Thèse de Doctorat

Pour l'obtention du titre de

Docteur de l'Université de Reims Champagne Ardenne
en Informatique

Réutilisation des protocoles d'interaction et Démarche orientée modèles pour le développement multi-agents

Tarek Jarraya

Soutenue publiquement le 8 décembre 2006, à Reims.

Composition du jury :

Président :
Monsieur Jean Paul Sansonnet, Directeur de recherche au CNRS
Rapporteurs :
Monsieur Juan Pavón, Professeur à l'Université de Madrid
Monsieur Mikal Ziane, Maître de conférences - HDR à l'Université de Paris 5
Examinateur :
Monsieur René Mandiau, Professeur à l'Université de Valenciennes
Directeur de thèse :
Madame Zahia Guessoum, Maître de conférences - HDR à l'Université de Reims
Co-Directeur :
Monsieur Herman Akdag, Professeur à l'Université de Reims

A mes parents.

Remerciements

Il est toujours délicat de remercier l'ensemble des personnes qui ont contribué à l'aboutissement de ce travail de recherche. Que ceux qui ne sont pas mentionnés ne m'en tiennent pas rigueur.

Je tiens à remercier tout d'abord mes directeurs de thèse, Zahia Guessoum et Herman Akdag. Je ne serais pas arrivé jusque là sans l'aide de Zahia. Je la remercie pour la qualité de ses conseils, pour sa disponibilité et pour son investissement constant. J'espère ne pas l'avoir trop déçu. Je remercie également Herman Akdag pour m'avoir facilité l'inscription ainsi que pour ses conseils. Je me souviens encore de notre première rencontre au LIP6.

Je tiens à remercier Juan Pavón et Mikal Ziane pour avoir accepté de rapporter mon document de thèse et de m'avoir apporté leur regard sur mon travail. Je remercie également René Mandiau pour avoir examiné ce document. Je remercie Jean-Paul Sansonnet de m'avoir honoré en acceptant de présider mon jury.

Je remercie tous les membres du LERI, plus particulièrement Yannick Rémion qui m'ont accueilli et qui m'ont permis de m'épanouir aussi bien dans mes recherches que dans mes enseignements. Je ne manque pas de remercier aussi le personnel administratif de l'IUT, et particulièrement ceux du département informatique.

Je tiens à remercier Nora Faci ma collègue avec qui j'ai partagé le même bureau. Je lui souhaite un bon achèvement de ses travaux de thèse. Je remercie aussi mon ami Smaine Mazouzi pour son aide et ses conseils qui m'ont été très utiles jusqu'aux dernières lignes de ce manuscrit. Mes remerciements vont aussi à mon ami Ghassen Bouslama avec qui j'ai partagé les bons et mauvais moments de ces années de thèse.

Je tiens à exprimer toute ma gratitude et mon amitié à mon cher frère Mohamed Anouar Sallami, pour m'avoir accompagné et constamment soutenu. Je tiens à manifester toute mon affection et mon amitié à mes amis Rabie Benattitallah, Walid Bouslimi et Mohamed Zikri dont le soutien et l'aide étaient sans limites.

Je remercie enfin mes parents, qui m'ont toujours soutenu dans mes études. Mes pensées vont également vers mon frère Mahmoud et mes soeurs Myriam et Fatma.

Résumé

Les architectures modulaires d'agents offrent plusieurs facilités telles que la possibilité d'inclure des bibliothèques de composants réutilisables. Ainsi, pour améliorer le développement des agents interactifs, nous proposons une représentation componentielle des protocoles d'interaction.

La démarche que nous avons adoptée consiste à étudier d'abord les principaux protocoles d'interaction, plus particulièrement les protocoles d'interaction standardisés par FIPA. Cette étude consiste à analyser le fonctionnement de chaque protocole pour identifier les données que doit spécifier un développeur afin de l'intégrer à l'agent. Grâce à cette étude nous avons pu construire une ontologie qui regroupe les concepts de l'interaction, communs à l'ensemble des protocoles d'interaction.

Nous nous sommes basé sur cette ontologie pour développer le framework INAF (INteractive Agent Framework). L'intérêt de ce framework est de promouvoir la réutilisation et l'adaptabilité des agents dans un environnement dynamique. INAF offre une bibliothèque de protocoles d'interaction et une architecture de base pour les agents interactifs. Afin de valider notre solution, nous avons utilisé le framework développé pour réaliser plusieurs applications, telles qu'un système de vente aux enchères, un système de gestion des emplois du temps, et un système de gestion pour une chaîne de montage.

Un constat de l'expérience avec INAF c'est que son utilisation demande une certaine connaissance des concepts et techniques des systèmes multi-agents. Dans le but de réduire la complexité inhérente à la diversité des concepts multi-agents, nous proposons une nouvelle méthode de développement, nommée MDAD (Model Driven Agent Development), qui se base sur l'approche MDA (Model Driven Architecture), proposée par l'OMG.

Notre méthode MDAD décrit le système multi-agents à travers un niveau conceptuel (PIM) et un niveau d'implémentation (PSM) qui sont décrits par un ou plusieurs méta-modèles. Le passage d'un niveau à un autre est un processus automatique de transformation, piloté par un ensemble de règles, qui représentent le savoir-faire des experts du domaine d'application et des concepteurs en multi-agents.

Mots-clés

Systèmes multi-agents, Protocoles d'interaction, Agent interactif, Réutilisation, Méthodologies mutli-agents, MDA, Transformation de modèles.

Abstract

The modular agent architectures provide many facilities, like the possibility to include libraries of reusable components. Thus, to improve the development of interactive agents, we propose a component representation of interaction protocols.

The approach that we have adopted consisted first in studying principal interaction protocols, specifically protocols standardized by FIPA. This study consists in analyzing the process of each protocol, in the aim to identify data that the developer specifies to integrate it in the agent. Thanks to this study we have build an ontology which contains interaction concepts, common to all protocols.

We were based on this ontology to develop the INAF (INteractive Agent Framework) framework. The aim of this framework is to promote the reuse and the adaptation of agents in dynamic environment. INAF provide a library of interaction protocols and a basic architecture for interactive agents. To validate our solution, we have used our framework to develop different applications, like an auction system, a timetable management system.

A review with experience with INAF, that its use requires the knowledge of some concepts and techniques in multi-agent systems. To reduce the complexity inherent in the diversity of multi-agent concepts, we propose a new development method, named MDAD (Model Driven Agent Development), which is based on the MDA (Model Driven Architecture) approach, proposed by OMG.

Our method MDAD describes the multi-agent system through a conceptual level (PIM) and an implementation level (PSM) which are described by one or many metamodels. The transition from one level to the other is an automatic transformation process, driven by a set of rules. These rules represent the know-how of experts in application domain and multi-agent systems designers.

Key words

Multi-Agent Systems, Interaction protocols, Interactive agent, Reusability, Agent oriented software engineering, Model driven architecture, Models transformation.

Table des matières

Chapitre 1

Introduction Générale

1.1 Contexte et problématique

Les systèmes multi-agents constituent aujourd'hui une nouvelle technologie pour la conception et le contrôle de systèmes complexes. Un système multi-agents est un système composé d'entités logicielles ou matérielles autonomes appelées agents. L'approche multi-agents repose sur plusieurs théories et concepts qui trouvent leurs origines dans plusieurs disciplines tels que la sociologie, la psychologie, les systèmes réparties, le génie logiciel.

Le cadre général de cette thèse est l'ingénierie des systèmes multi-agents, dont les principales activités de recherche s'intéressent à l'élaboration de méthodologies pour la conception des applications utilisant une approche multi-agents, et au développement de plates-formes et de frameworks aidant à l'implémentation de ces applications.

Demazeau définit le système multi-agents selon cinq aspects [Dem95] : l'agent, l'environnement, l'interaction, l'organisation et l'utilisateur. L'interaction est ainsi un des aspects clés des systèmes multi-agents. Elle offre un moyen pour assurer la coopération et la négociation entre agents. Sans interaction (ou communication), l'agent n'est qu'un individu isolé, sourd et muet, renfermé sur sa boucle perception-délibération-action [Fer95]. Bien que l'interaction et la communication sont souvent confondues dans la littérature, la communication représente la transmission d'information entre agents, alors que l'interaction comprend l'action sur le monde, ainsi que la communication entre les agents du système [BD01]. En effet, deux modes de communication se distinguent dans la littérature : la communication indirecte qui se fait par transmission de signaux via l'environnement, et la communication directe, qui correspond aux échanges de messages entre les agents.

Nous nous penchons sur les systèmes multi-agents dont l'interaction est directe, régie par les protocoles d'interaction. Un protocole d'interaction est un enchaînement prédéfini de messages. Les protocoles d'interaction sont introduits dans les systèmes multi-agents dans le but de faciliter la spécification et l'implémentation de l'interaction entre les agents. D'après la définition de *FIPA*[1] (*Foundation of Intelligent Physical Agents*), un protocole d'interaction est un pattern commun de communication. Ainsi la spécification et l'implémentation du protocole doivent être indépendantes du domaine d'application et de l'architecture interne de l'agent.

Plusieurs travaux se sont préoccupés de la conception, la représentation et la validation des protocoles d'interaction [Hug01] [Maz01] [OPB00]. Mais peu de travaux se sont intéressés à l'implémentation des protocoles d'interaction et des agents interactifs. En effet, rares sont les plates-formes qui proposent une implémentation des protocoles d'interaction. A notre connaissance seule la plate-forme *JADE* [BCTR04] propose une bibliothèque de protocoles d'interaction, que le développeur peut réutiliser pour le développement de ses agents.

L'absence d'une implémentation réutilisable des protocoles d'interaction oblige le développeur à implémenter lui même le protocole d'interaction. Cette tâche exige une connaissance de la spécification du protocole pour pouvoir dégager l'activité des différents rôles et de les intégrer au sein de l'agent ; une lourde tâche pour un utilisateur novice. En outre, ce développement engendre des difficultés dans la maintenance du code de l'agent en cas de besoin, par exemple, pour changer de protocole. De plus, l'agent reste limité à l'utilisation de quelques protocoles d'interaction, spécifié à l'avance par le développeur. Les architectures d'agent proposées par Sierra et al. [SFJ00], Karacapilidis et. al.[KM01] et Rahwan [Rah02], se limitent à l'utilisation d'un seul protocole d'interaction. En conséquence, l'agent ne peut pas ajouter dynamiquement de nouveaux protocoles, une propriété importante pour un agent évoluant dans un environnement ouvert et dynamique.

Par contre, au niveau des méthodologies multi-agents, nous remarquons une grande utilisation des protocoles d'interactions. Par exemple, PASSI [CP02] utilise les protocoles d'interaction de FIPA [FIP02b] pour la spécification de l'interaction des agents. Cependant, un système multi-agents conçu avec PASSI et implémenté sous MadKit [GJM00] ou JACK [Gro04] demandera un effort supplémentaire pour l'implémentation de l'interaction car, ces plates-formes ne proposent pas une implémentation des protocoles d'interaction. Par conséquent, l'implémentation de l'interaction des agents nécessitera un grand effort du développeur.

Cela est aussi vrai pour les autres aspects multi-agents (i.e. agent, organisation...) qui présentent des différences entre leurs conceptions méthodologiques et leurs implémentations avec les plates-formes. Ces différences, qui constituent en réalité une richesse de l'approche multi-agents pour traiter des problèmes dans des domaines assez divers, créent des difficultés aux

[1] www.fipa.org

utilisateurs novices qui n'ont pas nécessairement la double maîtrise des méthodologies et des plates-formes. Cela est principalement du à l'absence d'une démarche claire qui assure le passage de la conception à l'implémentation.

