des méthodes orientées agent depuis quelques années. En effet, malgré la résonance du paradigme multi-agent dans la communauté scientifique, la technologie n'est pas encore devenue un standard pour l'industrie. Nous pouvons attribuer, en partie, ce défaut de reconnaissance au fait que la communauté multi-agent n'a pas encore (ou pas assez en tout cas) développé les outils pédagogiques de communication vers les non spécialistes (étudiants et entreprises). Ceci inclut des livres de méthodes, des notations spécifiques et des outils facilitant la réalisation de tels systèmes.

Depuis quelques années les méthodes orientées agent fleurissent dans le domaine académique (Arlabosse *et al.,* 2004, Bergenti *et al.,* 2004, Henderson-Sellers *et al.,* 2005) – chacune ayant ses spécificités : modèle d'agent, formalisme, paradigme de programmation, domaine d'application, etc. Pourtant, cet effort méthodologique se limite souvent aux phases préliminaires du cycle de vie du logiciel : l'analyse et la conception. Face à ce constat, nous proposons une réflexion sur la base de la méthode ADELFE[1] pour le développement de systèmes multi-agents adaptatifs (Picard, 2004, Picard *et al.,* 2004, Georgé *et al.,* 2003). Dans la continuité du processus de développement existant, nous discutons des outils à mettre en œuvre pour faciliter la génération de code à partir de modèles d'agents donnés – comme par exemple, le modèle d'agent coopératif dans ADELFE. Cette approche est positionnée dans le cadre du MDA (*Model Driven Architecture*) de l'OMG (*Object Management Group*), dont le but est de raisonner sur les modèles et leurs règles de transformation tout au long du développement logiciel, jusqu'à l'implémentation, voire la maintenance (Kleppe *et al.,* 2003). Une première tentative d'utilisation du MDA dans le cadre d'une ingénierie des agents (à notre connaissance) est établie dans le projet Méta-DIMA (Thiefaine *et al.,* 2003). L'intérêt d'une telle démarche est une amélioration de la structuration du système afin de dégager des composants logiciels de granularités différentes et réutilisables (agents, constituants des agents, protocoles, etc.).

Dans la section 2, nous présentons le modèle d'agent coopératif propre à la méthode ADELFE et grâce auquel nous illustrerons notre propos. Les méthodes orientées agent, leurs apports et leurs manques seront discutés dans la section 3. Le processus actuel de la méthode ADELFE, sur lequel se greffent les outils proposés, sera décrit dans la section 4. Dans cette méthode, l'utilisation du modèle d'agent est contrainte par une extension du métamodèle via un stéréotypage spécifique et l'utilisation des protocoles AUML (Huget *et al.,* 2004). Ces aspects statiques et dynamiques seront développés dans la section 5. Dans la section 6, nous présenterons les méthodes et techniques mises en œuvre et leur positionnement par rapport à l'approche MDA, pour le passage de la conception (diagrammes de classes, machines à états et protocoles) à l'implémentation de manière semi-automatique. Enfin, ce travail est discuté et positionné dans la section 7 avant de conclure dans la section 8.

---

1. *A*telier de *DE*veloppement de *L*ogiciels à *F*onctionnalité *E*mergente

2. Un modèle d'agent pour l'auto-organisation coopérative

2.1. *Les systèmes multi-agents coopératifs*

L'approche par systèmes multi-agents coopératifs propose un processus d'auto-organisation et se focalise sur la coopération des comportements locaux des entités (les agents) composant le système, afin de garantir un comportement collectif cohérent (Georgé *et al.,* 2003). Cette approche a pour but de construire des systèmes artificiels atteignant une adéquation fonctionnelle dans leur environnement alors que les agents ne cherchent qu'à atteindre des objectifs individuels, dans le sens où les agents n'ont pas de vision du système global. Pour obtenir une telle adéquation, dans le cadre du théorème de l'adéquation fonctionnelle (Camps *et al.,* 1998), il a été démontré que tout agent autonome, suivant un cycle "perception-décision-action" doit garder des relations aussi coopératives[2] que possible avec son environnement social (les autres agents) ou physique.

La définition de la coopération abordée ici n'est pas classique – simple partage de ressources, travail en commun, etc. – et prend ses sources dans des interactions naturelles comme la symbiose. Elle repose sur trois métarègles instanciables : (c_{per}) tout signal perçu par une agent doit être compris par lui, (c_{dec}) l'information provenant de ses perceptions doit être utile à son raisonnement et (c_{act}) son raisonnement doit le mener à des actions utiles aux autres et à l'environnement. L'approche proposée par les auteurs est une vision proscriptive du comportement : le comportement prioritaire des agents est d'éviter ou de réparer des situations non coopératives (ou *SNC*). De telles situations surviennent lorsqu'au moins une des trois conditions de la coopération n'est pas remplie et détectée localement par un agent. C'est alors un moment de réorganisation pour le collectif. L'algorithme relatif à un agent consiste donc à (i) réaliser sa fonction locale lorsqu'il est en état coopératif ou (ii) agir pour revenir à un état coopératif lorsqu'il perçoit une SNC.

Cette approche a de fortes implications méthodologiques : concevoir un système réside principalement dans la définition et l'affectation des règles de coopération aux agents. Le concepteur de tels systèmes doit, dans un premier temps, identifier les agents du système (les entités devant faire face à des situations non coopératives) puis déterminer la coopération idéale de leur point de vue ainsi que leur comportement coopératif. Cette dernière étape consiste à trouver les SNC auxquelles les agents peuvent être confrontés durant leur fonctionnement et définir les actions qu'ils doivent alors effectuer pour revenir à un état coopératif. Ceci revient donc à trouver le jeu de règles minimal (*ex :* les articles du code de la route) permettant au système de fournir une fonction souhaitée (*ex :* que l'ensemble des conducteurs n'ait pas d'accident, et donc que la circulation automobile fonctionne). Ces règles sont des instances des métarègles précédemment présentées.

---

2. Cette classification se retrouve en partie chez (Ferber, 1995) dans les catégories d'interaction qu'il définit.

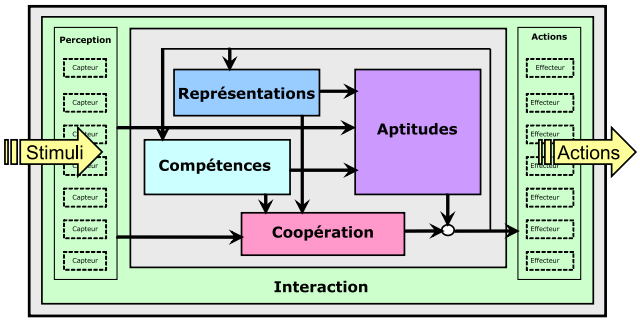

Figure 1. *Les différents modules d'un agent coopératif et leurs dépendances*

2.2. *Les agents coopératifs*

Ce paragraphe expose le modèle d'agent utilisant la coopération comme moteur de l'auto-organisation du système multi-agent afin d'atteindre l'adéquation fonctionnelle (Picard, 2004). Les agents coopératifs sont composés de différents modules représentant une partition de leurs capacités physiques, cognitives ou sociales (voir figure 1). Chaque module représente une ressource spécifique pour l'agent, et est utilisée lors de son cycle de vie composé des trois phases "perception-décision-action".

2.2.1. *Les différents modules*

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

Le *module de compétences* concerne les tâches des agents. Même si les agents coopératifs essaient principalement d'éviter les situations non coopératives, ils ont une ou plusieurs tâches à remplir en temps normal. Les compétences représentent un champ de connaissances permettant à l'agent de réaliser sa fonction partielle – en tant que partie d'un système réalisant une fonction globale. Aucune contrainte technique n'est requise pour concevoir et développer ce module. Par exemple, les compétences peuvent être implémentées comme une base de faits et de règles sur un domaine particulier. Dans ce module, nous pouvons aussi intégrer les caractéristiques intrinsèques aux agents, comme par exemple, leur poids, leur couleur, etc.

Comme pour les compétences, le *module de représentations* peut être implémenté sous forme d'une base de faits et de règles, mais il porte sur la connaissance locale et subjective de l'environnement (physique ou social) et sur l'agent lui-même. Les croyances sur les autres agents et sur lui-même sont considérées comme des représentations. Les représentations peuvent être décomposées en systèmes multi-agents de plus bas niveau pour obtenir des représentations dynamiques par le biais de l'auto-organisation (Gleizes *et al.,* 2000).

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

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

2.2.2. *Fonctionnement interne*

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

2.3. *Applications de l'approche par agents coopératifs*

L'approche par AMAS (*Adaptive Multi-Agent Systems*), à travers la méthode ADELFE, a été appliquée à divers problèmes provenant de domaines variés. Compte tenu de l'état courant de la méthode, ce sont principalement les travaux d'analyse et de conception qui ont été mis en œuvre dans les domaines suivants :

Robotique collective – afin de concevoir un collectif de robots devant transporter des ressources dans un environnement contraint et afin d'étudier l'émergence de sens

de circulation, des robots ont été munis de règles simples de réorganisation coopérative (Picard *et al.,* 2005b). Bien qu'il n'y ait pas de communication directe entre agents, le collectif est capable de trouver une organisation leur permettant d'atteindre leur but de manière adaptative, même lorsque l'environnement est dynamique et difficile ;

Conception de mécanismes aéronautiques dans le projet SYNAMEC – assistée par un AMAS dans lequel les composantes d'un mécanisme vont s'auto-organiser dans l'espace pour respecter la fonctionnalité (une trajectoire, par exemple) et les contraintes posées par le concepteur (Capera *et al.,* 2004, Capera *et al.,* 2005) ;

Gestion adaptative d'emploi du temps – dans laquelle des agents coopératifs, représentant des enseignants et des étudiants vont coopérer afin de trouver une organisation – c-à-d. un emploi du temps – satisfaisant au mieux les contraintes de chacun des participants. Cette application est notamment l'exemple donnée dans ADELFE pour illustrer chacun des travaux et étapes du processus (Picard, 2004, Picard *et al.,* 2005a) ;

Apprentissage comportemental des gènes dans le projet *MicroMega* – dans lequel, à partir des données observables, le système artificiel doit parvenir à prédire l'expression génomique de la levure cible (*Saccharomyces Cerevisiæ*) en déterminant l'organisation individuelle et collective de ses constituants afin de tendre vers les données observables (Macchion, 2004) ;

Conformation moléculaire dans le projet *Bio-S* – afin d'apprendre les interactions atomiques puis de les composer, de manière simple, locale et efficace. Dans un premier temps, l'objectif de ce travail a été de résoudre des problèmes de conformation de protéines par minimisation de l'énergie de Lennard-Jones induite des interactions de Van der Waals. Dans l'étape suivante, la capacité à minimiser l'énergie potentielle d'une molécule a été utilisée afin d'apprendre les fonctions régissant les interactions entre atomes (Besse, 2005).