Les méthodologies actuelles proposent des solutions plus ou moins complètes pour le développement des systèmes multi-agents. Certaines méthodologies, tels que *GAIA* [ZJW03] et *ROADMAP* [JPS02], couvrent seulement les phases d'analyse et de conception du cycle du développement, mais elles ne proposent pas de solutions pour l'implémentation de leurs systèmes multi-agents. Leurs différents modèles, et particulièrement leurs modèles conceptuels, sont de simples diagrammes graphiques et textuels, qui ne peuvent pas être exploités, d'une manière automatique, pour l'implémentation du système.

D'autres méthodologies, comme *TROPOS* [BPG+04] et *PASSI* [CP02], vont jusqu'à l'implémentation du système avec une plate-forme donnée, en générant une partie du code. La démarche proposée pour passer à l'implémentation est généralement manuelle. De plus, les outils que proposent ces méthodologies assistent le développeur seulement dans la définition des modèles d'analyse et de conception.

En résumé, aucun savoir-faire des spécialistes des méthodologies et des plates-formes n'est proposé à l'utilisateur, pour aller à une implémentation opérationnelle de son système. Ce travail de recherche propose des solutions pour ces problèmes d'ingénierie des systèmes multi-agents.

1.2 Une représentation et une implémentation réutilisables des protocoles d'interaction

Pour améliorer le développement des agents interactifs, nous proposons une représentation componentielle des protocoles d'interactions. Ainsi, les agents pourront définir dynamiquement leurs interactions [JGD05]. La démarche que nous avons adopté consiste à étudier d'abord les principaux protocoles d'interaction, plus particulièrement les protocoles d'interaction proposés par *FIPA*. Cette étude consiste à analyser les informations nécessaires pour l'exécution de chaque protocole. Cela nous a permis de déterminer les informations minimales que doit spécifier le développeur pour exécuter un protocole donné. Pour faciliter la réutilisation des protocoles, nous avons défini une ontologie qui regroupe les différents concepts de l'interaction communs à l'ensemble des protocoles d'interaction [JGF04].

Ensuite, nous nous sommes basé sur cette représentation des protocoles d'interaction pour développer le framework *INAF* (*INteractive Agent Framework*) [JGD04] [JG06]. *INAF* offre une bibliothèque de protocoles d'interaction et une architecture de base pour les agents inter-

actifs. L'intérêt de ce framework est de faciliter le développement des agents interactifs par la réutilisation des protocoles d'interaction.

1.3 Une démarche de développement multi-agents basée sur la transformation de modèles

A la différence des méthodologies existantes, nous proposons une nouvelle démarche de développement qui se base sur l'approche *MDA* [orm01] (*Model Driven Architecture*), proposée par l'*OMG*[2] (*Object Managment Group*). L'idée principale de cette approche est d'assurer le passage de la représentation conceptuelle d'un système vers son implémentation d'une manière automatique par transformation de modèles.

Notre démarche s'intéresse à la description du système multi-agents à travers les deux niveaux *PIM* (*Platform Independent Model*) et *PSM* (*Platform Specific Model*). Chacun de ces niveaux se base sur un méta-modèle. Le passage du niveau conceptuel (PIM) au niveau d'implémentation (PSM) est un processus automatique de transformation, qui se base sur un ensemble de règles, qui modélisent le savoir-faire des experts du domaine d'application et des concepteurs en multi-agents.

1.4 Organisation du document

Ce document est organisé en cinq chapitres dont les deux premiers présentent l'état de l'art des thèmes de recherche abordés dans cette thèse. Le premier chapitre est axé sur les protocoles d'interaction. Il présente et analyse plusieurs approches de représentation des protocoles afin de situer notre travail sur l'implémentation des protocoles d'interaction.

Le deuxième chapitre s'intéresse aux méthodologies multi-agents. Nous nous sommes intéressé plus particulièrement au passage de la phase de conception à la phase d'implémentation. Cette étude a montré une difficulté de passage à l'implémentation due principalement à l'absence d'une démarche claire et complète pour la phase d'implémentation des méthodologies.

Le troisième chapitre développe notre approche componentielle pour la représentation des protocoles d'interaction. Nous commençons par analyser les principaux protocoles d'interaction de *FIPA*. A partir de cette analyse, nous proposons notre ontologie *ONTOPRO* des protocoles d'interaction.

[2] www.omg.org

Le quatrième chapitre introduit notre framework d'interaction *INAF*. Nous donnons un aperçu sur les structures et les services offerts par *INAF* pour le développement des agents interactifs. *INAF* a été testé avec deux applications : la gestion des emplois du temps et la gestion logistique d'une chaîne de montage de composants.

Le cinquième chapitre présente notre démarche pour le développement des systèmes multi-agents. Nous commençons par donner un aperçu sur l'approche *MDA* (*Model Driven* Architecture). Nous décrivons ensuite les différentes étapes de notre démarche *MDAD* (*Model Driven Agent Development*) : la spécification du méta-modèle du *PIM*, la spécification du méta-modèle du *PSM*, et la définition des règles de transformation. Nous illustrons notre démarche avec l'application de gestion des emplois du temps.

Chapitre 2

Protocoles d'Interaction

2.1 Introduction

Les communications dans les systèmes multi-agents, comme chez les humains, sont à la base des interactions et de l'organisation des agents. Nous distinguons essentiellement deux modes de communication : la communication indirecte qui est une communication par signaux via l'environnement, et la communication directe qui procède à un échange de messages entre les agents.

Le mode de communication par échange de messages entre les agents provient initialement du domaine des acteurs [Hew77]. Il permet à un agent de planifier un acte de communication envers un autre agent, on parle alors de communication intentionnelle. Cette approche s'est inspirée de l'interaction sociale telle que nous la trouvons dans d'autres contextes comme la communication entre les humains. A la différence des autres approches, celle-ci a donné lieu à des théories calculatoires formelles permettant de savoir comment communiquer incrémentalement ou comment ajuster un plan de communication. Ceci est fait dans le but de s'adapter à un monde incertain, où les circonstances changent, rendant ainsi possible la communication complexe qui consiste en l'échange d'informations stratégiques [KP01].

L'interaction inter-agents regroupe et combine plusieurs types de messages. Cette combinaison se traduit par des enchaînements conversationnels qui peuvent être modélisés par deux approches :

– Dans la première approche, l'agent construit son propre plan de communication au moment où il en a besoin pour réaliser sa tâche. Ainsi, l'interaction n'est pas définie a priori, elle est émergente. Les connaissances et les buts des agents régissent l'interaction en spécifiant le message à envoyer (acte de communication, agents destinataires, le contenu

du message,...). Cette approche donne à l'agent plus de flexibilité dans son interaction. Cependant, dans cette approche l'agent doit avoir suffisamment de connaissances sur la sémantique des messages et qu'il doit être doté d'un mode de raisonnement lui permettant de mener ses interactions.

– Dans la deuxième approche, l'interaction se conforme à un enchaînement de messages spécifié à l'avance. Les règles qui régissent l'échange de messages forment le *protocole d'interaction*. De plus, le protocole d'interaction définit le type des messages qui doivent être échangés. L'agent interactif doit se conformer aux règles de conversation dictées dans le protocole. Comparée à la première approche, l'interaction basée sur les protocoles d'interaction ne nécessite pas d'architecture complexe au sein de l'agent. *"Of course a straightfoward way of reducing some of the search space of possible responses to agent messages is by using conversation policies, or interaction protocols"* [GB99a]. Ainsi, un protocole d'interaction spécifie un ensemble limité de réponses possibles pour un type spécifique de messages. Cette approche d'interaction, basée sur les protocoles d'interaction, est celle considérée dans la suite de ce rapport.

Le but de ce chapitre est de présenter la notion de protocole d'interaction et la problème de son implémentation ad-hoc pour le développement des agents interactifs.

Nous commençons ce chapitre par donner un bref aperçu sur la théorie des actes de langages qui est à l'origine des langages de communication en multi-agents. Ces derniers sont décrits dans la section 2.3. La section 2.4 présente quelques définitions des protocoles d'interaction. La section 2.5 donne la spécification des principaux protocoles d'interaction, et plus particulièrement ceux définis par la FIPA. Nous consacrons la section 2.6 à la classification des protocoles d'interaction. Nous présentons dans la section 2.7 les différentes situations dans lesquelles les agents font appel à l'interaction. Nous présenterons dans la section 2.8 les récents travaux relatifs à la représentation des protocoles d'interaction. La section 2.9 est dédiée aux outils et aux plates-formes qui offrent une implémentation des protocoles d'interaction. Nous introduisons, à la fin de ce chapitre, la problématique de l'implémentation ad-hoc des agents interactifs et des protocoles d'interaction.

2.2 La théorie des actes de langage

La théorie de la philosophie du langage a été introduite dans les années 60 par les travaux d'Austin [Aus62]. Dans son ouvrage *How to do things with words*, Austin montre que toute énonciation, permettant la réalisation d'un acte qui, en tant que tel, vise à accomplir quelque chose. Cet acte est appelé *acte de langage*.

Austin a défini trois types d'actes de langage pour une énonciation :
- l'*acte locutoire* qui est l'acte de dire quelque chose. Il est satisfait lorsque l'énoncé est correctement formulé,
- l'*acte illocutoire* qui représente l'action effectuée en disant quelque chose, par exemple une requête ou une déclaration,
- l'*acte perlocutoire* qui correspond à l'effet obtenu en disant quelque chose, par exemple dissuader, convaincre autrui.

Pour Austin, l'élément illocutoire est l'acte le plus important et il est la clé de l'énonciation. Il est même d'usage de réduire l'acte de langage à une simple composante illocutoire.

Les travaux d'Austin ont été repris par plusieurs chercheurs, en particulier Searle [Sea69], qui a considéré que tout acte de langage porte un acte illocutoire qui appartient à l'une des cinq catégories suivantes :
- acte assertif : le locuteur exprime comment les objets auxquels il se réfère sont dans le monde. Il s'agit des assertions, des informations, des témoignages, des démentis, etc.
- acte commissif : l'auteur s'engage à accomplir une action. Il s'agit des promesses, des voeux, des menaces, etc.
- acte directif : le locuteur fait en sorte que l'interlocuteur accomplisse une action. Il s'agit des demandes, des questions, des ordres, des conseils, etc.
- acte expressif : le locuteur manifeste son état mental face à un état de chose. Il s'agit des excuses, remerciements, félicitations, récriminations, etc.
- acte déclaratif : le locuteur accomplit au moment de l'énonciation l'action qu'il dit accomplir. Il s'agit de définitions, de condamnations, des ratifications, etc.

Ces travaux sont à la base des langages de communication multi-agents (i.e. KQML et FIPA-ACL) que nous présentons dans la section suivante.

2.3 Langages de communication

L'intérêt des langages de communication est de faciliter l'échange et l'interprétation des messages et l'interopérabilité entre les agents. Ces langages se focalisent essentiellement sur la manière de décrire exhaustivement des actes de communication d'un point de vue syntaxique et sémantique.

Finin et al. [FFMM94] déclarent que les langages de communication entre les agents doivent satisfaire certaines propriétés :
- une forme déclarative,
- une syntaxe simple et lisible,

- faire la distinction entre le langage de communication et le langage de contenu. Le langage de communication décrit les actes de langage et le langage du contenu permet d'exprimer les informations contenues dans le message,
- avoir une sémantique du langage ayant un fondement théorique et une description formelle,
- avoir une implémentation efficace, en vitesse et en bande passante.

Le premier langage qui a été introduit est *KQML*, pour *Knowledge Query and Manipulation Language*. A l'origine, *KQML* a été développé pour échanger des informations et des connaissances entre des systèmes à base de connaissances. Il a été ensuite repris dans [FFMM94] pour décrire les messages échangés entre les agents. Le deuxième langage dit *FIPA-ACL* [FIP02a] (*FIPA Agent Communication Language*), est proposé dans le cadre d'un travail de standardisation mené au sein l'organisation *FIPA*[1] (*Foundation of Intelligent Physical Agents*). *FIPA-ACL* est une extension du langage *KQML*.

FIPA-ACL est fondé sur vingt et un actes communicatifs, exprimés par des performatifs, qui peuvent être groupés selon leurs fonctionnalités de la façon suivante[2] :

- passage d'information : *Inform, Inform-if, Inform-ref, Confirm, Disconfirm,*
- réquisition d'information : *Query-if, Query-ref, Subscribe,*
- négociation : *Accept-proposal, Cfp, Propose, Reject-proposal,*
- distribution de tâches (ou exécution d'une action) : *Request, Request-when, Request-whenever, Agree, Cancel, Refuse,*
- manipulation des erreurs : *Failure, Not-understood.*

Un message *FIPA-ACL* peut contenir une partie ou la totalité des éléments décrits dans le tableau 2.1. Les éléments nécessaires pour la transmission d'un message changent selon la situation. Si un agent ne reconnaît pas ou ne peut pas traiter un ou plusieurs éléments, alors il peut répondre avec le message *Not-understood*.

Voici un exemple de message *FIPA-ACL* qui est envoyé par l'agent A à l'agent B :

```
(inform
        :sender A
        :receiver B
        :reply-with devis12
        :language Prolog
        :ontology Ordinateur
        :content prix(HP,1500 EUR))
        :conversation-id conv01
        :reply-by 10 min)
```

[1] www.fipa.org

[2] http ://turing.cs.pub.ro/auf2/html/chapters/chapter4/chapter_4_5_1.html

TAB. 2.1 – Les éléments d'un message *FIPA-ACL*

Elément	**Signification**
performatif	le type de l'acte communicatif
sender	l'émetteur du message
receiver	le destinataire du message
reply-to	le participant à l'acte de communication
content	le contenu du message (l'information transportée par le performatif)
language	le langage dans lequel le contenu est représenté
encoding	décrit le mode d'encodage du contenu du message
ontology	le nom de l'ontologie utilisé pour donner un sens aux termes utilisés dans le content
protocol	le nom du protocole d'interaction
conversation-id	l'identifiant de la conversation
reply-with	un identifiant du message, en vue d'une référence ultérieure
in-reply-to	il référence le message auquel l'agent est entrain de répondre (précisé par l'attribut reply-with dans le précédent message de l'émetteur)
reply-by	Un délai pour répondre au message

L'agent A informe son interlocuteur que le prix d'un ordinateur *HP* est de 1500 euro. Le contenu du message est exprimé avec le langage *Prolog*. L'ontologie utilisée est celle des ordinateurs. Ce message fait partie d'une conversation ayant comme identifiant *conv01*. L'agent B est contraint par un durée de 10 minutes pour donner suite à ce message.

Les langages de communication, sont pleinement utilisés pour la spécification des protocoles d'interaction. Les protocoles d'interaction sont utilisés par les agents pour régir leurs interactions. Nous donnons dans la section suivante quelques définitions, existantes dans la littérature, des protocoles d'interaction.

2.4 Quelques définitions des protocoles d'interaction

Nous retrouvons dans la littérature plusieurs définitions des protocoles d'interaction. Nous présentons, dans ce qui suit, les plus répondues.

L'organisation de standardisation FIPA donne la définition suivante : *"The interaction protocol is a common pattern of conversation, used generally to perform some tasks"*. Cette définition est très générale et manque de détails. Un pattern est solution récurrente décrivant et résolvant un problème général dans un contexte particulier. Le concepteur d'un pattern propose une solution générique pour un problème qui est souvent rencontré dans divers développements. Cette

solution est présentée sous une forme indépendante d'un langage de programmation particulier, ce qui oblige de l'implémenter pour une application donnée. Définir un protocole d'interaction comme *pattern* laisse supposer que la description du protocole devrait être générique et indépendante du contexte applicatif.

La définition de FIPA a été reprise par Cossentino [Cos03a], qui décrit de la sorte un protocole d'interaction : *"Agent interaction protocol is common pattern of conversations used to perform some generally useful task"*. Et il ajoute :*" The protocol is often used to facilitate a simplification of computational machinery needed to support a given dialogue task between two agents"*. La deuxième partie de cette définition résume bien le but des protocoles d'interaction d'un point de vue fonctionnel.

Dans la méthodologie Gaia [ZJW03], Zambonelli et al. ont utilisé la définition suivante : *"...a protocol can be viewed as an institutionalized pattern of interaction. That is, a pattern of interaction that has been formally defined and abstracted away from any particular sequence of execution steps, to focus attention on the essential nature and purpose of the interaction, rather than on the precise ordering of particular message exchanges.*

Huget propose une définition assez détaillée :*"Un protocole est un ensemble de règles qui guident l'interaction entre plusieurs agents. Pour un état donné du protocole, il n'existe qu'un nombre fini de messages admis en émission ou réception. Si un agent accepte d'utiliser un protocole, il accepte de se conformer à ce protocole et à en respecter les règles. De plus, il approuve la sémantique du protocole.*

Une règle est soit syntaxique, soit sémantique. Une règle syntaxique porte sur l'architecture du protocole, i.e. la construction des transitions reliant les états du protocole. Ces règles syntaxiques sont décrites à l'aide d'un formalisme. Les règles sémantiques définissent les actions que les agents doivent effectuer lors de l'émission et de la réception des messages. En effet, la réception d'un message dont le performatif est inform influe sur le comportement de l'agent car l'agent va ajouter à sa base de connaissances, les informations du message. Il croit cette information à partir de cet instant. L'utilisation de protocole est intéressante car elle permet d'arriver plus vite à la solution, i.e. la fonction du protocole est réalisée. En effet, les agents connaissent quels sont les messages qu'ils peuvent recevoir pour un état donné de l'interaction, quels sont les messages qu'ils peuvent envoyer et quelles sont les règles qui guident le choix lorsqu'il y a indéterminisme entre plusieurs messages possibles. Les agents convergent plus vite vers la solution".

Greaves et al. [GHB00] donnent la définition suivante : *"Interaction protocols are descriptions of standard patterns of interaction between two or more agents. They constrain the possible sequences of messages that can be sent amongst a set of agents to form a conversation of a particular type."*

Il sort de ces définitions quatre notions fondamentales qui caractérisent un protocole d'interaction :

- un protocole d'interaction est un pattern d'interaction. Ce qui explique le besoin de le représenter d'une manière générique, indépendamment du contexte d'application.
- chaque protocole d'interaction a un but.
- un protocole d'interaction fait intervenir deux ou plusieurs agents. Chacun de ces agents joue un rôle qui permet de l'identifier au cours de l'interaction.
- le protocole d'interaction définit les règles qui permettent de régir une interaction. Ces règles définissent l'ordonnencement des messages, ainsi que les actions auxquelles le protocole fait appel.

2.5 Exemples de protocoles d'interaction

Nous décrivons dans cette section certains protocoles d'interaction de FIPA qui serviront comme point de départ pour une analyse détaillée de l'activité des protocoles d'interaction (voir section 4.4 du chapitre 4).

2.5.1 Le protocole Contract Net

Le protocole Contract Net [Smi81], proposé par Smith en 1981, fut l'une des premières solutions au problème d'allocation de tâches auquel fait face généralement un ensemble de résolveurs de problèmes. Il facilite le contrôle distribué de l'exécution de tâches coopératives avec une communication efficace entre les noeuds d'un résolveur distribué de problèmes.

Dans ce protocole, les agents peuvent prendre deux rôles (figure 2.1) : *manager* ou *contractant*. Le manager est responsable de la surveillance de l'exécution d'une tâche et du traitement des résultats de cette exécution. Un contractant fait une proposition au manager pour réaliser la tâche. En cas d'acceptation, il est responsable de l'exécution effective de cette tâche.

2.5.2 Les protocoles d'enchère

Nous retrouvons dans la littérature plusieurs types d'enchères, mais on se limite dans cette section à la présentation des types d'enchères les plus utilisés dans les systèmes multi-agents. Les protocoles enchères qui nous décrivons dans cette section font appel à un commissaire-priseur et plusieurs participants.

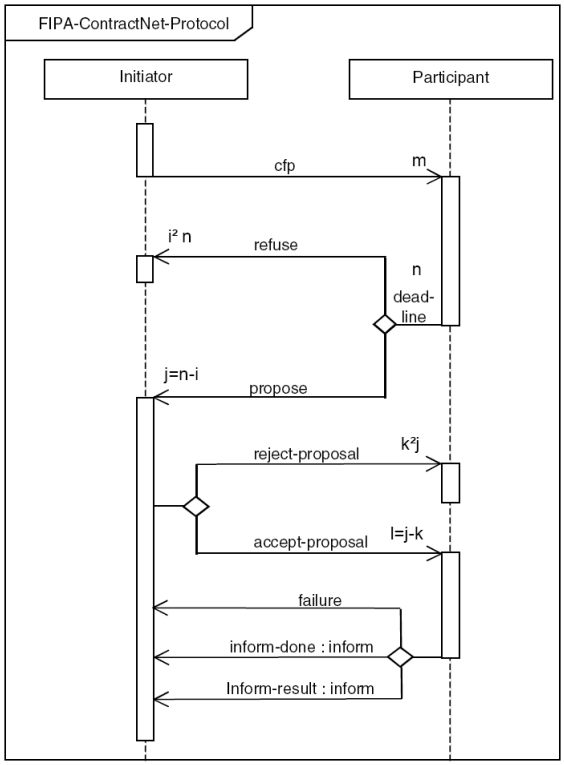


FIG. 2.1 – Les deux rôles du protocole *FIPA-Contract-Net*

Dans **l'enchère anglaise** [FIP02b], le commissaire-priseur initie l'enchère en annonçant un premier prix initialement inférieur à la valeur estimée du produit. Après chaque annonce, le commissaire-priseur attend pendant un certain temps pour voir s'il existe des participants acceptant cette offre. Dès qu'un participant se manifeste, le commissaire-priseur annonce une nouvelle proposition de prix égale au dernier prix incrémenté d'une valeur appelée *pas d'incrémentation*. L'enchère se termine quand aucun des participants ne se manifeste. A ce moment là, le commissaire-priseur compare le prix de la dernière offre acceptée avec la valeur estimée du produit. Si le prix de l'offre est supérieur à la valeur estimée alors l'enchère est accordée au participant qui s'est prononcé pour cette offre. Dans le cas contraire, les participants sont informés de l'annulation de la vente. Le manager peut réexaminer au cours de l'enchère la valeur estimée du produit, pour éviter toute annulation de la vente.

Dans **l'enchère hollandaise [FIP02b] (ou allemande)**, le commissaire-priseur annonce au départ une valeur très élevée par rapport à la valeur estimée du produit. Puis il commence par baisser le prix jusqu'à ce qu'un participant signale son acceptation du prix proposé. Le vendeur possède une estimation de la valeur du produit (*prix de réserve*) au-dessous de laquelle il ne vend pas le produit. Si l'enchère atteint le prix de réserve sans qu'il y ait d'acheteurs, l'enchère se termine.

Dans l'**enchère Vikrey**, les propositions sont privées, aucun participant ne connaît le contenu des autres enchères. Les enchères Vickrey permettent de vendre un article unique, comme les enchères anglaises. La différence est que le soumissionnaire le plus élevé obtient l'article au prix offert par le deuxième plus haut soumissionnaire.

Dans l'**enchère au premier prix**, des enchères sont soumises au vendeur sous un certain format, sans qu'aucun des participants ne sache le contenu des autres enchères. Le gagnant paye le prix qu'il avait soumis.