Toutes ces applications ont abouti à des implémentations différentes du modèle d'agent. Ceci nous pousse alors à définir – ce qui est le but de cet article – des règles de transformations permettant le passage de la conception à l'implémentation, dans le processus ADELFE, pour une meilleure lisibilité et modularité de l'approche, mais aussi pour un meilleur respect de l'utilisation du modèle d'agent coopératif et de ses différents modules.

3. Des méthodes orientées agent/multi-agent

Depuis quelques années les méthodes de développement de systèmes à agents (pas forcément multi-agents) se sont multipliées. (Arlabosse *et al.,* 2004) et (Picard, 2004) en dressent des comparaisons assez complètes. Ne seront citées ici que certaines méthodes parmi les nombreuses existantes. Pour une information plus complète sur les méthodes orientées agent, nous aiguillons le lecteur vers les deux livres récents suivants : (Bergenti *et al.,* 2004, Henderson-Sellers *et al.,* 2005).

3.1. *Analyse de l'existant*

3.1.1. *Analyse par sources d'inspiration*

Nous pouvons analyser l'ensemble des méthodes existantes suivant plusieurs schémas. Tout d'abord, nous pouvons distinguer les méthodes suivant leurs sources d'inspiration :

– les *approches par mimétisme de phénomènes naturels ou sociaux* : compte tenu des problématiques soulevées par les systèmes multi-agents, les sciences naturelles ont apporté leurs connaissances et modèles. Ainsi, un axe de l'ingénierie multi-agent se focalise sur des modèles issus des sciences naturelles ou sociales. Ici, la spécification consiste plus en un effort de correspondance entre les entités du système à définir et les modèles prédéfinis, aux paramètres et constantes près (comme la vitesse d'évaporation de phéromones dans le cas des fourmis) (Van Dyke Parunak, 1997) ;

– l'*ingénierie des connaissances* : plusieurs méthodes pour le développement des systèmes multi-agents ont été proposées en partant de celles qui ont été dédiées à la modélisation des systèmes à base de connaissances et des systèmes experts. Par exemple, la méthode MAS-CommonKADS est une extension de CommonKADS qui ajoute des techniques venant des méthodes orientées objet à des techniques de conception des protocoles dans le but de modéliser les agents et les interactions entre agents (Iglesias *et al.,* 2005) ;

– l'*ingénierie des besoins* : cette approche fournit des méthodes, des techniques, et des outils permettant de développer et d'implanter des systèmes informatiques fournissant les services et les informations attendus par leurs utilisateurs, exigés par leurs acquéreurs, et qui soient compatibles avec leur environnement de fonctionnement. Bien que cette problématique soit non directement connectée à la notion de système multi-agent, certains travaux se focalisant sur des agents logiciels communicants fournisseurs de services pour des utilisateurs humains se sont fortement inspirés de cette branche de l'ingénierie des systèmes. Par exemple, la notation $i^\star$ de Yu se focalise sur l'ingénierie des besoins centrée sur les caractéristiques intentionnelles des agents (leurs buts). Cette notation a notamment été adoptée par la méthode Tropos (Giorgini *et al.,* 2005) ;

– l'*ingénierie des objets* : cette dernière est sans nul doute la source principale d'inspiration des méthodes orientées agent. En effet, cette approche est déjà bien connue des ingénieurs et des industriels – autant en ce qui concerne les notations, que les outils. Nous pouvons citer, entre autre, les méthodes PASSI et MESSAGE qui proposent des extensions de notations et de processus issus du domaine objet, ce qui a pour conséquence une meilleure intégration dans des outils de conception existants (Cossentino, 2005, Garijo *et al.,* 2005).

3.1.2. *Analyse par concepts/notations/processus/pragmatique*

Un autre schéma d'analyse des méthodes a été proposé par (Shehory *et al.,* 2001) qui considèrent quatre axes de comparaison : les concepts, les notations, le processus et la pragmatique. La partie sur les *concepts* s'interroge sur la capacité pour une mé-

thode à prendre en compte des notions généralement admises comme étant propres aux agents : autonomie, buts, croyances, etc. C'est principalement là que les méthodes se différencient. Sauf cas exceptionnel, les auteurs des différentes méthodes s'accordent à dire que leur approche n'est pas générale, mais est plutôt spécifique à une problématique, un modèle d'agent ou un contexte particulier. La partie sur les *notations* s'intéresse aux différents langages, formels ou semi-formels, utilisés pour la spécification des agents et de leurs composantes. Bien sûr, comme la plupart des méthodes considèrent les agents comme des objets, elles reposent sur l'utilisation plus ou moins étendue d'UML. La partie concernant le *processus* s'interroge sur les phases de développement qui sont couvertes par les méthodes. Dans l'état présent du domaine, les deux phases pleinement abordées sont l'analyse ("*quels sont les agents du systèmes ?*") et la conception ("*comment décrire les agents du système ?*"). Le constat est donc plutôt négatif, même si quelques méthodes proposent des outils spécifiques au codage de leurs agents, comme PASSI, par exemple (Cossentino, 2005). Enfin, le dernier critère de comparaison et d'évaluation des méthodes est la *pragmatique*, c-à-d. la facilité d'intégration et d'utilisation des méthodes. Depuis deux ans environ, ce qui fût un des points négatifs de ces méthodes est devenu une des activités massives du domaine. Publications de livres, mise en place de sites, téléchargements libres des outils sont de plus en plus communs et sont donc un moyen de pallier le manque de visibilité des méthodes orientées agent.

3.1.3. *Enseignements*

Le principal bilan à faire à l'issu de l'analyse des méthodes existantes peut s'exprimer en deux points :

– les méthodes qui proposent le plus d'outils et couvrent la plus grande partie du cycle de vie du logiciel sont celles issues du monde de l'objet. Ceci est très fortement dû à l'avantage de reposer sur des concepts déjà connus de la plupart des développeurs ;

– malheureusement, peu d'entre elles, voire aucune, ne proposent de méthode générique, automatique ou semi-automatique, d'implémentation ; elles restent au niveau de l'analyse et de la conception. Ceci est en grande partie dû au manque de standardisation du domaine agent – bien que certains axes soient de mieux en mieux balisés, comme les communications avec AUML (Huget *et al.,* 2004).

3.2. *Utilisation du paradigme objet*

Compte tenu du constat établi dans la section précédente, le projet ADELFE a pour but de fournir une méthode, se basant sur les technologies des objets, et proposant des outils d'aide au développement de systèmes multi-agents adaptatifs.

De nombreuses approches considèrent l'agent comme une extension de l'objet. (Van Dyke Parunak, 1997) et (Odell, 2002) expliquent l'évolution des approches de programmation, de la programmation monolithique à la programmation par agent.

Dans la programmation monolithique, les unités de logiciel représentaient le programme dans sa globalité, entièrement spécifié par le programmeur et contrôlé par le système. La programmation modulaire répondit à l'augmentation en terme de complexité, de besoins de mémoire des applications, ainsi qu'au besoin de réutilisabilité. Par contre l'état des unités (modules) restait toujours sous contrôle externe par instructions d'appel. L'évolution logique fut la décomposition des programmes en objets. Ici, l'état des objets est encapsulé grâce à des méthodes d'accès ou d'instanciation. En programmation par objet classique, les objets sont considérés comme passifs et donc soumis au contrôle par envoi de messages et appel de méthodes. Enfin, afin de répondre aux besoins des agents, la programmation par agent, qui n'est pour l'instant qu'à l'état de concept, se propose de concevoir les agents comme des "objets actifs avec initiative" – un peu comme les acteurs en conception objet – qui ont leur propre "thread" de contrôle et sont mus par des règles, des buts ou des intentions. Cependant, des approches récentes, provenant directement de l'ingénierie des objets sont très proches des besoins des l'ingénierie des agents. Par exemple, la librairie Java ProActive pour le développement de grilles repose sur un schéma d'objets actifs et mobiles rigoureux (Baude *et al.,* 2000) utilisant le modèle de composants Fractal (Baude *et al.,* 2000, David, 2005). Un autre modèle de composants, ACEEL, propose, via le schéma de conception *Strategy*, des capacités d'auto-adaptation des objets (Chefrour *et al.,* 2003). Malgré cela, ces approches restent loin des problématiques décisionnelles attachées aux agents, même si ACEEL propose un protocole d'adaptation nécessitant une capacité de décision des stratégies à mettre en oeuvre.

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

Les interactions entre agents comparées aux interactions objet à objet sont plus riches sémantiquement et ne sont pas uniquement des demandes d'exécution. Les agents peuvent bien sûr faire appel à des méthodes au sens objet du terme, mais y ajoutent un contenu plus riche, comme par exemple avec les actes de langage FIPA[3] ou KQML[4] – il n'est pas rare d'ailleurs de trouver des librairies dédiées à ces actes dans les plates-formes de développement de systèmes multi-agents. Alors que les objets voient les autres objets comme des fournisseurs de services par appel de méthodes, les notions de conversations, de communications à long terme et de réseaux de partenaires sont aussi des notions importantes en programmation par agent. En effet, les agents peuvent construire des réseaux de contacts et donc posséder des capacités cognitives correspondantes comme une mémoire ou des croyances sur les autres agents.