Dans la **double enchère**, les participants sont des acheteurs et des vendeurs qui négocient sur un même produit. L'enchère peut démarrer par l'envoie d'une offre ou d'une proposition. Les vendeurs envoient des offres qui démarrent avec le plus haut prix puis elles décroissent, et des propositions sont faites par les acheteurs dont les prix évoluent dans le sens inverse des offres. Les offres et les propositions sont publiques, chaque participant est au courant de toutes les offres et les propositions soumises (l'émetteur et le prix proposé). La négociation peut s'achever si un acheteur accepte une offre proposée par un vendeur ou si un vendeur accepte une proposition d'un acheteur.

Nous avons cité dans cette section les protocoles d'interaction les plus utilisés par la communauté multi-agents. Cependant, il existe d'autres protocoles d'interaction que nous n'avons pas cités dans cette section, tels que les protocoles de vote [Tay95] et le protocole de Bargaining [SGBP03].

2.6 Classification des protocoles d'interaction

Cette section présente quelques propositions de classification des protocoles d'interaction. Vu que la négociation est la forme d'interaction qui nécessite le plus de communication entre les agents, la majorité des classifications existantes dans la littérature concerne les protocoles de négociation.

2.6.1 Les protocoles compétitifs vs. coopératifs

P. Maes [GM98] identifie deux classes de protocoles. La première classe regroupe les protocoles distributifs qui se basent sur le principe du partage des gains entre les participants. Chacun des participants cherche à maximiser sa part de profit ; la réussite de l'acheteur signifie la défaite du vendeur, et vice versa. Dans la théorie de jeu, on les appelle des jeux à somme nulle. P. Maes les considère comme des protocoles compétitifs, car c'est l'esprit qui domine dans ce

genre de protocoles. Les protocoles de vente aux enchères sont considérés comme des protocoles compétitifs.

La deuxième classe regroupe les protocoles intégratifs, dont l'objectif de ses participants est d'essayer d'élargir l'espace d'entente, et de chercher de nouvelles solutions qui maximisent les profits mutuels des différents participants. Dans la théorie de jeu, on les appelle des jeux à somme non nulle, ils sont considérés comme des protocoles coopératifs en intégrant plusieurs paramètres de négociation.

2.6.2 Les protocoles unidirectionnels vs. bidirectionnels

Dans le protocole de négociation unidirectionnel, on se limite à l'acceptation ou le rejet d'une proposition. On trouve dans cette catégorie les protocoles de vente aux enchères et le protocole Contract Net.

Le protocole de négociation bidirectionnel, dont le scénario est plus complexe, se base sur un échange de messages (offre et contre-offre). Ce type de protocoles offre plus de flexibilité à la négociation soit en changeant la valeur d'un ou de plusieurs attributs négociables, via une contre-proposition, soit en changeant la structure de la proposition, en ajoutant par exemple un nouveaux attribut négociable. Le protocole Bargaining [SGBP03] est exemple de protocole bidirectionnel, qui utilise les performatives *propose* et *counter-propose*.

2.6.3 Autres classifications

Différentes autres classifications des protocoles d'interaction ont été proposées. Wurman et al. [WWE98] proposent une classification basée sur les critères suivants : 1) Simple ou double : un protocole est dit simple si le type des enchérisseurs est soit des agents acheteurs ou soit des agents vendeurs. Dans la double enchère on trouve des agents acheteurs et des agents vendeurs qui participent à la même enchère. 2) Privé ou public : dans le premier cas, les propositions soumises par les participants ne sont connues qu'après la clôture de l'enchère. Dans le deuxième cas, les propositions sont publiques, connues par tous les participants. 3) Conditionné ou inconditionné : un protocole est conditionné si les propositions sont régies par des règles, par exemple, le prix proposé doit être ascendant ou descendant.

London Classification est un modèle de classification des systèmes de négociation, défini à l'issue du workshop *Negotiations in Electronic Markets* [Dex00]. Il traite l'aspect participant, l'aspect produit, l'aspect décisionnel et l'aspect processus de négociation. Ce dernier aspect est défini par un ensemble d'attributs, parmi lesquels :

1. Le type de l'agent initiateur de la négociation qui peut être l'agent vendeur ou l'agent acheteur.
2. Le nombre de phases : Un protocole est dit à simple phase si les règles de négociations ne changent pas tout au long du processus de négociation, dans le cas contraire, on le considère comme un protocole multi-phases.
3. L'ordonnancement dans la négociation multi-attributs peut être en une seule étape si tous les attributs sont négociés en même temps ou multi-étapes si on négocie chaque attribut à part.
4. La publication des offres ou des propositions, si la proposition est entendue par tous les participants le protocole est dit public (ou *open cry* pour les protocoles de ventes aux enchères). Dans le cas contraire on l'appelle protocole privé ou *sealed bid*.
5. Le prix que paie le gagnant dans l'enchère dépend des règles du protocole utilisé : Dans les protocoles discriminantifs le gagnant paie le prix de la proposition qu'il a soumis (par exemple l'enchère anglaise). Dans les protocoles non discriminatifs le gagnant paie un prix plus bas que celui qu'il avait proposé. Par exemple, dans le protocole Vikrey le gagnant paye le prix de la deuxième proposition.
6. Le désistement d'un participant sur un engagement qu'il avait pris au cours d'une négociation peut être permis ou non par le protocole de négociation.

Une taxonomie des principaux protocoles de ventes aux enchères est proposée par Reck [Rec94] qui s'est basé sur les critères suivants : le nombre des offres et des propositions permises par le protocole, la publication des offres et des propositions (privée ou public), le nombre des produits négociés par transaction (un ou plusieurs), l'ordonnancement des transactions (périodique ou continue), les règles appliquées sur les propositions ou sur les offres (le prix est ascendant ou descendant). La figure 2.2 illustre l'arborescence des principaux protocoles de ventes aux d'enchères.

Mazouzi [MFSH02] classifie les protocoles d'interaction en quatre principales classes : les protocoles de coordination, les protocoles de coopération, les protocoles de négociation et les protocoles de vente aux enchères.

Sandholm [San99] identifie cinq types de protocoles de négociation : les systèmes de vote, les enchères, marchandage, systèmes contractuels et la formation de coalition.

Verrons [Ver04a] regroupe les différentes formes de négociation en plusieurs familles. La première famille est celle des systèmes de vote. Ces derniers permettent de choisir une solution parmi plusieurs. La deuxième famille concerne les enchères dont les principaux types sont l'enchère anglaise, l'enchère hollandaise et l'enchère Vikrey. La troisième famille regroupe les négociations se basant sur le protocole *Contract Net*. La dernière famille est celle des protocoles d'argumentation.

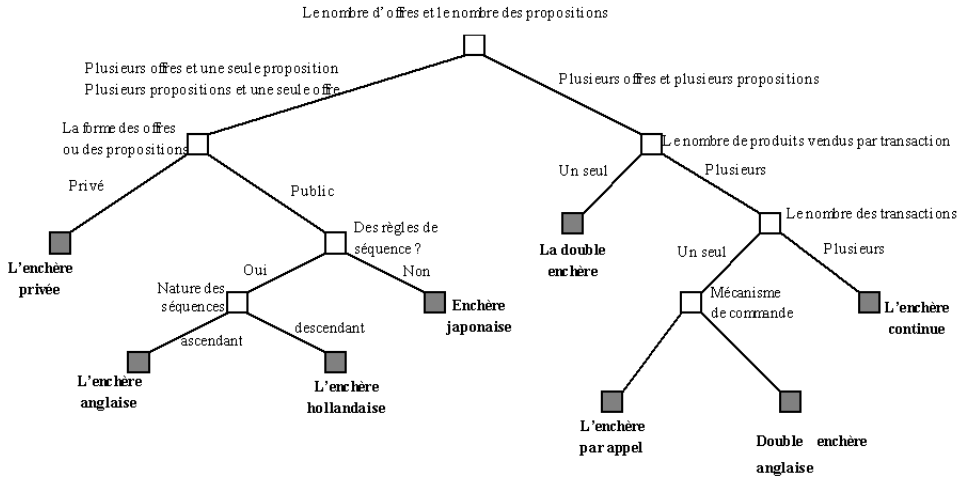


FIG. 2.2 – Taxonomie des protocoles d'enchère [Rec94]

Ainsi, il existe deux types de classification des protocoles d'interaction. Les classifications du première type utilisent des propriété descriptive des protocoles d'interaction comme critères de classification, tels que le nombre de participants, la nature des propositions et le nombre de phases. Par contre les classification du deuxième type (i.e. classification de Sandholm, de Mazouzi et de Verrons) utilisent un seul critère, celui de la situation d'interaction, que nous détaillons dans la section suivante.

2.7 Situations d'interaction

Selon Ferber [Fer95] situation d'interaction est un ensemble de comportements résultant du regroupement d'agents qui doivent agir pour satisfaire leurs objectifs en tenant compte des contraintes provenant des ressources plus ou moins limitées dont ils disposent et de leurs compétences individuelles. Ainsi, une situation d'interaction permet de décrire des types d'interactions en les reliant aux éléments qui la composent.

Cette notion de situation d'interaction permet de définir des catégories d'interaction abstraites indépendantes de leurs réalisations concrètes. Ferber identifie trois critères pour classifier les principales situations d'interaction :

- les objectifs ou intentions des agents,
- les relations qu'entretiennent envers les ressources qu'ils possèdent,
- les moyens (ou compétences) dont ils disposent pour parvenir à leurs buts.

Ces trois composantes principales de l'interaction ont permis de faire une première classification des situations d'interaction, comme le montre le tableau 2.2. Les agents sociaux composant un système multi-agents sont régulièrement en interaction, mais cette interaction peut se révéler conflictuelle : les buts des agents peuvent être différents et antagonistes. Ainsi, plusieurs formes d'interactions au sein d'un système multi-agents sont identifiées : la collaboration (addition simple des compétences des agents), la coopération (collaboration coordonnée), la compétition (les agents ont des buts antagonistes ou doivent se partager peu de ressources), l'encombrement (les agents se gênent les uns les autres, mais ne sont pas en compétition).

TAB. 2.2 – Une taxonomie des situations d'interaction [Fer95]

Buts	**Ressources**	**Compétences**	**Types de situation**
Compatibles	Suffisantes	Suffisantes	Indépendance
Compatibles	Suffisantes	Insuffisantes	Collaboration simple
Compatibles	Insuffisantes	Suffisantes	Encombrant
Compatibles	Insuffisantes	Insuffisantes	Collaboration coordonnée
Incompatibles	Suffisantes	Suffisantes	Compétition individuelle pure
Incompatibles	Suffisantes	Insuffisantes	Compétition collective pure
Incompatibles	Insuffisantes	Suffisantes	Conflits individuels pour des ressources
Incompatibles	Insuffisantes	Insuffisantes	Conflits collectifs pour des ressources

La situation d'*indépendance* ne pose aucun problème d'interaction du point de vue multi-agents et se résume à la simple juxtaposition des actions des agents pris indépendamment sans qu'il y ait effectivement d'interaction [Fer95]. Il s'agit d'une situation qui ne réclame aucune interaction, de point de vue communication. Nous rappelons que nous nous intéressons dans ce travail aux interactions basées sur la communication entre agents.

Dans la situation d'*encombrement* les agents se gênent mutuellement dans l'accomplissement de leurs tâches, cela est dû à l'insuffisance des ressources. Puisque les buts sont compatibles, les agents essayent de coordonner leurs actions.

Comme son nom l'indique, la situation de *collaboration coordonnée* représente une situation où les agents doivent collaborer pour combler leur manque de compétences.En plus de la collaboration, les agents doivent coordonner leurs actions à cause de l'insuffisance des ressources. C'est la plus complexe des situations de collaboration.

La situation de *compétition individuelle pure*, ne pose pas de problème sur les ressources ou les compétences des agents. Cela ne crée pas de problèmes spécifiques d'interaction liés à ce type de situation.

Dans la situation de *conflits individuels pour des ressources*, chaque agent essaye d'acquérir les ressources pour pouvoir atteindre ces buts. Cette situation pousse les agents à négocier l'accès aux ressources. On parle dans ce cas de protocoles de négociation.

Dans la situation de *compétition collective pure*, il faut créer des coalitions et des groupes d'agents ayant des objectifs compatibles, et ramener ainsi le système à une situation de *compétition individuelle pure*. L'interaction entre les agents d'un même groupe est *collaborative simple*. Nous constatons que cette situation est une composition de la situation de *compétition individuelle pure* et celle de *collaborative simple*. Le même raisonnement est vrai pour la situation de *conflits collectifs des ressources*, qui peut être considérée comme une combinaison de la *collaboration coordonnée* et de la situation des *conflits individuels pour des ressources*.

En résumé, nous retenons trois grandes classes de situations d'interaction : les situations de *coordination*, les situations de *coopération* (ou de collaboration), et les situations de *résolution de conflit*.

Après cette brève présentation des protocoles d'interaction nous étudions dans le reste de ce chapitre les travaux existants qui s'interessent à la représentation et l'implémentation des protocoles d'interaction.

2.8 Représentation des protocoles d'interaction

Nous distinguons deux catégories de travaux sur la représentation des protocoles d'interaction. Les travaux de la première catégorie s'interessent principalement à la conception, la description formelle et la validation des protocoles d'interaction. La description formelle décrit le protocole selon un formalisme afin d'éviter toute ambiguïté due à l'utilisation de la langue naturelle. La validation s'assure que le protocole dispose de certaines propriétés prédéfinies [Hug01]. Plusieurs travaux ont traité ces étapes de l'ingénierie des protocoles d'interaction [Maz01]. Dans ces travaux les protocoles d'interaction sont décrits dans un seul modèle, qui contient l'ensemble des messages échangés entre les agents.

La deuxième catégorie de travaux décrit séparément le comportement interactif de chacun des participants. C'est une vision locale du protocole d'interaction. Cette représentation facilite l'intégration des protocoles d'interaction dans le comportement des agents. Le but de ces travaux est de faciliter l'implémentation des protocoles d'interaction. C'est dans cette catégorie de travaux que nous situons notre travail sur les protocoles d'interaction. Dans cette section nous présentons les travaux qui ont traité le problème de la représentation réutilisable des protocoles d'interaction.

2.8.1 GeNCA

Verrons [Ver04a] propose le modèle général pour la négociation de contrats GeNCA (*Generic Negotiation of Contracts API*). L'idée est d'étudier les différentes formes de négociation et de proposer un protocole de négociation qui est une unification de ces différentes formes de négociation. Le protocole général de négociation proposé est paramétrable afin de faciliter la définition de nouveaux protocoles, sans demander à un utilisateur de lourdes charges de travail.

GeNCA est basé sur une approche en trois couches (voir figure 2.3) : communication, négociation et stratégie. Les trois couches sont indépendantes les unes des autres pour offrir plus de généricité. Selon Verrons, la façon dont les agents communiquent n'a pas d'influence sur leur façon de négocier. De même, les stratégies de négociation sont intrinsèquement liées à une application donnée et il est bien évident que la stratégie à utiliser lors d'enchères anglaises (où les prix augmente) et lors d'enchères hollandaises (où les prix diminuent) n'est pas la même. La couche de communication fournit le moyen de communication utilisé par les agents, tels que la communication fournie par les plates-formes multi-agents. La couche de négociation forme le coeur du modèle GeNCA. Elle contient le protocole de négociation utilisé, ainsi que la gestion des différentes négociations [Ver04b].

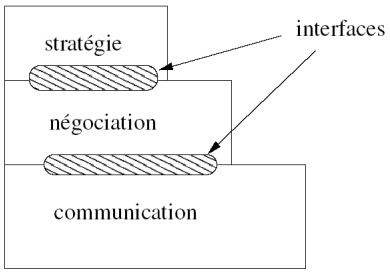


FIG. 2.3 – Les trois niveaux du modèle GeNCA [Ver04b]

Le protocole de négociation de GeNCA ressemble dans son activité au protocole *FIPA Iterated Contract Net* mais il définit en plus les notions de clauses minimales suffisantes pour valider un contrat et le nombre maximal de re-négociation au-delà duquel le contrat est non validé. GeNCA est un modèle de négociation générique qui apporte de nouvelles caractéristiques qui n'existent pas dans d'autres modèles, telles que la renégociation automatique produit lorsqu'un agent se rétracte d'un contrat qu'il a signé, la gestion des négociations simultanées portent sur des ressources communes ou disjointes, etc.

Le modèle GeNCA a permis de développer un framework qui facilite la définition de la négociation entre les agents. En effet, un certain nombre d'applications ont été développées en

utilisant ce framework : un système de vente aux enchères, un système de prise de rendez-vous et un jeu de négociation de ressources appelé JNego [MV04].

A la différence de ce travail, nous pensons qu'une bibliothèque de protocoles componentionnels est plus facile à maintenir. Il est plus facile d'ajouter un nouveau protocole d'interaction, avec une éventuelle extension de l'ontologie d'interaction, que d'essayer d'adapter un protocole général de négociation pour intégrer une nouvelle forme de négociation avec ses éventuelles paramètres d'entrée. Les agents ne peuvent pas endosser ce protocole général de négociation, car ce dernier peut faire l'objet de mises à jour.

2.8.2 Le framework de Bartolini

Bartolini et al. [BPJ02] ont développé un framework générique pour la négociation automatique dédié aux mécanismes de marché. Ce framework comporte un protocole général de négociation qui se paramètre par des règles de négociation. En choisissant un ensemble de règles, différents mécanismes de négociation peuvent être réalisés. La négociation se déroule dans une scène de négociation qui est une abstraction du système de messages utilisés par les participants de la négociation pour communiquer. Après avoir été admis à la négociation, un participant peut accéder à la scène de négociation. Chaque participant peut envoyer des propositions via un message destiné à l'hôte de négociation. Ce dernier se charge de la transmission du message aux participants concernés.

Ce framework de négociation se base sur : (i) un protocole général de négociation, (ii) une taxonomie des règles de négociation, (iii) un langage pour définir les règles de négociation, et (iv) un langage pour exprimer les propositions de négociation. Il y considère deux rôles principaux dans la négociation : le participant et le hôte. Les participants sont ceux qui veulent aboutir à un accord. Un participant peut poster des propositions selon les règles fournies par l'hôte de négociation. L'hôte est chargé de créer et de faire respecter les règles gouvernant la participation, l'exécution, la résolution et la terminaison de la négociation. Il peut ou non être également un participant. Il a les sous-rôles suivants :

- *Gatekeeper* : fait respecter la politique d'admission à la négociation,
- *Proposal validator* : assure qu'une proposition est conforme au template de négociation,
- *Protocol enforcer* : assure que les propositions des participants sont postées et enlevées selon les règles de négociation,
- *Agreement maker* : assure que les accords sont formés selon les règles,
- *Information updater* : notifie les participants de l'état courant de la négociation, selon les règles de visibilité et d'affichage,
- *Negotiation terminator* : déclare la fin de la négociation en se basant sur la règle de terminaison.

Ce framework a été implémenté en utilisant la plate-forme Jade, les agents communiquent par envoi de messages *FIPA-ACL* et Jess [] a été utilisé pour le traitement des règles. Une première critique pour ce framework est l'utilisation du Jess qui est un système assez lourd pour l'interprétation des règles. Similaire au protocole *GeNCA*, ce framework propose une représentation générale des protocoles de négociation. Il a en effet les mêmes caractéristiques et inconvénients que framework GeNCA.

2.8.3 Le travail de Jouvin

Pour faciliter l'implémentation des protocoles d'interaction et leur adaptation dynamique, Jouvin et al. [Jou00], proposent un mécanisme de délégation de conversations entre agents. L'approche proposée consiste à standardiser sous forme de protocole une délégation dynamique généralisée, et utiliser ce mécanisme de délégation comme "liant" pour résoudre dynamiquement des problèmes d'incompatibilité de protocole, des problèmes d'optimisation de protocole, ou encore masquer partiellement la complexité et la spécificité induite par le suivi d'un protocole complexe.

Pour gérer une conversation cible, dont l'agent n'implémente pas le protocole, il va déléguer dynamiquement son rôle dans cette conversation à un agent délégué (proxy) [Jou03], voir figure 2.4. Vis-à-vis des autres participants, le proxy prendra la place de l'agent qu'il représente, et prendra donc en charge son rôle pour toute la durée de la conversation.

Cet acte de délégation est en lui même une conversation régie par un protocole d'interaction. La conversation cible (déléguée), et le dialogue entre le *proxy* et l'initiateur de la délégation, sont deux sous-conversations imbriquées dans l'acte de délégation. Le dialogue entre le proxy et l'initiateur est spécifique au protocole de la conversation cible et au type de la délégation envisagé, personnifié par le *proxy*.

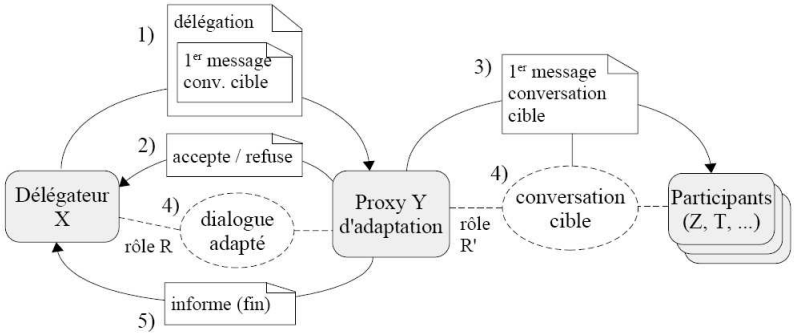


FIG. 2.4 – L'approche de délégation de conversation

Comparée à une approche componentielle, cette approche fait sortir le gestionnaire des conversations de l'agent, pour en faire un agent à part entière, le proxy. Ainsi, les auteurs ont proposé une bibliothèque d'agents qui facilite l'implémentation de différentes formes de négociation dérivée du protocole *Contract Net*.

L'avantage de cette approche est la dynamique dans la connexion entre l'agent et le proxy. Le proxy est un agent donc il est indépendant de l'environnement d'implémentation. De plus, la délégation permet de réduire la complexité communicationnelle de la conversation cible, prise en charge par le proxy, facilitant ainsi le développement des agents conversationnels.

Par contre, cette approche est coûteuse de point de vue temps d'interaction, due à l'ajout d'une conversation supplémentaire, celle de la délégation. De plus, pour adapter un protocole d'interaction donnée, l'agent délégateur doit avoir une implémentation d'un protocole de délégation compatible avec le protocole à adapter, ce qui crée une contrainte supplémentaire pour la mise en oeuvre de conversation cible.

2.8.4 La démarche de Freire et Botelho

Freire et Botelho [FB03] proposent une démarche qui permet aux agents d'exécuter des protocoles d'interaction. Les protocoles d'interaction sont représentés en XML, en se basant sur sa description graphique en AUML (*Agent Unified Modelling Language*) [HOB04]. Ce dernier est aussi utilisé pour la spécification des protocoles d'interaction de FIPA [FIP02b]. Ensuite, chaque diagramme AUML est transformé en un ensemble de fichiers XML. Chaque rôle du protocole est ainsi décrit par un fichier XML. Le schéma XML, utilisé pour définir les protocoles d'interaction, contient le nom du protocole et et une spécification propre à chaque rôle d'interaction. Ce dernier correspond à un ensemble de couples (condition, action). La figure 2.3 montre le premier couple condition-action correspondant à la spécification du rôle participant du protocole *FIPA-request*. La condition contient le nom du performatif *FIPA-ACL* du message à recevoir. Les actions indiquent le nom des performatifs des messages que le rôle peut envoyer.