3. http://www.fipa.org/repository/aclspecs.html
4. http://www.cs.umbc.edu/kqml/

En définitive, les agents peuvent être vus comme des objets ayant des règles structurelles ou comportementales supplémentaires – définies par des stéréotypes ou des extensions du métamodèle UML. La plupart des concepts manipulés par la communauté agent peut être modélisée en objet (ontologie, dynamique d'états, ...) mais certains axes sont encore peu explorés. Il semble aussi que la caractéristique d'autonomie (qu'elle soit dynamique ou non-déterministe) d'un agent reste le point difficile de la conception d'agent – comme cela l'est toujours dans le domaine de l'intelligence artificielle. Le problème est de trouver le bon niveau d'abstraction permettant de qualifier un agent d'autonome ou non. En effet, un agent reste toujours dépendant du système d'exploitation ou de la plate-forme sur laquelle il s'exécute, mais pas vis-à-vis des autres agents ou des objets qui l'entourent.

Ce rapprochement entre les concepts d'objet et d'agent a bien entendu l'avantage de permettre de réutiliser/étendre des *modèles et notations*, comme UML, mais aussi des *processus*, comme le RUP (*Rational Unified Process*) dans ADELFE (voir section 4) et MESSAGE, par exemple.

4. Le processus de la méthode ADELFE

La méthode ADELFE, dédiée au développement de systèmes multi-agents adaptatifs, est décrite suivant trois axes fondamentaux : le processus, les notations et les outils. Dans cet article, nous discutons principalement les deux premiers points (sections 4 et 5) qui sont issus des technologies des objets. Pour plus d'information concernant les outils, nous invitons le lecteur à aller consulter le site du projet ADELFE, où ils sont téléchargeables librement : `www.irit.fr/ADELFE`.

4.1. *Survol du processus*

Proposer une méthode de développement relève principalement d'une démarche à suivre pour atteindre des objectifs donnés. Le processus fourni doit permettre à tout concepteur d'effectuer un suivi pas-à-pas des étapes de développement de logiciels et des méthodes mises en jeu. Définir un processus consiste non seulement en la définition de ses étapes (ou phases), mais aussi en leur ordonnancement et en l'expression des modèles et artefacts produits et nécessaires. Le processus d'ADELFE ne déroge pas à la règle, comme l'illustre la figure 2 (Picard, 2004). Il se divise en cinq *définitions de travaux* : les besoins préliminaires, les besoins finals, l'analyse, la conception et le codage. Ces travaux s'intègrent dans le RUP et ont été étendus afin de répondre au métier agent (Jacobson *et al.,* 2000). Chaque ensemble de travaux ainsi que ses artefacts sont définis en utilisant la notation issue de SPEM (pour *Software Process Engineering Metamodel*) permettant de définir des processus de manière simple et compréhensible pour les utilisateurs d'UML (OMG, 2005b).

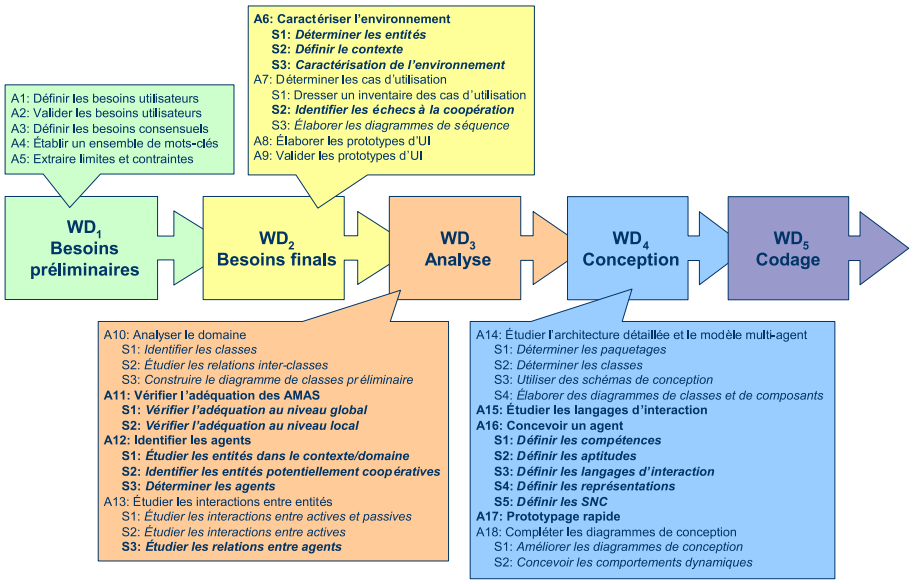

Figure 2. *Les cinq premières définitions de travaux du processus d'ADELFE*

4.2. *Spécificités*

ADELFE est spécifique à la conception d'AMAS ; aussi, certaines activités (marquées en gras dans la figure 2) ont été ajoutées au RUP pour l'adapter à cette technologie.

Durant l'expression des besoins finals (WD_2) :

A_6 : *Caractérisation de l'environnement*. En effet, il semble nécessaire de se concentrer, dès les premières analyses des interactions entre le système et son environnement, sur la nature de ces interactions car ce sont elles qui guideront l'auto-organisation du système et qui influeront sur la définition des règles de coopération ;

A_7-S_2 : *Identification des échecs à la coopération*. Cette étape est une suite logique du point précédant. Une fois que les propriétés de l'environnement ont été identifiées, il faut définir quelles sont les interactions entre acteurs (environnement) et cas d'utilisations (système) par lesquelles les problèmes d'inadéquation entre système et environnement passeront.

Durant l'analyse (WD_3) :

A_{11} : *Vérification de l'adéquation aux AMAS*. Certes, nous proposons de développer des systèmes dont le fonctionnement adaptatif repose sur l'auto-organisation par coopération, mais tous les problèmes n'ont pas forcément besoin d'être résolus par cette approche. De plus, même si l'utilisation de tels systèmes semble

convenir, l'analyse ne sait pas forcément où placer des agents coopératifs, ou bien quelles sont les entités du domaine qui devront/pourront être décomposées en agents ;

A_{12} : *Identification des agents impliqués dans le système à construire*. Les agents ne sont par forcément évidents à identifier dans des systèmes non orientés utilisateurs. En effet, dans des systèmes de courtage en ligne, par exemple, les agents sont directement associés aux utilisateurs. Mais dans un problème comme la prévision de crue, la présence d'agents horaires, devant produire des prévisions pour une heure, n'est pas une identification directement extractible de l'analyse des entités du domaine. Il nous a donc semblé nécessaire d'ajouter une telle activité d'identification des agents à partir de l'analyse des interactions inter et extra système ;

A_{13}-S_3 : *Étude des relations entre ces agents*. Comme pour les objets, les interactions entre agents doivent être étudiées, avec les diagrammes de séquences ou de collaboration. Cependant, les agents peuvent faire preuve de plus de richesse dans leurs protocoles de communication, ce qui nécessite l'ajout d'une activité spécifique devant produire des modèles d'interaction entre agents (protocoles AUML).

Durant la conception (WD_4) :

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

A_{16} : *Conception complète de ces agents*. Un agent qui intervient dans un AMAS est composé de différentes parties qui forment son comportement : des compétences, des aptitudes, un langage d'interaction, des représentations du monde et des Situations Non Coopératives (SNC) (voir section 2). L'ajout de cette activité est essentiel à la conception de systèmes multi-agents adaptatifs, puisque elle consiste à attribuer un comportement nominal, pour résoudre une tâche courante, et un comportement coopératif aux agents afin de résoudre le problème voulu. C'est donc dans cette activité que les règles d'auto-organisation coopérative vont être définies et attribuées aux agents ;

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

La description complète des activités s'arrête actuellement à la fin de la conception. Dans cet article, nous étudions les ponts possibles entre la conception et le codage, en

prenant pour acquis que la programmation des systèmes obtenus sera effectuée en programmation par objets.

5. Notations et stéréotypage

Les notations sont une dimension importante dans la définition de méthodes. Compte tenu des remarques et conclusions faites lors de l'analyse des méthodes existantes, nous avons fait le choix de la notation UML. Cependant nous sommes conscients des limites d'UML, tant au niveau de l'expressivité que de la validation. En effet, les concepts UML ne sont pas suffisants pour exprimer les propriétés des systèmes multi-agents. De plus, UML propose des solutions de vérification syntaxique mais pas sémantique. Nous allons donc proposer des extensions à UML, en nous appuyant, notamment sur AUML (Huget *et al.,* 2004), qui propose déjà des notations spécifiques pour les interactions entre agents. Les extensions que nous proposons portent principalement sur deux aspects de la conception des agents :

– La *vue statique* des agents, par la définition de stéréotypes d'attributs et d'opérations, permettant un enrichissement sémantique et l'intégration des modules des agents coopératifs ;

– La *vue dynamique* ou comportement des agents par l'utilisation et l'extension de AUML, ainsi que la transformation de protocoles en machines à états, afin de réaliser des vérifications et de simuler le comportement social des agents communicants.

5.1. *Modélisation statique*

La première extension apportée à UML par ADELFE est le stéréotypage d'opérations et d'attributs en fonction de leur appartenance aux modules définis pour le modèle d'agent coopératif fourni par ADELFE (voir section 2). Tous ces stéréotypes s'appliquent à des traits (de classe Feature du métamodèle UML), c-à-d. à des attributs ou des opérations, excepté le stéréotype «cooperative agent» qui s'applique à des classes. Les stéréotypes forment une extension du métamodèle UML afin de contraindre l'utilisation du modèle d'agent coopératif. Chaque stéréotype correspond à une sémantique particulière, mais surtout à des contraintes de visibilité entre modules. Ceci est résumé dans la figure 3 et détaillé dans les paragraphes suivants.