Quand l'agent reçoit un message contenant la spécification d'un protocole en XML, il la convertit en un objet interne Java, en utilisant l'outil Castor. Ensuite l'agent traduit cet objet en un ensemble de règles de production, utilisées pour contrôler le comportement de l'agent conformément à la spécification du protocole. Ces règles représentant les aspects opérationnels du protocole interagissent avec les processus de décision internes de l'agent via une interface. Pour éviter des confusions sur les actions décisionnelles de l'agent, les auteurs proposent d'indiquer l'état du protocole au niveau des règles. Chaque action décisionnelle sera associée à un état particulier du protocole.

TAB. 2.3 – (a) Le schéma XML des protocoles (b) Une partie de la spécification du rôle participant du protocole *FIPA-Request* [FB03]

```
<schema>
  <complexType name = "Protocol">
    <all>
      <element name = "name" type = "string" />
      <element name = "role_spec" type = "ProtocolRoleSpec"
           minOccurs = "2" maxOccurs = "unbounded"/>
    </all>
  </complexType>
  <complexType name = "ProtocolRoleSpec">
    <all>
      <element name = "role" type = "string"/>
      <element name = "condition_action"
                type = "ConditionAction"
                minOccurs = "1" maxOccurs = "unbounded"/>
    </all>
  </complexType>
   <complexType name = "ConditionAction">
      <sequence>
        <element name = "condition" type = "Propositon"/>
        <element name = "action" type = "ActionTerm"/>
      </sequence>
   </complexType>
</schema>
```

```
<ConditionAction>
  <condition>
    <AtomicProposition>
      <received> request </received>
    </AtomicProposition>
  </condition>
  <action>
    <ActionAlternative>
      <SimpleAction>
        <ActionName> not-understood </ActionName>
      </SimpleAction>
      <SimpleAction>
        <ActionName> refuse </ActionName>
      </SimpleAction>
      <ActionSequence>
        <SimpleAction>
          <ActionName> agree </ActionName>
        </SimpleAction>
        <SimpleAction>
          <ActionName>assert</ActionName>
          <argument> agreed_state </argument>
        </SimpleAction>
      </ActionSequence>
    </ActionAlternative>
  </action>
</ConditionAction>
```

En terme de règles de production, les conditions des règles représentant des choix multiples doivent indiquer l'état du protocole aux quels ils sont associés. La règle de la figure 2.5 veut dire que si le protocole utilisé est *FIPA-Request*, si ce protocole est à l'état initial, et si le message reçu est un *Request*, alors l'agent doit choisir entre trois réponses possibles : *Agree*, *Refuse* et *Not-understood*. Quand le participant décide de choisir une action, il doit spécifier tous les paramètres de l'action. Le langage *Jess* est utilisé pour implémenter les règles de production.

Pour garder une représentation abstraite des protocoles d'interaction, les auteurs définissent une interface entre les processus décisionnelles de l'agent et le protocole d'interaction. Les deux principaux prédicats de cette interface sont `alternative_actions(ProtocoleInstanceID, ProtocoleState, AlternativeActions)` et `choose_alternative(ProtocoleInstanceID, ProtocoleState, Action)`. Le premier prédicat permet au processeur du protocole d'informer les processus décisionnels de l'agent du choix possible d'actions. Le deuxième prédicat est utilisé par les processus décisionnels de l'agent pour informer le processeur du protocole de l'action choisie.

Dans cette approche, les agents sont sensés connaître à l'avance la sémantique des messages *FIPA-ACL* pour pouvoir choisir quel est le message à envoyer, ce qui pose un problème de dépendance entre les actions internes de l'agent et la sémantique des messages. Par ailleurs,

```
If    protocol_instance(fipa-request,
      ?ProtocolInstanceID) and
      protocol_state(?ProtocolInstanceID,
      initial_state) and
      received(?ProtocolInstanceID,
      request(?Initr, ?Particip, ?Action,
      ?ConvID)) and not
      alternative_actions(?ProtocolInstanceID
      , initial_state, _)
Then
      assert(alternative_actions(?ProtocolIns
      tanceID, initial_state,
      [agree(?Particip, ?Initr, (?Action,
      ?Condition), ?ConvID),
      not-undertsood(?Particip, ?Initr,
      (?Action, ?Reason1), ?ConvID),
      refuse(?Particip, ?Initr, (?Action,
      ?Reason2), ?ConvID)]) )
```

FIG. 2.5 – Une règle de production du protocole *FIPA-Request* [FB03]

cette approche n'est pas bien applicable à tous les protocoles, par exemple le protocole d'enchère anglaise. La première action de choix alternatif du rôle participant de l'enchère anglaise est (*propose*, *not_ undrstand*). Par contre, le participant peut s'abstenir d'enchérir et se contenter de regarder l'évolution de l'enchère. La transformation de la spécification AUML en format XML, proposée par les auteurs, oblige le participant à faire une proposition à chaque tour.

2.8.5 Le travail de Quenum

Quenum et al. [QSA03], proposent un mécanisme automatique pour la reconfiguration des protocoles d'interaction. Au lieu d'indiquer explicitement le protocole et le rôle pour l'ensemble des agents, les auteurs suggèrent que c'est le développeur donne la description des tâches collaboratives de l'agent dans le code source de l'agent initiateur. Le but est de libérer le développeur de la lourde tâche de codage du protocole. Le développeur doit indiquer le code de l'agent initiateur correspondant à la description de la tâche collaborative à exécuter.

L'approche proposée [Que05], est basée sur un appariement entre les actions du rôle et les méthodes de l'agent. Ils comparent les types des paramètres d'entrée et de sortie des méthodes d'agent à la définition des actions de rôle. Les types de paramètres correspondent aux types prédéfinis des langages de programmation. Or, il est difficile de reconnaître avec cet appariement, d'une manière automatique, la sémantique exacte des méthodes de l'agent.

2.8.6 Le travail de De Silva

De Silva et al. [dSWL] s'intéressent à la réutilisation des protocoles d'interaction dans le cadre des systèmes ouverts. Les auteurs donnent une représentation des protocoles d'interaction basée sur une extension des réseaux de Petri. Ils introduisent ainsi trois concepts *Recv*, *Send*, *Action* et *Pred* au niveau des transitions et des noeuds du RdP. *Recv* et *Send* spécifient respectivement la réception et l'envoi de message. *Action* décrit la fonction à invoquer et les éventuels paramètres.

La communication dans un système ouvert doit vérifier certaines propriétés comme la protection des informations privées des agents et la sécurité dans l'exécution des protocoles d'interaction. Ils proposent d'ajouter des pré-conditions et post-conditions (i.e. *Pred*) pour assurer la bonne exécution des actions du protocole. Ils donnent l'exemple de l'action de payement qui doit être sécurisée en ajoutant une pré-condition qui vérifie que l'acheteur a bien reçu le produit avant d'entamer l'action de payement.

2.8.7 Synthèse

Le but des travaux que nous avons présenté dans cette section est de faciliter l'utilisation des protocoles d'interaction, sans demander un grand effort au développeur. Les solutions proposées peuvent être classées en trois approches. La première approche préconise un protocole général pour l'interaction. Ce protocole regroupe des propriétés et des comportements communs à plusieurs types de protocoles d'interaction. Ce protocole est spécialisable par un ensemble de paramètres, comme les travaux de Verrons et al. [Ver04a], et de Bartolini et al. [BPJ02]).

La deuxième approche consiste à représenter chaque protocole d'interaction comme un composant paramétrable. C'est la cas des travaux [FB03], [EC04], [QSA03] et [dSWL03]. Notre proposition de représentation des protocoles d'interaction s'inscrit dans cette deuxième approche. La principale limité de ces approches est la réutilisation statique des protocoles d'interaction.

La troisième approche préconise un mécanisme de délégation de la tâche d'interaction. La délégation garantie la sécurité du système et de son ouverture. Elle facilite la tâche d'interaction de l'agent. Elle permet d'avoir un système adaptable et robuste dans un environnement dynamique. Par contre, elle nécessite une interaction supplémentaire, entre l'agent délégateur et l'agent délégué. De plus, l'agent délégateur est contraint d'avoir une implémentation du protocole de délégation propre au protocole à adapter.

Dans cette section, nous avons analysé les différentes solutions proposées pour l'implémentation des protocoles d'interaction et leur réutilisation.

Pour comparer les différents travaux nous avons défini quatre propriétés :

- le(s) type(s) des protocoles supportés : négociation, coordination, interaction, etc.
- le type de la solution proposée : nous avons identifié trois approches possibles pour la mise en oeuvre d'un mécanisme pour la réutilisation des protocoles d'interaction, par unification, par délégation et par composant.
- le type de la réutilisation : nous distinguons la réutilisation statique et la réutilisation dynamique. Cette terminologie trouve son origine dans les travaux sur les composants logiciels. On parle de réutilisation statique quand le concepteur assemble au moment de la conception les composants. Cela consiste à relier les bornes d'entrées et de sorties des différents composants les unes aux autres. Par contre, l'assemblage ou le découplage des composants dans la réutilisation dynamique se fait au cours de l'execution du système, dans notre cas c'est l'agent. Dans le contexte de l'interaction, la réutilisation concerne l'ajout, la substitution et la suppression de protocoles d'interaction.
- la gestion des interactions : il existe le mode de gestion séquentielle, avec lequel l'agent ne peut participer qu'à une seule interaction à la fois, et la gestion simultanée des interactions, dans ce cas l'agent peut jouer plusieurs rôles en même temps. Avec ce deuxième mode, l'agent doit gérer les accès concurrents des rôles aux ressources partagées de l'agent.

Le tableau 2.4 présente une synthèse des travaux étudiés. Il montre qu'aucune solution n'a proposé une implémentation d'agents interactifs avec réutilisation dynamique des protocoles d'interaction, et une gestion explicite de ces derniers.

TAB. 2.4 – Synthèse des travaux sur l'implémentation des protocoles d'interaction. Le ? signifie qu'aucune donnée ne nous permet d'affirmer ou d'infirmer la propriété énoncée.

	Type protocole	Type approche	Type réutilisation des PI	Gestion des interactions
GeNCA	Négociation	Modèle général paramétrable	Statique	Séquentielle et Simultanée
Bartolini	Négociation	Modèle général à base de règles	Statique	Séquentielle
Jouvin	Interaction	Délégation	non	Séquentielle et Simultanée
Quenum	Interaction	Approche par appariement	Statique	Non traitée
De Silva	Interaction	Approche par appariement	Statique	?
Freire	Interaction	Approche par appariement	Statique	?
Ehrler	Interaction	Approche par appariement	Statique	?

Notre objectif est de proposer une solution pour la réutilisation et la gestion dynamique des protocoles d'interaction. Cette solution est décrite dans le chapitre 4.

2.9 Implémentation des protocoles d'interaction

Il existe actuellement plusieurs plates-formes pour le développement des systèmes multi-agents, telles que Madkit [Gut01], Zeus [NNLC99], FIPA-OS [PBH00], JACK [Gro04] et JADE [BCTR04], etc. Cependant, ces plates-formes ne proposent pas une solution faciliter l'utilisation des protocoles d'interaction, à l'exception de la plate-forme JADE. Nous nous sommes ainsi restreint dans notre étude aux deux plates-formes JACK [Gro04] et JADE [BCTR04]. A notre connaissance JADE est la seule plate-forme multi-agents qui propose une bibliothèque de protocoles d'interaction. Bien que JACK ne propose pas une implémentation des protocoles d'interaction, nous avons voulu l'étudier pour deux raisons : montrer le grand effort nécessaire pour implémenter un agent interactif à travers avec une solution proposée par Pasquier pour l'utilisation des protocoles, comme une extension de la plate-forme JACK.

2.9.1 JACK

JACK [Gro04] est un environnement de développement orienté agent conçu comme une extension de langage de programmation Java. Les agents dans JACK sont considérés comme des composants logiciels autonomes qui ont des buts explicites à accomplir. Pour décrire comment il faut procéder pour atteindre ces buts, les agents sont programmé avec un ensemble de plans (intensions).

Chaque plan décrit comment atteindre le but sous différentes circonstances. Mis en action, l'agent poursuit ces buts (désires), en adoptant des plans appropriés (intentions) en se basant sur les données (croyances) de l'état de l'environnement. Pour supporter la programmation des agents BDI (Belief Desire Intention), JACK propose un langage se basant sur cinq principaux concepts : l'agent, les compétences, les relations de base de données, les événements et les plans.

Dans JACK, le comportement d'un agent est décrit par des plans dont l'exécution est conditionnée par l'occurrence d'événements. Chaque plan peut traiter un certain type d'événements décrit dans la clause *Handle* du plan. Actuellement, il n'existe pas d'API dans JACK pour le développement et l'utilisation des protocoles d'interaction.

Il n'existe pas une implémentation proprement dite des protocoles d'interaction dans JACK. Les protocoles d'interaction sont implémentés sous forme de plans. L'activité du protocole est

décrite d'une manière procédurale dans la méthode body() du plan. Cette méthode décrit les actions exécutées par l'agent suite à la réception des messages.

Les messages échangés entre les agents, et plus précisément le nom des performatifs sont implémentés comme une extension de MessageEvent. Dans JACK, les langages de communication tels que FIPA-ACL ou KQML ne sont pas utilisé. Ceci pose certaines difficultés pour exprimer la sémantique des messages entre les agents et les informations qu'ils comportent.

Pour utiliser le langage FIPA-ACL, Pasquier [Pas01] propose d'ajouter une classe Java qui implémente la couche syntaxique de FIPA-ACL. D'utilisation simple, cette classe (nommée FipaMessage) permet de créer des messages *FIPA*, d'indiquer de quel acte de langage il s'agit (*Inform*, *Request*, *Cancel*, *Agree...*) et de renseigner les différents champs du message à l'aide des méthodes prévues à cet effet (setEmeteur, setRecepteur, setLangage, setOntologie, setContenu...).

Les actions qui traitent les messages envoyés et reçus sont implémentées comme extension de plan et ils définissent le nom du ou des performatifs dans le *Handle*. Les données utilisées au cours de l'interaction, tel que le *"Time-Out"*, sont considérées comme des croyances. Ils font partie des données de la base de données de l'agent.

Castro et al. [CKM01] donnent un exemple d'application développé sous JACK. Dans cette application les deux agents *ShoppingCart* et *Customer* utilisent le protocole *Contract Net*. D'après les auteurs, le code de base des agents ainsi que leur comportement d'interaction peut être généré à partir d'une spécification AUML du protocole d'interaction utilisé.

JACK n'a pas ajout de nouveaux concepts relatifs à l'interaction, tels que le rôle et le protocole d'interaction. De cet fait, l'implémentation de l'aspect interactif des agents peut être considéré comme *had-doc*. Cela engendre des problèmes de maintenance des agents, en voulant par exemple changer de protocole d'interaction, ce qui complique la tâche du développeur. Une autre limite de cette approche, est la difficulté de réutiliser le code d'implémentation d'un protocole d'interaction dans différentes applications.

2.9.2 JADE

JADE [BCTR04] (Java Agent Development Framework), est un framework pour le développement logiciel qui vise de développer des systèmes multi-agents et des applications conformes aux standards de FIPA. Il inclue deux principaux produits : une plate-forme agent FIPA compliant et un package pour le développement des agents en Java. JADE est codé en Java et le développeur doit implémenter ces agents en Java pour pouvoir exploiter ce framework. Ce der-

nier offre un ensemble de packages, parmi lesquels le package `jade.proto` qui est un ensemble de classes qui modélisent un certain nombre de protocoles d'interaction standardisés (*Fipa-Request, Fipa-Query, Fipa-Contract-Net, Fipa-subscribe* et d'autres protocoles définie par *FIPA*). Cette API aide le programmeur à utiliser les protocoles d'interaction. Les messages sont décrits avec le langage *FIPA-ACL*. Le package `jade.lang.acl` contient des classes qui implémentent les performatifs de FIPA-ACL.

La classe `Agent` offre un ensemble de méthodes qui assurent la communication inter-agents avec envoie de messages en mode asynchrone. Pour chaque interaction, JADE distingue un rôle initiateur et un rôle participant. Chaque rôle d'interaction est décrit par un automate à états finis. Les actions qui traitent les messages reçus font appel à des méthodes abstraites de la classe du rôle, que le développeur doit surcharger pour pouvoir instancier le rôle. JADE propose des méthodes génériques qui peuvent être utilisées, par défaut, pour traiter les actions du rôle.

Cette modélisation ne facilite pas la tâche du développeur puisqu'il doit connaître le fonctionnement du protocole et réécrire toutes les méthodes du rôle. L'intervention nécessaire du concepteur lors de l'instanciation des protocoles d'interaction ne permet pas aux agents de pouvoir changer dynamiquement de rôles.

JADE présentent des solutions intéressantes pour résoudre certains problèmes liés à la représentation et l'implémentation des protocoles d'interaction. Néanmoins, aucune d'elles ne propose une implémentation des protocoles d'interaction qui peut être utilisée dynamiquement pour développer des agents interactifs, c'est ce que nous proposons dans *INAF* (voir le chapitre 5.2).

2.10 Conclusion

Nous avons présenté dans ce chapitre un état de l'art sur la représentation et l'implémentation des protocoles d'interaction. Et nous nous sommes arrivé à un constat qu'une représentation réutilisable des protocoles d'interaction est cruciale pour faciliter la tâche du développeur puisqu'il n'aura pas à redéfinir les services de base de l'agent interactif. De plus il aura à sa disposition une bibliothèque de protocoles d'interaction qu'il peut utiliser par simple instanciation de leurs paramètres.

D'un point de vue méthodologique, il existe aussi des problèmes liés la spécification des interactions des agents. Le chapitre suivant analyse les methodologies existantes afin d'identifier le(s) problème(s) lié(s) à l'ingénierie des systèmes multi-agents.

Chapitre 3

Ingénierie des systèmes multi-agents

3.1 Introduction

La technologie agent a suscité beaucoup d'attention ces dernières années, en conséquence, l'industrie commence à s'intéresser et à employer cette technologie. Malgré le développement de différentes théories d'agent, de langages et d'architectures d'agents, très peu de travaux se sont intéressés aux techniques de développement utilisant la technologie d'agent. De ce fait, un nouveau thème de recherche, celui de l'ingénierie logicielle orientée agent (i.e. *AOSE*[1]), s'est développé ces dernières années. L'objectif principal de l'ingénierie logicielle orientée agent, est de développer des méthodologies et des outils qui facilitent le développement et la maintenance des applications multi-agents [Tve01]. Nous trouvons dans le domaine de l'ingénierie logicielle, trois activités de recherche [CCZ05] : (i) le développement d'outils conceptuels (outils de notation ou de modélisation formelle), (ii) le développement des outils d'assistance au cours du processus du développement (les CASE tools) et (iii) les méthodologies multi-agents.

Nous nous intéressons dans ce chapitre aux méthodologies et aux plates-formes multi-agents. Le rôle de ces méthodologies multi-agents est d'assurer l'assistance durant toutes les phases du cycle de vie d'une application basée sur les agents. Dans cet état de l'art nous étudions les principales méthodologies existantes en multi-agents, et plus particulièrement celles qui intègrent la phase d'implémentation.

La section 3.2 présente et analyse cinq méthodologies : *Gaia*, *ROADMAP*, *PASSI*, *INGENIAS* et *Tropos*. Cette étude met l'accent sur la représentation du niveau conceptuel et son utilisation pour passer à une implémentation avec une plate-forme multi-agents. Nous étudions

[1] Agent Oriented Software Engineering

pour chaque méthodologie la démarche de développement et les outils qu'elle offre au cours de cette phase de transition entre la conception et l'implémentation. La section 3.3 concernent l'autres travaux portants principalement sur l'utilisation de la méta-modélisation dans l'ingénierie des systèmes multi-agents. Nous terminons ce chapitre par une synthèse sur les méthodologies multi-agents et la présentation du problème de la difficulté de passage à l'implémentation, rencontré dans la plupart des méthodologies existantes.

3.2 Méthodologies multi-agents

Une méthodologie est une démarche progressive qui commence par la définition des besoins préalables de l'utilisateur pour arriver à une implémentation d'un système capable de satisfaire les besoins initiaux.

Plusieurs travaux [IGG99, WC01] ont étudié et classifié selon plusieurs critères les méthodologies multi-agents, dans le but de les améliorer ou pour aller à une unification (ou une standardisation) de la démarche de développement des systèmes multi-agents. Dans [CCZ05], Cernuzzi et ses collègues se sont intéressés aux modèles de développement adoptés par les méthodologies multi-agents. Ils ont classifié quelques méthodologies selon les quatre modèles de développement les plus connus en génie logiciel : modèle en cascade, modèle par incrémentation, modèle en spirale et modèle par transformation, voir la figure 3.1. Ils ont donné aussi pour chaque méthodologie les phases de développement qu'elle intègre dans sa démarche.

Sur la base de cette classification, nous avons choisi d'étudier les méthodologies *PASSI*, *INGENIAS* et *Tropos*, qui vont jusqu'à l'implémentation du système. Nous avons voulu ajouter à cet état de l'art la méthodologie *GAIA*, bien qu'elle ne couvre pas la phase d'implémentation. Ce choix est justifié par le fait que *GAIA* est l'une des premières méthodologies en multi-agents et parmi les méthodologies les plus citées dans la littérature. Nous étudions aussi la méthodologie *ROADMAP* qui se base sur *GAIA*. *ROADMAP* essaie de remédier à certaines limites dans *GAIA*, comme la modélisation des applications définies dans un environnement ouvert.

Le but de cette section est d'étudier le passage de la conception à l'implémentation. Nous donnons pour chaque méthodologie : (i) le ou les méta-modèles qui regroupent les concepts qu'elle représente, (ii) son processus de développement et (iii) une description des éventuels techniques et outils, qu'elle propose pour la phase d'implémentation. Nous commençons par décrire la méthodologie *AALAADIN* [FG98], qui est le premier travail qui propose un méta-modèle pour l'organisationnel en multi-agents.

Phases ⟶ **Process Model** and **Methodology** ↓	Requirements Elicitation	Requirements Analysis	Design	Coding and Implementation	Verification & Testing	Deployment
Waterfall Like						
Gaia		X	X			
Roadmap	X (partially)	X	X			
Prometheus	X (partially)	X	X	X	X	
MaSE	X (partially)	X	X	X	X(partially)	
AOR		X	X	X		
Evolutionary and Incremental						
OPM/MAS	X	X	X			X
MASSIVE		X	X	X	X	X
Ingenias		X	X	X		
Tropos	X	X	X	X		
PASSI andAgile PASSI		X	X	X	X	X
Transformation						
DESIRE	X	X	X	X	X(partially)	
Spiral						
MAS-CommonKADs	X	X	X	X	X(partially)	X

FIG. 3.1 – Une classification des méthodologies [CCZ05]

3.2.1 *AALAADIN*

La méthodologie AALAADIN [FG98] s'interesse à la conception et à la réalisation des systèmes multi-agents dans une perspective organisationnelle.

3.2.1.1 Méta-modèle

AALAADIN a introduit le premier méta-modèle multi-agents qui est fondé sur les notions d'*Agent*, de *Groupe* et de *Rôle* (*AGR*). Un système multi-agents est vu comme un ensemble de groupes d'agents interconnectés par l'intermédiaire de rôles appartenant à plusieurs groupes, voir figure 3.2.