5.1.1. *Le stéréotype de classe «cooperative agent»*

Le stéréotype «cooperative agent» exprime le fait qu'une classe est un agent possédant une attitude coopérative participant à la construction d'un AMAS. Un agent sera donc modélisé comme une classe stéréotypée. Cette classe devra posséder une opération run() simulant le cycle de vie de l'agent. Par conséquent, pour assurer que cette opération existe, un agent doit hériter d'une superclasse abstraite appelée CooperativeAgent, fournie par ADELFE. Cette classe possède les opérations abstraites perceive(), decide() et act() et une opération run(). Ceci implique que les agents seront

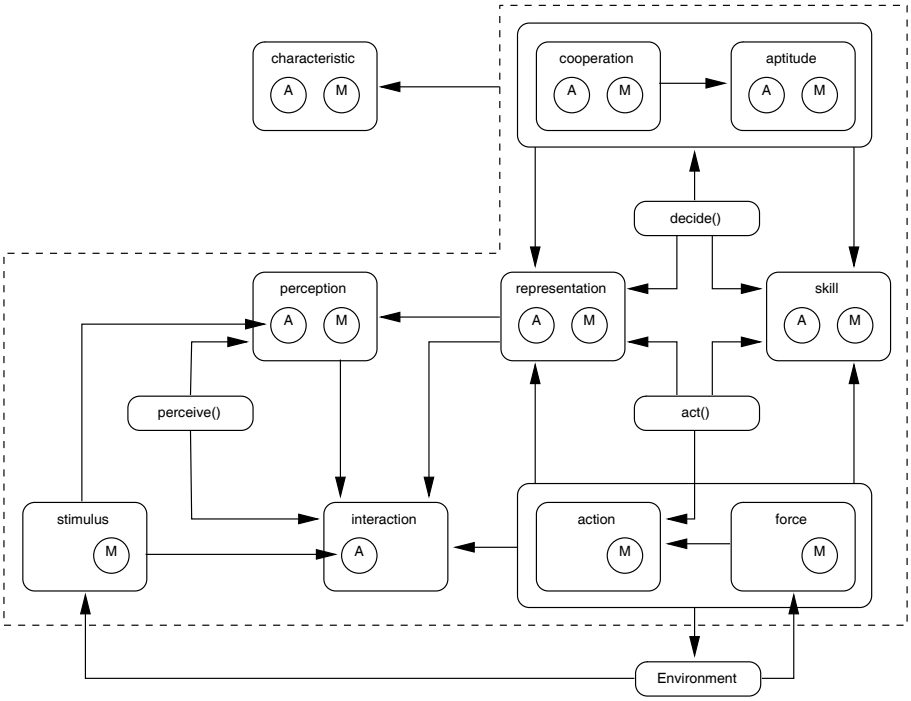

Figure 3. *Constituants d'un agent coopératif et visibilités associées (A pour attributs, M pour méthodes)*

animés par l'opération run(), que ce soit par un ordonnanceur préétabli (dans le cas de *threads*, par exemple) ou développé pour l'occasion. Ainsi, à chaque pas d'exécution, l'ordonnanceur appellera l'opération run() de chaque agent dans un ordre préétabli ou aléatoire (dans le cas de *threads* par exemple). Cependant, c'est au développeur de remplir les opérations perceive(), decide() et act() de la classe d'agent, héritant de CooperativeAgent, qu'il code. Une classe stéréotypée «cooperative agent» doit hériter directement ou indirectement de la classe CooperativeAgent. De plus, afin de modéliser un agent coopératif "viable", une classe «cooperative agent» devra nécessairement posséder au moins des opérations ou attributs stéréotypés de chacun des stéréotypes suivants :

– «perception», car un agent doit forcément percevoir son environnement ;

– «action», car un agent doit agir s'il ne veut pas être inutile et donc non coopératif ;

– «aptitude», car un agent est une entité autonome dans ses décisions et raisonnements ;

– «cooperation», car un agent coopératif doit forcément avoir une attitude sociale coopérative.

5.1.2. *Les stéréotypes d'attributs et de méthodes*

Le stéréotype «characteristic» est utilisé pour pointer les propriétés intrinsèques ou physiques d'un agent coopératif. Un attribut ainsi stéréotypé représente la valeur de cette propriété. Une opération «characteristic» modifie ou met à jour ces propriétés. Une caractéristique peut être lue ou modifiée à n'importe quel moment du cycle de vie de l'agent. Elle peut aussi être lue ou appelée par les autres agents.

Le stéréotype «perception» exprime un moyen que l'agent a pour traiter et décoder de l'information venant de son environnement physique ou social. Les attributs représentent les données provenant de l'environnement ou décodées. Les opérations sont des moyens de mettre à jour ou de modifier les attributs «perception». Une opération ou un attribut stéréotypé «perception» est nécessairement privé ou protégé. Un attribut «perception» peut être manipulé par l'opération perceive(), des opérations «stimulus» ou des opérations «representation». Une opération «perception» peut être appelée par l'opération perceive() ou des opérations «representation».

Le stéréotype «stimulus» exprime un moyen que l'agent a de recevoir de l'information venant de son environnement physique ou social (ou d'être prévenu de la disponibilité d'informations dans son environnement). Seules des opérations peuvent être stéréotypées «stimulus». Une opération stéréotypée «stimulus» peut être appelée à n'importe quel moment du cycle de vie de l'agent. Elle peut aussi être appelée par les autres agents. Enfin, une telle opération ne peut accéder qu'à des attributs stéréotypés «perception» ou «interaction».

Le stéréotype «action» est utilisé pour signaler un moyen d'agir sur l'environnement durant la phase d'action. Il ne peut être utilisé que pour des opérations qui sont alors les actions possibles pour un agent. Un agent est le seul à pouvoir activer ses propres actions afin d'assurer l'autonomie de contrôle. Une opération stéréotypée «action» est nécessairement privée et ne peut être appelée que lors de la phase d'action. Ceci signifie que les opérations stéréotypées «action» ne peuvent être appelées (directement ou indirectement) que par l'opération act(). Enfin, une telle opération ne peut accéder qu'à l'environnement, des attributs stéréotypés «interaction», des traits stéréotypés «representation» ou des traits stéréotypés «skill».

Le stéréotype «force» est utilisé pour modéliser des actions que l'environnement social ou physique peut réaliser sur l'agent sans que ce dernier ne puisse intervenir. Il ne peut être utilisé que pour des opérations qui sont alors accessibles uniquement à l'environnement. Il convient d'utiliser ce stéréotype avec parcimonie puisqu'il peut permettre de briser l'autonomie de contrôle. Il n'est disponible que pour modéliser des phénomènes d'ordre physique, par exemple un robot ne peut pas aller à l'encontre du déplacement engendré par un tremblement de terre. Enfin, une opération stéréotypée «force» ne peut accéder qu'à l'environnement, des opérations stéréotypées «action», des attributs stéréotypés «interaction», des traits stéréotypés «representation» ou des traits stéréotypés «skill».

Le stéréotype «interaction» étiquette les attributs qui permettent à l'agent de communiquer ou d'agir sur son environnement. Ils peuvent donc être utilisés depuis une opération étiquetée «perception», «stimulus», «action», «force» ou «representation».

Le stéréotype «skill» est utilisé pour étiqueter les compétences, c-à-d. des connaissances spécifiques à un domaine, permettant à l'agent de réaliser sa fonction partielle. Les opérations représentent les règles de raisonnement que peuvent avoir les agents. Les attributs sont des données (ou faits) sur le monde ou les paramètres des opérations stéréotypées «skill». De tels attributs ou opérations ne sont accessibles qu'à l'agent les possédant pour exprimer son autonomie de décision. Les compétences peuvent être représentées par un système multi-agent adaptatif si les compétences ont besoin de changer au cours du temps. Une telle décomposition doit être détectée lors de l'analyse du système (voir activité A_{11}). Le système multi-agent résultant sera alors un attribut stéréotypé «skill» de la classe d'agent décomposée. Un attribut ou opération stéréotypé «skill» est nécessairement privé. De plus, de tels traits ne peuvent être uniquement appelés lors des phases de décision ou d'action de l'agent (seules les opérations decide() et act() peuvent l'appeler de manière directe ou indirecte).

Le stéréotype «aptitude» exprime la capacité d'un agent à raisonner sur ses perceptions, ses connaissances et ses compétences. Les opérations expriment le raisonnement qu'un agent est capable de faire, par exemple de l'inférence sur ses compétences (qui peuvent alors être des règles et des faits). Les attributs représentent les données de fonctionnement ou les paramètres du raisonnement. Une opération ou attribut stéréotypé «aptitude» peut seulement être accessible à l'agent lui-même, pour exprimer son autonomie de décision. Une opération ou attribut stéréotypé «aptitude» est nécessairement privé ou protégé. Un attribut «aptitude» peut seulement être utilisé par une opération «aptitude». Une opération «aptitude» peut uniquement être appelée lors de la phase décision du cycle de vie de l'agent (opération decide()). Une opération «aptitude» peut uniquement faire appel à des attributs ou opérations stéréotypés «characteristic», «skill», «representation». Cette dernière règle met en place le fonctionnement interne de l'agent, comme exprimé dans le paragraphe 2.

Le stéréotype «representation» est un moyen d'indiquer les représentations du monde que possède l'agent. Les attributs «representation» sont des unités de connaissances. Les opérations «representation» sont des moyens de les manipuler : accès, modification, etc. Les représentations peuvent évoluer, et, de ce fait, être modélisées par un AMAS. Une opération (ou attribut) «representation» est nécessairement privée ou protégée. En effet, la connaissance d'un agent est locale et personnelle, à moins qu'il ne la communique via des messages, par exemple. Un trait «representation» peut seulement être accédé lors des phases de décision ou d'action (opérations decide() et act()). Enfin une opération «representation» peut faire appel à des traits «perception» et «interaction».

Le stéréotype «cooperation» exprime l'attitude sociale coopérative de l'agent en implantant les règles de résolution des situations non coopératives. L'agent doit avoir un ensemble de règles (ou prédicats) permettant de détecter les SNC. Ces règles sont écrites en utilisant les perceptions, les représentations et les compétences. L'agent doit

aussi posséder des opérations de résolution associées aux règles de détection. Une opération «cooperation» pourra être, par exemple :

– une opération renvoyant un booléen indiquant la détection d'une SNC et possédant des paramètres provenant de perceptions, de représentations et/ou de compétences ;

– une opération de résolution associant une ou plusieurs actions possibles à chaque SNC. Le choix, dans le cas d'actions multiples, devra aussi être géré par une opération «cooperation» ou «aptitude».

Ces opérations «cooperation» doivent subsumer les opérations «aptitude», c-à-d. que les actions choisies par les opérations «cooperation» prennent le pas sur les actions choisies en cas de fonctionnement nominal. Ceci peut être obtenu de plusieurs manières : instructions conditionnées ou usage d'exceptions pour simuler la priorité des résolutions de situations non coopératives. Une opération ou un attribut «cooperation» est nécessairement privé ou protégé et peut seulement être manipulé durant la phase de décision (opération decide()). Une opération «cooperation» peut accéder à des traits «aptitude», «representation», «skill», ou «characteristic»

5.2. *Modélisation dynamique*

Dans ADELFE, le comportement dynamique des agents est modélisé grâce à des machines à états finis ou des protocoles d'interaction dans le cas d'agents communicants. Nous avons dû étendre les notations AUML afin de prendre en compte la coopération des agents lors de communications. Nous disposons alors de deux formalismes de modélisation de la dynamique : les protocoles et les machines à états finis. Nous proposons un algorithme de transformation de protocoles en machine à états afin d'unifier les notations dans la section 6.3. L'usage des machines à états finis est classique, et ne sera donc pas détaillé dans cette section.

5.2.1. *Les protocoles AUML dans ADELFE*

Le diagramme de protocole développé pour ADELFE est une évolution du diagramme de séquences *générique* d'UML. Il permet de faire intervenir des classes et des rôles de classe d'agent, et de les faire communiquer entre eux. Il peut être attribué à n'importe quel objet, car il s'agit d'un diagramme générique.

Outre les lignes de vie avec les rectangles d'activation et les envois de message, il est possible de saisir des embranchements (ou jonctions) AND (barre verticale), OR (losange), et XOR (losange avec une croix), comme il est spécifié dans les propositions AUML, et de fournir certaines connaissances supplémentaires sur les envois de messages, comme le traitement coopératif à la réception d'un message. Il est également possible de spécifier des clauses IF...ELSE...ENDIF, permettant plus d'expressivité lors de la conception des protocoles, mais ce point ne sera pas détaillé dans cet article (Picard, 2004).

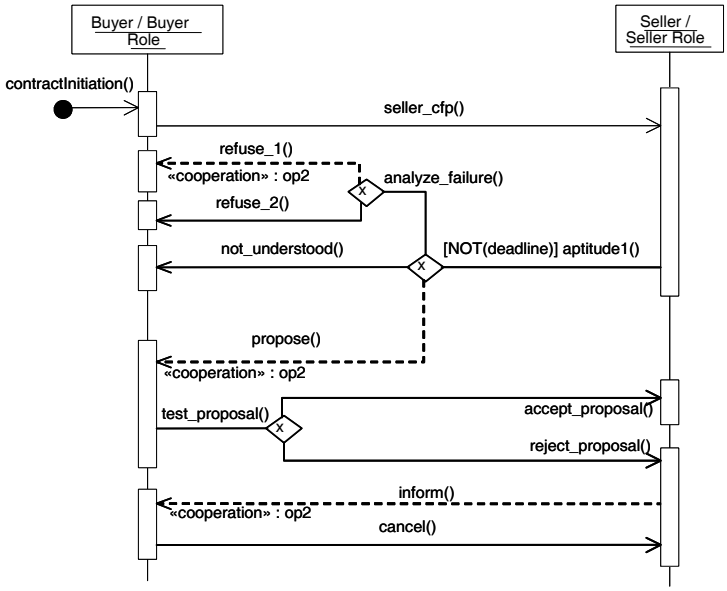

Figure 4. *Un exemple de protocole dans ADELFE*

Un exemple de diagramme de protocole est présenté en figure 4. Ce protocole est celui présenté dans les premières spécifications d'AUML (Odell *et al.,* 2000). Cependant, quelques différences existent : les flèches en pointillés signalent des interactions potentiellement non coopératives auxquelles sont associées des opérations de traitement «cooperation».

5.2.2. *Contraintes de spécification des protocoles*

Les diagrammes de protocoles doivent obéir à certaines contraintes :

– le temps graphique n'a de sens qu'au sein d'une activation, ceci en raison de l'introduction des jonctions AND, OR et XOR qui perturbent la disposition graphique ;

– le diagramme de protocoles doit commencer par un message initial ;

– un déroulement du protocole sous forme d'arbre doit être possible à partir du message initial ; toutefois, certaines exceptions sont possibles.

Les rectangles d'activation du protocole fournissent une trame de continuité au sein d'un enchaînement possible d'envois de messages : l'événement associé à un message est recherché dans l'activation réceptrice de ce message (message graphiquement situé immédiatement en dessous du départ de cette activation). Les embranchements sont dus à l'introduction des jonctions AND, OR ou XOR. Les cas suivants, qui présentent une rupture de continuité, sont cependant compris :

– envoi de plusieurs messages successifs à partir d'une classe ou d'un rôle d'agent ;

– réception de plusieurs messages successifs sur une classe ou un rôle d'agent.

5.2.3. *Utilisation des diagrammes de protocole*

Lors de la création d'une ligne de vie, il faut lui associer une classe, et un rôle d'agent s'il s'agit d'un agent qui en possède plusieurs. Il est important de créer au moins un rôle d'agent par classe d'agent susceptible d'en avoir plusieurs, pour pouvoir éventuellement en introduire d'autres par la suite, et d'utiliser le rôle d'agent pour créer une ligne de vie, et non l'agent lui-même.

Les envois de messages ne peuvent être créés que sur des rectangles d'activation et des jonctions AND, OR ou XOR. Il est possible de créer sur une ligne de vie autant de rectangles d'activation que l'on veut. Seules des opérations publiques stéréotypées «interaction» doivent être proposées sur la création d'un envoi de message. Il est possible de préciser une opération stéréotypée «cooperation» à la réception d'un envoi de message : ceci permet de renvoyer à d'autres comportements traités dans d'autres protocoles, par exemple.

Une jonction AND, OR ou XOR doit être rattachée à une classe ou à un rôle d'agent. Cependant, il faut faire attention aux règles qui sont différentes selon le type de jonction. Le message interne qui va d'un agent à un losange " OR " ou " XOR " doit porter une opération agent stéréotypée «aptitude» : l'indéterminisme des jonctions AUML représente l'autonomie de décision des agents.

6. De la conception à l'implémentation

6.1. *Positionnement MDA*

L'approche envisagée dans cet article pour prendre en charge le passage de la conception à l'implémentation dans le processus ADELFE, s'inscrit dans la proposition de MDA (Model Driven Architecture ou architecture basée sur les modèles) par l'OMG[5] (Kleppe *et al.,* 2003). Le MDA a pour but d'accélérer le développement d'applications en séparant les aspects spécifiques de l'application, des aspects liés à son implantation sur une architecture particulière. Il propose ainsi de résoudre les problèmes d'interopérabilité et de portabilité. Plus concrètement, l'approche MDA consiste à définir les fonctionnalités du système dans un modèle indépendant de la plate-forme (PIM : Platform Independant Model) puis, de le traduire dans un modèle spécifique à une plate-forme (PSM : Platform Specific Model), base pour la génération de code exécutable. L'objectif visé par l'OMG, et en partie atteint, consiste à rendre toutes ces transformations de modèles automatiques. Nous désirons, dans le cadre de notre travail, aller vers la manipulation d'un modèle d'agent de haut niveau, indépendant de la plateforme et permettant de générer au moins semi-automatiquement un ou plusieurs modèles spécifiques à une plateforme puis du code.

5. http://www.omg.org/mda

Dans cette optique, il nous paraît nécessaire de tenir compte du caractère agent du système dès le PIM. Certains pourront argumenter qu'il s'agit là d'une décision de conception conditionnant la plateforme utilisée pour le système. Il faut toutefois considérer que l'utilisation d'agents ou non est un choix de très haut niveau et indépendant d'une plateforme agent particulière. Nous notons aussi que la distinction entre PIM et PSM se fait finalement de manière relative (Kleppe *et al.,* 2003). Le type de modèle que nous considérons comme PIM dans la suite de cet article pourra donc être considéré comme un PSM pour quelqu'un voulant faire une modélisation de plus haut niveau.

Dans ce cadre, notre PIM sera un modèle dans lequel il est possible de distinguer trois parties. Premièrement, la partie décrivant les agents du système – qui tirera bien évidemment partie des notations discutées en 5.1, comme le montre la figure 5. Deuxièmement, les protocoles manipulés par les agents seront modélisés dans le formalisme décrit en 5.2, comme le précise la figure 11. Enfin, il convient de décrire les objets métiers manipulés par les agents et au sein des protocoles, pour cela une approche objet classique peut convenir.

Une fois la première transformation opérée, les modèles disponibles seront un PSM objet décrivant la structure des agents du système, un PSM constitué de machines à états décrivant l'exécution des protocoles, et un ou plusieurs PSM transposant les objets métiers du PIM. À partir de ces modèles il devient tout à fait possible de générer du code. Notons toutefois que la solution proposée est incomplète puisque nous ne couvrons pas toute la dynamique de nos agents. Les protocoles sont bien sûr une part importante du modèle, mais pour atteindre une génération complète du code nécessaire il faudrait s'attacher à modéliser la phase de décision des agents.

6.2. *Du modèle statique à la structure objet de l'agent*

Dans la section 5.1, nous avons présenté un modèle fonctionnel à visibilités restreintes. Il s'agit bien d'un modèle de haut niveau qui nécessite de progresser vers une solution implémentée et conforme au modèle initial. En ce sens, il s'agit d'un métamodèle relativement au PIM. Dans ce cas, les règles de transformation pour passer du PIM au PSM doivent garantir ses propriétés. Pour cela nous avons mis au point une procédure permettant de créer un modèle objet restreignant la visibilité des méthodes de l'agent conformément aux contraintes des stéréotypes ADELFE. Cette procédure s'inscrit donc dans une chaîne MDA, comme précisé dans la figure 5. Nous allons la détailler pour un agent donné (nous utiliserons l'agent décrit figure 6 comme exemple).

Tout d'abord pour chaque stéréotype «s1», nous insérons deux classes nommées S1Fields et S1MethodsBase (voir figure 7). Elles contiennent respectivement les attributs et les méthodes appartenant à «s1». Tous les traits ainsi créés sont d'accès public et les opérations sont abstraites.

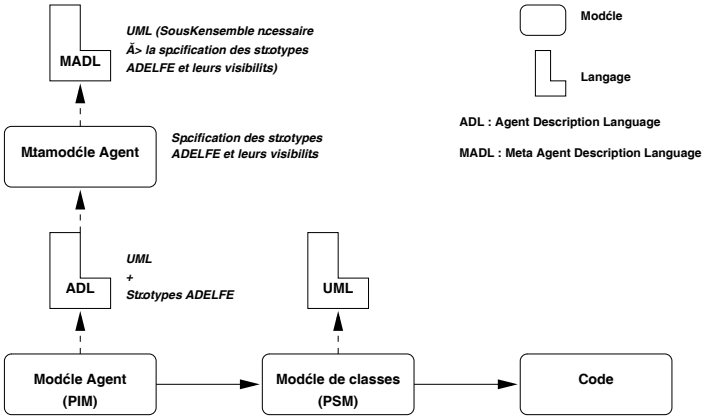

Figure 5. *Une chaîne de type MDA pour l'aspect statique d'un agent*

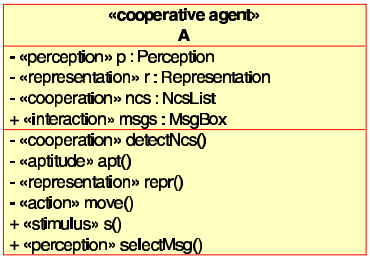

Figure 6. *Exemple de classe d'agent coopératif obtenue en fin de conception*

Ensuite, pour chaque stéréotype «s1» un deuxième traitement est opéré. Nous insérons une classe dont le nom est S1Namespace (voir figure 8). Dans cette classe nous insérons un attribut s1Fields de type S1Fields. Et, pour chaque stéréotype «s2» directement accessible (sans transitivité) depuis le «s1» on insère dans la classe S1Namespace un attribut s2Fields de type S2Fields, et un attribut s2Methods de type S2MethodsBase en fonction des accès autorisés dans le modèle. Il ne reste alors qu'à insérer dans la classe S1MethodBase un attribut protégé nommé self et

Figure 7. *Étape 1 de la génération de la structure : ajout des classes d'attributs et de méthodes pour un stéréotype*

de type S1Namespace. Ainsi les méthodes du stéréotype «s1» ont uniquement accès aux constituants de l'agent autorisés par le modèle par le biais de self.

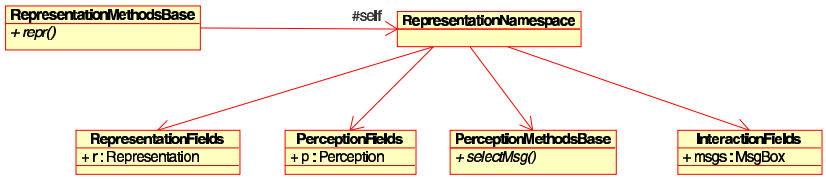

Figure 8. *Étape 2 de la génération de la structure : ajout de la classe d'espace de nom pour le stéréotype et de l'attribut self pour l'accès à ses méthodes*

Ensuite, une procédure similaire à la description ci-dessus est utilisée pour les phases du cycle de vie de l'agent (voir figure 9). Nous insérons donc pour chaque phase ph() une classe PhPhase (contenant une unique opération ph()) et une classe PhNamespace. De même la classe PhPhase dispose d'un attribut self de type PhNamespace.

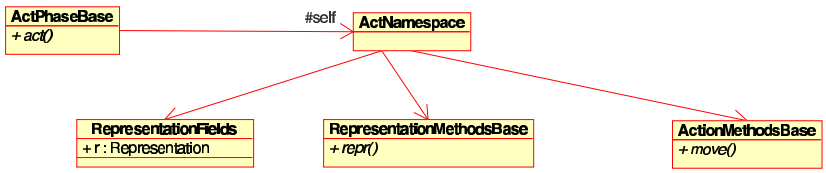

Figure 9. *Étape 3 de la génération de la structure : ajout de la classe d'espace de nom pour chaque phase de l'agent (ici, action) et de l'attribut self correspondant*

Enfin, nous insérons une classe AgentBase contenant pour chaque stéréotype «s1» un attribut s1Fields de type S1Fields et un attribut s1Methods de type S1MethodsBase, et, pour chaque phase ph() un attribut phPhase de type PhPhase (voir figure 10). Tous ces attributs sont privés. Ceci conclut le passage du PIM au PSM.

Lors du passage du PSM au code, il convient d'obtenir une initialisation correcte des champs de la classe AgentBase. En effet, elle doit disposer d'un constructeur qui va initialiser les différents champs suffixés Base (en général avec une sous-classe non abstraite pour garantir que les méthodes sont bien implémentées) ainsi que leurs champs self.

6.3. *Des protocoles aux machines à états finis*

Comme il a été précisé plus haut, afin d'unifier les notations et de faciliter la transformation de modèle de la conception vers l'implémentation, nous proposons de transformer les protocoles d'interaction en machines à états finis. Ce paragraphe expose la

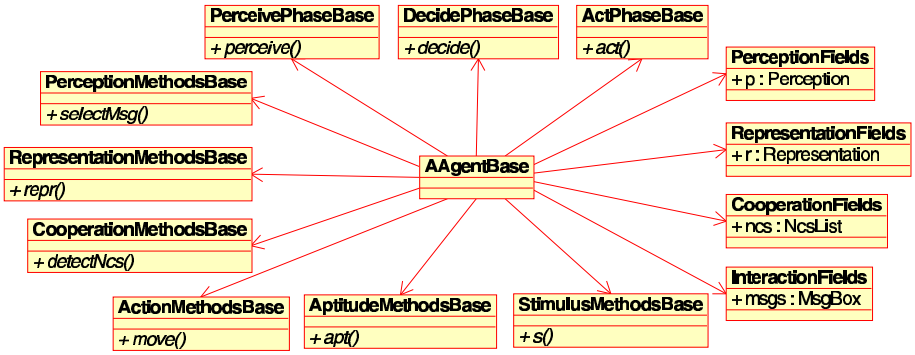

Figure 10. *Étape 4 de la génération de la structure : ajout de la classe de l'agent*

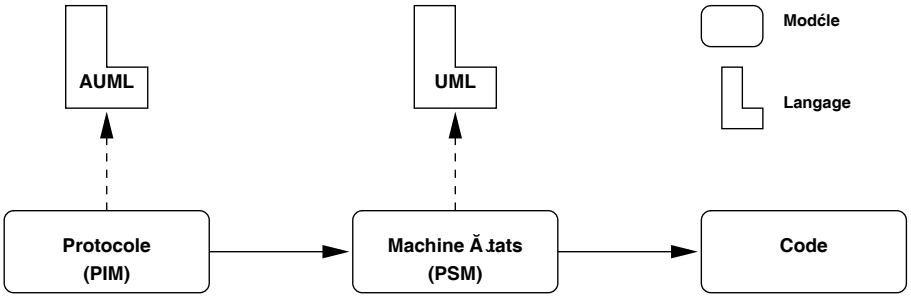

Figure 11. *Une chaîne de type MDA pour l'aspect dynamique d'un agent*

méthode à suivre et est illustré par la transformation du protocole de la figure 4. Cette approche s'inscrit elle aussi dans une chaîne MDA, comme précisé dans la figure 11.

6.3.1. *Construction de machines à états finis à régions concurrentes*

Tout d'abord, une machine à état initiale est attribuée à chaque classe «cooperative agent». Cette machine est composée d'un état initial et d'un état composite Alive. Dans ce dernier, une région concurrente est ajoutée pour chaque rôle tenu par l'agent, comme le montrent les figures 12 et 13. Ici, un seul rôle provient d'un protocole (région du haut). Bien sûr, d'autres régions concurrentes peuvent être ajoutées pour d'autres comportements dynamiques, par exemple la région du bas dans les figures présentées. Ensuite, chaque région concurrente est développée comme une machine à états. Ceci permet à l'agent de pouvoir répondre et participer à plusieurs protocoles à la fois.

Chacune de ces sous-machines est composée de deux états principaux : un état Idle, par défaut, et un état, ayant, par convention, le même nom que l'événement initiateur

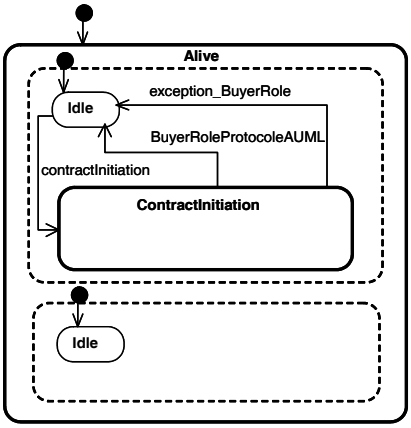

Figure 12. *Machine à états finis pour la classe* Buyer *du protocole de la figure 4*

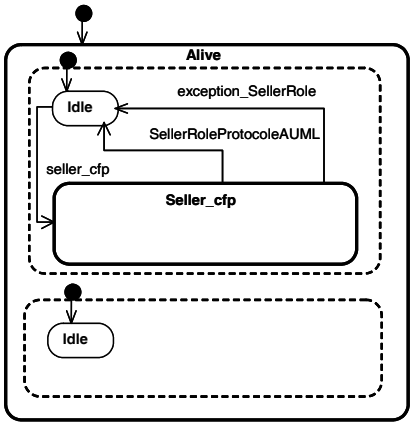

Figure 13. *Machine à états finis pour la classe* Seller *du protocole de la figure 4.*

du protocole pour le rôle avec la première lettre en majuscule. Dans le protocole, le rôle BuyerRole est activé à la réception de l'événement contractInitiation. Par conséquent, la région concurrente correspondant à ce rôle contient un état ContractInitiation. De même, le rôle SellerRole est activé à la réception de l'événement seller_cfp. Et donc, la région concurrente correspondant à ce rôle contient un état Seller_cfp. Ces deux états principaux sont reliés par des transitions bidirectionnelles : ainsi, un agent peut exécuter plusieurs fois un protocole. L'agent sort de l'état Idle lorsqu'il reçoit l'événement de début de protocole (transition Idle vers "état de rôle"). Pour sortir du protocole (transition "état de rôle" vers Idle), il y a deux possibilités : soit le protocole est arrivé à son terme pour ce rôle, soit une exception s'est produite suite à une SNC, par exemple.

En considérant la sémantique donnée aux lignes de vie des protocoles, un sous-état est ajouté à l'état composite du rôle pour chaque réception ou émission de message, comme le montre la figure 14. Par exemple, un sous-état Seller_cfp_sent est associé à l'envoi du message seller_cfp. De même, un état Refuse_1 est associé à la réception du message refuse_1.

Les transitions entre états sont créées à la réception ou à l'émission d'un message. Par exemple, quand l'agent Buyer se trouve dans l'état Seller_cfp_sent et reçoit le message refuse_1, une transition est créée entre les états Seller_cfp_sent et Refuse_1. Si un état ne reçoit ou n'envoie plus aucun message, il est relié à un état final avec l'action de retour à l'état Idle.

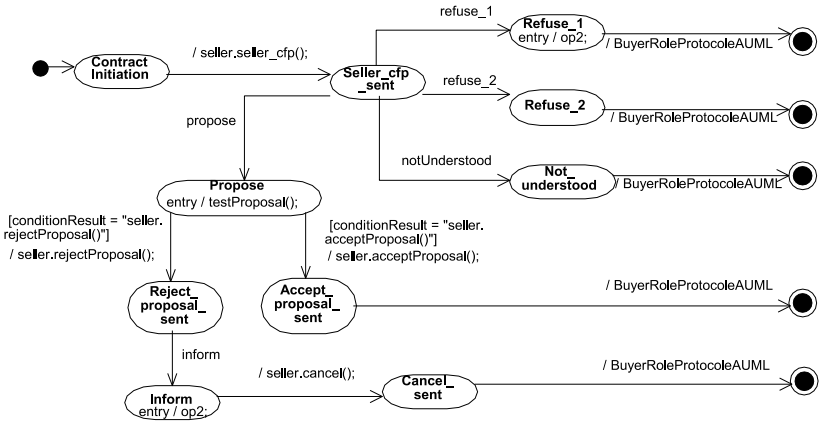

Figure 14. *Sous-machine de l'état ContractInitiation de la classe Buyer pour le rôle BuyerRole*

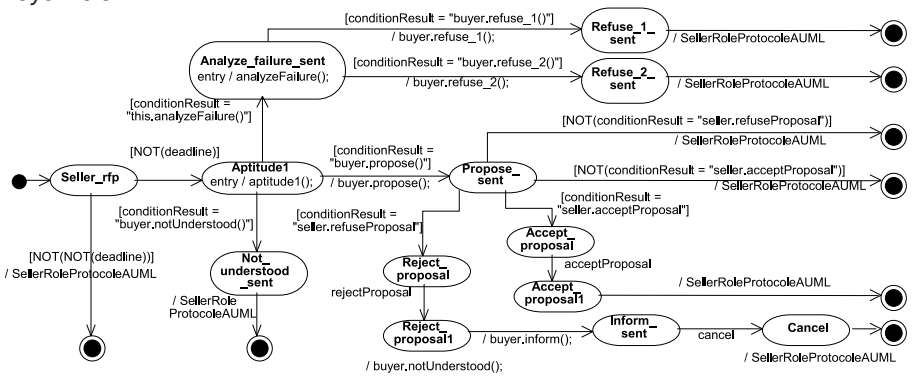

Figure 15. *Sous-machine de l'état Seller_cfp de la classe Seller pour le rôle SellerRole*

6.3.2. *Jonctions AUML*

Concernant les jonctions AUML, une opération stéréotypée «aptitude» est associée au processus de décision du branchement, et, par conséquent, une action d'entrée (entry) est ajoutée dans l'état correspondant à l'envoi du message. Cette action exécute l'aptitude qui renvoie la signature de l'opération à activer et le résultat est mémorisé dans une variable conditionResult. Dans le cas de jonctions XOR ou OR à n branches, il faut créer n transitions. Dans la figure 14, les conditions correspondant à la jonction XOR sont [conditionResult = "seller.acceptProposal"] et [conditionResult = "seller.refuseProposal"]. Ces conditions sont attachées aux transitions vers les états Accept_proposal_sent et Reject_proposal_sent.

Pour les branchements AND, un état appelé AND est créé, et la transition vers cet état est créée avec une action composite exécutant les opérations à effectuer en

séquence. Les contraintes, comme des échéances (*deadlines*), sont exprimées par des opérations renvoyant des booléens et traitées par des transitions conditionnelles : si la condition est vraie, l'automate continue (ainsi que le protocole), sinon il revient à l'état Idle, comme le montre la figure 15.

La distinction des rôles d'agent d'un agent est faite dans la structure concurrente des machines à états ; ainsi, cette distinction n'est pas à faire au moment de l'envoi : l'événement est envoyé à un agent, qui possède une ou plusieurs régions concurrentes capables de traiter l'événement, à ce moment-là.

La transition équivalente à une jonction AND porte, en action à exécuter, l'envoi de tous les événements des branches du AND. Par contre, le même message envoyé à différents rôles d'une instance ne fait l'objet que d'un seul envoi. Le nom de l'état d'arrivée est constitué de AND, auquel s'ajoute éventuellement un numéro pour obtenir un identifiant unique. Il convient de rechercher ensuite les chemins de continuité (ceux qui permettent d'arriver à une réception d'événement pour le rôle d'agent courant) pour chacun des événements envoyés afin de poursuivre l'arbre de la machine à états.

Avec la jonction AND, tous les événements sont envoyés ; avec la jonction XOR, un seul est envoyé. Nous étions alors dans des cas simples. Dans le cas des jonctions OR se pose le problème de la combinatoire de n événements. La proposition actuelle laisse l'opération «aptitude» effectuer dans son code les envois de messages selon son analyse, et la machine à états ne présume pas des chemins de continuité qui peuvent en résulter ; par contre, l'exception est chargée de lever un éventuel blocage dans un état (cas où aucune des transitions proposées en sortie n'est pas franchissable). Un message réflexif, lui, est considéré comme un traitement interne et vient compléter le code de la clause *entry* de l'état courant.

Lors de la génération de la machine à états, il y a prise en compte de la branche courante du protocole si des jonctions XOR sont rencontrées. Un problème peut survenir si l'on revient sur des rectangles d'activation distincts d'une même classe (ou rôle d'agent) via un même événement, sans qu'il y ait moyen de distinguer dans la machine à états d'où l'on vient ; cependant cette ambiguïté peut être détectée en traçant les messages et en les identifiant.

7. Discussion

Deux solutions sont envisageables pour étendre un métamodèle aux spécificités d'une méthode donnée : étendre les métamodèles ou utiliser un profil UML (Desfray, 2000). L'approche choisie pour l'expression des spécificités des agents coopératifs a été la définition d'un profil UML (plus précisément AUML) composé de stéréotypes de classes et de traits, ainsi que de règles de bon usage. Il aurait aussi été possible de définir un nouveau métamodèle au même niveau que UML ou AUML dans lequel apparaîtraient les agents et leurs composants. Un tel métamodèle peut bien sûr largement s'inspirer de métamodèles existants (comme AUML puise largement dans UML). Une telle approche a été envisagée pour ADELFE, et d'autres méthodes orien-

tées agent, afin d'unifier leurs métamodèles (Bernon *et al.,* 2005). Mais la proposition d'un nouveau métamodèle demande une définition rigoureuse et exhaustive du domaine envisagé. Or, il est indéniable, que la communauté agent, malgré les efforts nombreux, souffre d'un manque d'uniformité dans sa vision du domaine. Cependant, une approche par profil nous paraît préférable, compte tenu de sa facilité d'intégration dans des outils de conception existants (Rational Rose, OpenTool, etc.), contrairement à la proposition d'un nouveau métamodèle[6].

Outre les extensions apportées à OpenTool[7] (Picard, 2004), nous disposons d'un prototype implémentant une procédure de génération du code en Java. Il faut noter que notre chaîne est encore imparfaite et ne traite que de l'aspect statique sans possibilité d'unification avec l'aspect dynamique, ce qui motive l'utilisation de classes abstraites pour forcer une implémentation séparée des méthodes. Ainsi, il suffit au développeur d'hériter chacune des classes suffixées par `Base` puis de surcharger les méthodes pour implémenter son comportement. Cette approche permet de s'assurer que les visibilités sont bien respectées entre les différents constituants de l'agent. À terme, il pourrait être envisageable d'utiliser OCL (OMG, 2005a) pour spécifier des pré- et post-conditions sur les opérations des agents spécifiés dans le PIM. Ainsi, pour des opérations suffisamment simples, le code pourrait être directement généré à partir de la post-condition OCL. Bien sûr, ces contraintes OCL pourront permettre de vérifier la conformité du code écrit pour les opérations des agents. Cela pose naturellement la question de l'instanciation complète du modèle sous forme de code. Hormis les points suscités, il semble délicat d'obtenir une automatisation totale de la génération du code. En particulier, les paramètres utilisés pour conditionner le comportement d'un agent semble nécessiter l'utilisation de simulations afin de déterminer leurs valeurs.

Contrairement aux approches classiques du domaine plutôt orientées plateforme (comme JADE, par exemple) qui imposent un modèle d'agent particulier, notre proposition basée sur le MDA permet d'introduire une certaine souplesse. En effet, comme nous pouvons le voir sur la figure 5, le modèle d'agent utilisé est lui même conditionné par un métamodèle qui est donc modifiable. À ce jour, nous avons concentré nos efforts autour d'un seul modèle d'agents coopératifs proposé dans ADELFE, mais il est tout à fait possible d'introduire d'autres modèles d'agents très facilement, simplement en changeant de métamodèle. Cet aspect n'est pas actuellement exploité, mais il semble tout à fait possible de spécifier ce type de métamodèles en utilisant des paquetages UML stéréotypés. À terme, nous pourrons donc obtenir une solution complète de modélisation permettant de s'intéresser à la fois à des familles d'agents différentes, mais aussi à des plateformes agents différentes.

L'approche décrite dans cet article se focalise sur la définition du modèle d'agent et les règles de transformation associées à ce modèle. Comme l'approche par auto-organisation des AMAS considère l'organisation comme un résultat et non comme un moyen, nous ne tenons pas compte des organisations possibles ni même de l'expres-

6. Les éditeurs de logiciels de conception ayant déjà du mal à suivre les évolutions du métamodèle UML...
7. Outil de conception développé par TNI-Valiosys (`www.tni-valiosys.fr`)

sion de l'environnement du système. Ces deux aspects peuvent aussi faire l'objet d'une spécification au niveau PSM, après avoir établi, au niveau PIM, un langage de description pour les organisations multi-agents et pour l'environnement de ces systèmes. Il serait alors possible de passer du niveau agent au niveau multi-agent en spécifiant des propriétés de rôles fixes, de groupes en ce qui concerne les aspects organisationnels, par exemple, ou tout autre caractéristique attribuée aux systèmes multi-agents en général (Boissier *et al.,* 2004). Concernant l'environnement, celui-ci pourra être caractérisé par ses composants, son comportement, son ouverture, voire les contraintes de déploiement du système (type de réseaux, plate-forme de simulation, API orientée agent, etc).

8. Conclusion

Dans cet article, nous avons exposé des techniques de transformation de modèles pour assurer le passage de la conception à l'implémentation dans la continuité de la méthode ADELFE. Cette dernière repose sur un modèle d'agent coopératif structuré dont le comportement social est, entre autre, spécifié grâce à des protocoles AUML.

Les spécificités d'ADELFE sont intégrées par une extension du métamodèle UML/AUML par profilage. Les stéréotypes du profil permettent à la fois de contraindre la structure de l'agent, mais aussi son fonctionnement grâce à une restriction des visibilités entre modules et une extension de la sémantique des diagrammes de protocoles, et donc de leur implantation. Dans le cadre du MDA, à chacun des aspects (dynamique et statique) sont attribuées des règles de transformation de modèles. Tout d'abord, les classes issues de la conception sont transformées en ensembles de classes à visibilité restreinte, puis les protocoles en automates à état finis. Ces deux modèles peuvent ensuite être transformés en squelettes de code.

En l'état, nous avons illustré l'approche sur le modèle d'agent coopératif d'ADELFE, mais les techniques et outils utilisés sont paramétrables par des métamodèles d'agents et donc pertinents pour d'autres modèles d'agents (BDI, JADE, etc). Ces travaux ne s'avèrent donc pas limités aux agents coopératifs, mais peuvent être directement utilisés par des développeurs d'agents structurés en modules et utilisant des machines à état finis ou des protocoles AUML pour modéliser leur comportement dynamique et social.

9. Bibliographie

Arlabosse F., Gleizes M.-P., Occello M., « Méthodes de Conception », *Systèmes Multi-Agents*, vol. 29 of *Arago*, Editions Tech & Doc, p. 137-171, 2004.

Baude F., Caromel D., Huet F., Vayssiere J., « Communicating Mobile Active Objects in Java », *Proceedings of HPCN Europe 2000*, vol. 1823 of *LNCS*, Springer-Verlag, p. 633-643, 2000.

Bergenti F., Gleizes M.-P., Zambonelli F., (eds), *Methodologies and Software Engineering for Agent Systems*, Kluwer Publishing, 2004.

Bernon C., Cossentinon M., Gleizes M.-P., Turci P., Zambonelli F., « A Study of some Multiagent Meta-Models », *Agent-Oriented Software Engineering V : 5th International Workshop, (AOSE'04), New York, NY, USA, July 19, 2004. Revised Selected Papers*, vol. 3382 of *Lecture Notes in Computer Science (LNCS)*, Springer-Verlag, p. 62-77, 2005.

Besse C.*, « Recherche de conformation de molécules et apprentissage du potentiel de Lennard-Jones par systèmes multi-agent adaptatifs »*, Mémoire de Master 2 Recherche ¨Intelligence Artificielle, Raisonnement, Coopération, Langage¨, Université Paul Sabatier (Toulouse III), 2005.

Boissier O., Gitton S., Glize P., « Caractéristiques des systèmes et des applications », *Systèmes Multi-Agents*, vol. 29 of *Arago*, Editions Tech & Doc, p. 25-54, 2004.

Camps V., Gleizes M.-P., Glize P., « Une théorie des phénomènes globaux fondée sur des interactions locales », *Systèmes multi-agents – de l'interaction à la socialité – Actes des 6èmes JFIADSMA*, Hermès, p. 207-220, 1998.

Capera D., Gleizes M.-P., Glize P., « Mechanism Type Synthesis based on Self-Assembling Agents », *Journal on Applied Artificial Intelligence*, vol. 18, n° 9-10, p. 921-936, 2004.

Capera D., Picard G., Gleizes M.-P., Glize P., « A Sample Application of ADELFE Focusing on Analysis and Design : The Mechanism Design Problem », *Fifth International Workshop on Engineering Societies in the Agents World (ESAW'04), 20-22 October 2004, Toulouse, France*, vol. 3451 of *Lecture Notes in Artificial Intelligence (LNAI)*, Springer-Verlag, p. 231-244, 2005.

Chefrour D., André F., « Développement d'applications en environnements mobiles à l'aide du modèle de composant adaptatif ACEEL », *Langages et Modèles à Objets (LMO). Actes publiés dans la revue STI, série L'objet*, vol. 9, p. 77-90, 2003.

Cossentino M., « From Requirements to Code with the PASSI Methodology », *Agent-Oriented Methodologies*, Idea Group Publishing, p. 79-106, 2005.

David P.-C., Développement de composants Fractal adaptatifs : un langage dédié à l'aspect d'adaptation, PhD thesis, École des Mines de Nantes, 2005.

Desfray P., « UML Profiles Versus Metamodel Extensions : An Ongoing Debate », *OMG's UML Workshops : UML in the .com Enterprise : Modeling CORBA, Components, XML/XMI and Metadata Workshop*, p. 6-9, 2000.

Ferber J., *Les système multi-agents : Vers une intelligence collective*, InterEditions, 1995.

Garijo F., Gómez-Sanz J., Fuentes R., « The MESSAGE Methodology for Agent-Oriented Analysis and Design », *Agent-Oriented Methodologies*, Idea Group Publishing, p. 203-235, 2005.

Georgé J.-P., Gleizes M.-P., Glize P., « Conception de systèmes adaptatifs à fonctionnalité émergente : la théorie AMAS », *Revue d'intelligence artificielle*, vol. 17, n° 4, p. 591-626, 2003.

Giorgini P., Kolp M., Mylopoulos J., Castro J., « Tropos : A Requirements-Driven Methodology for Agent-Oriented Software », *Agent-Oriented Methodologies*, Idea Group Publishing, p. 20-45, 2005.

Gleizes M.-P., Link-Pezet J., Glize P., « An Adaptive Multi-Agent Tool for Electronic Commerce – IEEE Ninth International Workshops on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE 2000) », *Second International Workshop on Knowledge Media Networking, Gettysburg, USA*, IEEE Computer Society, p. 59-66, 2000.

Henderson-Sellers B., Giorgini P., (eds), *Agent-Oriented Methodologies*, Idea Group Publishing, 2005.

Huget M.-P., Odell J., Bauer B., « The AUML Approach », *in* , F. Bergenti, , M.-P. Gleizes, , F. Zambonelli (eds), *Methodologies and Software Engineering for Agent Systems (Chapter 8)*, Kluwer Publishing, p. 237-258, 2004.

Iglesias C. A., Garijo M., « The Agent-Oriented Methodology MAS-Common-KADS », *Agent-Oriented Methodologies*, Idea Group Publishing, p. 46-78, 2005.

Jacobson I., Booch G., Rumbaugh J., *Le processus unifié de développement logiciel*, Eyrolles, 2000.

Kleppe A., Warmer J., Bast W., *MDA Explained - The Model Driven Architecture : Practice and Promise*, Addison-Wesley, 2003.

Macchion E.*, « Plate-forme ¨système multi-agent adaptatif¨ pour une exploration comportementale transcriptionnelle - Application à la levure Saccharomyces cerevisiæ à partir de données métaboliques et de puces à ADN »*, Mémoire d'Ingénieur C.N.A.M. Informatique, Conservatoire National des Arts et Métiers de Toulouse, 2004.

Odell J., « Objects and Agents Compared », *Journal of Object Technology*, vol. 1, n° 1, p. 41-53, 2002.

Odell J., Van Dyke Parunak H., Bauer B., « Extending UML for Agents », *Proceedings of the Agent-Oriented Information Systems (AOIS) Workshop at the 17th National Conference on Artificial Intelligence (AAAI)*, p. 3-17, 2000.

OMG, OCL 2.0 Specification, Technical Report n° 2.0, ptc/2005-06-06, Object Management Group, 2005a.

OMG, Software Process Engineering Metamodel Specification, Technical Report n° 1.1, formal/05-01-06, Object Management Group, 2005b.

Picard G.*, « Méthodologie de développement de systèmes multi-agents adaptatifs et conception de logiciels à fonctionnalité émergente »*, Mémoire de thèse de doctorat, Université Paul Sabatier (Toulouse III), 2004.

Picard G., Bernon C., Gleizes M.-P., « Emergent Timetabling Organization », *Multi-Agent Systems and Applications IV - 4^{th} International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'05), 15-17 September 2005, Budapest, Hungary*, vol. 3690, Springer-Verlag, p. 440-449, 2005a.

Picard G., Gleizes M.-P., « The ADELFE Methodology – Designing Adaptive Cooperative Multi-Agent Systems », *Methodologies and Software Engineering for Agent Systems (Chapter 8)*, Kluwer Publishing, p. 157-176, 2004.

Picard G., Gleizes M.-P., « Cooperative Self-Organization to Design Robust and Adaptive Collectives », *2^{nd} International Conference on Informatics in Control, Automation and Robotics (ICINCO'05), 14-17 September 2005, Barcelona, Spain, Volume I*, INSTICC Press, p. 236-241, 2005b.

Shehory O., Sturm A., « Evaluation of Modeling Techniques for Agent-Based Systems », *Proceedings of the Fifth International Conference on Autonomous Agents*, ACM Press, p. 624-631, 2001.

Thiefaine A., Guessoum Z., Perrot J.-F., Blain G., « Génération de systèmes multi-agents à partir de modèles », *Actes des Journées Francophones sur les Systèmes Multi-Agents*, Hermès, p. 107-111, 2003.

Van Dyke Parunak H., « "Go to the Ant" : Engineering Principles from Natural Agent Systems », *Annals of Operations Research*, vol. 75, p. 69-101, 1997.