Un *agent* est une entité autonome qui joue des rôles dans des groupes. Un agent peut avoir plusieurs rôles et être membre de plusieurs groupes. *AALAADIN* ne se pose aucune contrainte ou pré-requis sur l'architecture interne de l'agent. Il ne se suppose pas de formalisme ou modèle pour décrire le comportement de l'agent.

Un *rôle* est une représentation abstraite d'une fonction, d'un service ou d'une identification d'un agent au sein d'un groupe particulier. Un rôle peut être joué par plusieurs agents. Le rôle

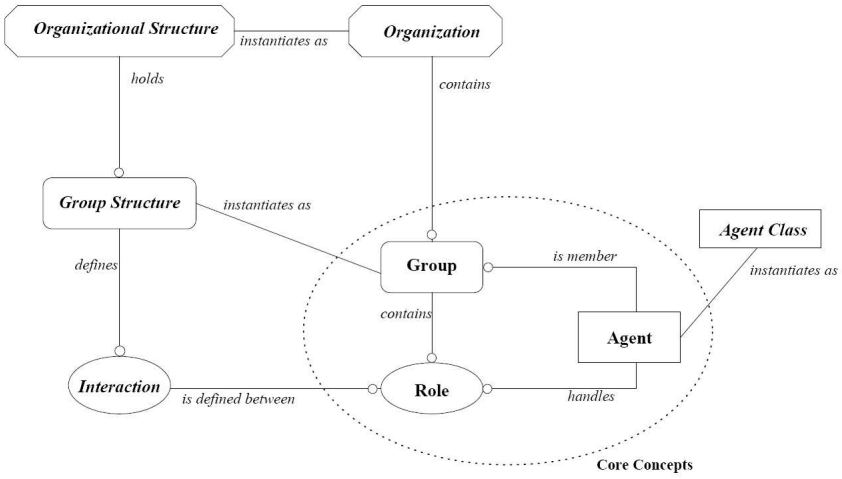


FIG. 3.2 – Le méta-modèle de la méthodologie *AALAADIN*

est une représentation abstraire de la fonction de l'agent. Les rôles sont locaux aux groupes. La tenue d'un rôle dans un groupe doit être par l'agent et n'est pas forcement accordée.

Un *groupe* est un ensemble d'agent partageant une caractéristique. Deux agents ne peuvent communiquer que s'ils appartiennent à un même groupe.

3.2.1.2 Démarche de développement

A partir des concepts AGR, une démarche méthodologique est définie pour le cadre organisationnel des applications multi-agents. Cette démarche définit l'ensemble des rôles possibles, spécifie les interactions, et décrit les structures abstraites de groupe et d'organisation. Ainsi, de nouveaux concepts à savoir les structures de groupe et les structures d'organisation sont ajoutées au niveau conceptuel. Ces structures sont des représentations abstraites des modèles de groupe et d'organisation, qu'on peut considérer comme des patterns. Toute instance de ces structures peut être une représentation partielle ou totale de la structure adoptée.

Une structure de groupe est définie comme un tuple : $S=<R,\ G,\ L>$. R est l'ensemble des identifiant des rôles. G est le graphe des interactions. L est le langage d'interaction. Une structure organisationnelle est définie comme un ensemble de structures de groupe, définissant un modèle organisationnel multi-agents.

Le modèle organisationnel peut être vu comme une spécification globale du problème initial. La structure organisationnelle est définie par le couple $O=<S,\ Rep>$, où S est un ensemble de structures de groupes et Rep est le graphe des représentants. Chaque arc relié à deux rôles

appartenant chacun à une structure de groupe, il décrit leur interaction. Un représentant entre deux structures de groupes est un agent qui possède obligatoirement un rôle dans chacune des structures.

3.2.1.3 Passage à l'implémentation

Ferber et al. [FGM03] ont développé à partir des concepts AGR une méthodologie centrée sur l'organisation, qui s'exprime de manière complémentaire aux autres méthodologies centrée agents. Par ailleurs, Ferber et al. [GF00] proposent aussi la plate-forme *MADKIT* qui supporte cette architecture organisationnelle et sa dynamique. L'organisation d'un système multi-agents ayant une architecture *AGR* peut être facilement généré sous *MADKIT*. La philosophie de *MADKIT* est d'offrir les briques de base pour faciliter l'implémentation d'un système multi-agents.

En résumé, *AALAADIN* propose un méta-modèle assez générique de l'aspect organisationnel, qui a été repris par plusieurs méthodologies multi-agents, tels que *GAIA*, *ROADMAP*, *PASSI*. Mais elle est insuffisant à elle seule de pouvoir représenter tous les aspects multi-agents (agent, interaction, environnement et organisation).

3.2.2 ADELFE (Atelier de DEveloppement de Logiciel à Fonctionnalité Emergente)

ADELFE est une méthodologie permet de modéliser des systèmes ayant des environnements imprédictibles. Cette méthodologie repose sur une théorie adaptative des systèmes multi-agents. Elle propose un processus, une notation et des outils.

3.2.2.1 méta-modèle

Les agents modélisés par *ADELFE* sont coopératifs. Ils ignorent le but global du système mais ils cherchent à réaliser leurs buts locaux tout en essayant d'être coopératifs entre eux. Le cycle de vie de l'agent se base sur une perception, puis une décision et enfin une action.

La structure de l'agent coopératif (Cooperative Agent) est basée sur des connaissances et des croyances obtenues par la perception de l'environnement, y compris les autres agents. Une *perception* est une information transmise soit par l'environnement ou par un agent. L'agent possède des *caractéristiques* qui sont ses propriétés intrinsèques ou physiques. Une caractéristique peut être une chose que l'agent peut réaliser pour modifier ou mettre à jour ses propriétés.

Chaque agent possède des *représentations* du système qui correspond à des croyances au sujet des autres agents, de l'*environnement* physique qui l'entoure et de lui même. Ces représentations permettent à l'agent de définir son comportement.

Un agent peut communiquer avec les autres agents ou avec son environnement. La *communication* directe avec les agents fait appel aux patterns des protocoles d'interaction.

Pour pouvoir agir sur son environnement, l'agent possède un ensemble de compétences et d'aptitudes. L'agent possède des *compétences* qui sont des connaissances particulières lui permettant de réaliser une fonction. Les *aptitudes* de l'agent montrent la capacité de l'agent à pouvoir raisonner sur les connaissances et les croyances qu'il possède. Une aptitude peut être représentée par un moteur d'inférences basé sur des règles.

L'activité de l'agent se base sur un ensemble de règles de coopération qui lui permettent d'identifier et de résoudre *des situations non coopératives* (NCS). *ADELFE* identifie une taxonomie des NCS qui dépendent du contexte d'application.

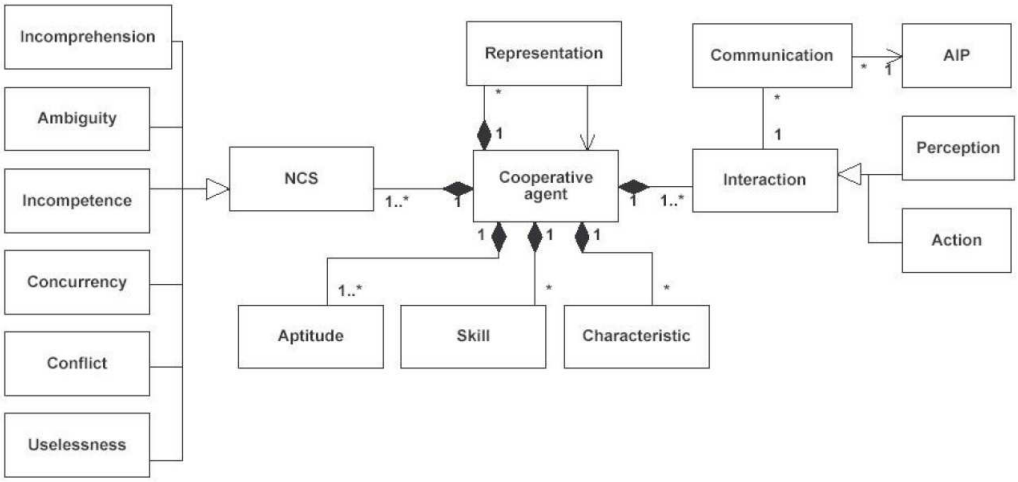


FIG. 3.3 – Le méta-modèle de ADELFE

3.2.2.2 Démarche de développement

Le processus de développement d'*ADELFE* est basé sur le processus *RUP* (*Rational Unified Process*) cite, un processus classique pour les méthodologies orientées objet. Pour tenir compte des spécificités de l'agent, certaines étapes sont ajoutées à *RUP*. Actuellement, *ADELFE* couvre les phases de définition des besoins jusqu'à la conception (voir figure 3.4).

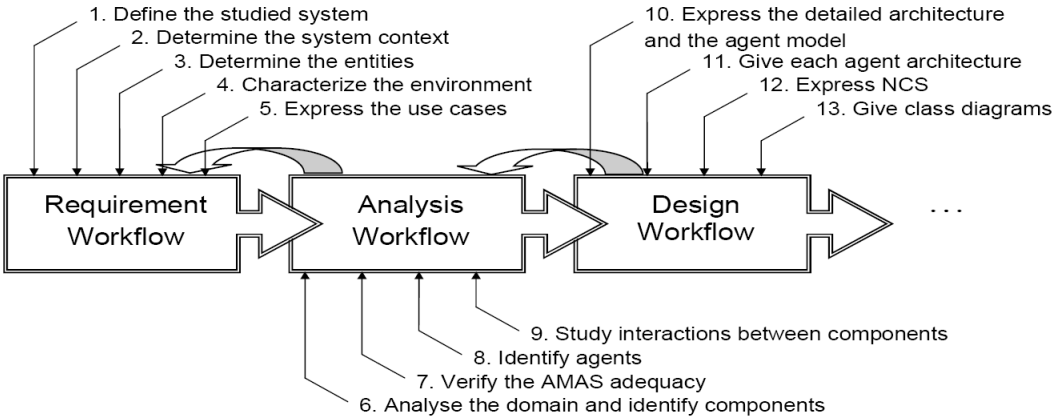


FIG. 3.4 – Le processus de développement de *ADELFE* [PG04]

3.2.2.3 Passage à l'implémentation

ADELFE utilise l'outil *OpenTool* [PG04] qui est un logiciel développé par TNI (http ://www.tni.fr). Il est CASE tool, tels que Rational Rose, qui permet de décrire des modèles du systèmes multi-agents. *OpenTool* est outil graphique qui supporte la notation UML et AUML pour la représentation des protocoles d'interaction. Cet outil propose des interfaces interactives qui guident les concepteurs durant le processus de développement.

Dans ADELFE, nous remarquons une absence de l'aspect organisationnel (structure organisationnelle, règles d'organisation, rôle) ce qui oblige le concepteur à assurer la synchronisation entre les différents agents suivant leurs comportements et leurs compétences. Le concepteur doit exprimer sa structure organisationnelle sous forme de règles de coopérations propres à chaque agent. De plus, l'absence du concept rôle offre moins de flexibilité dans la définition du comportement de l'agent.

3.2.3 GAIA

La méthodologie GAIA [WJK00] [ZJW03] utilise une approche centrée sur l'organisation pour analyser et concevoir un système multi-agents. Elle reprend et développe des concepts d'*AALAADIN*.

3.2.3.1 Méta-modèle

La figure 3.5 décrit le méta-modèle qui regroupe les concepts utilisés par GAIA pour décrire un système multi-agents. Les concepts utilisés au cours de la phase d'analyse sont : Rôle, Permission, Responsabilité, Protocole, Activité, Propriété d'animation et Propriété de sécurité, ils sont appelés concepts abstraits. Au niveau conceptuel, GAIA utilise des concepts concrets : Agent, Service et Accointance.

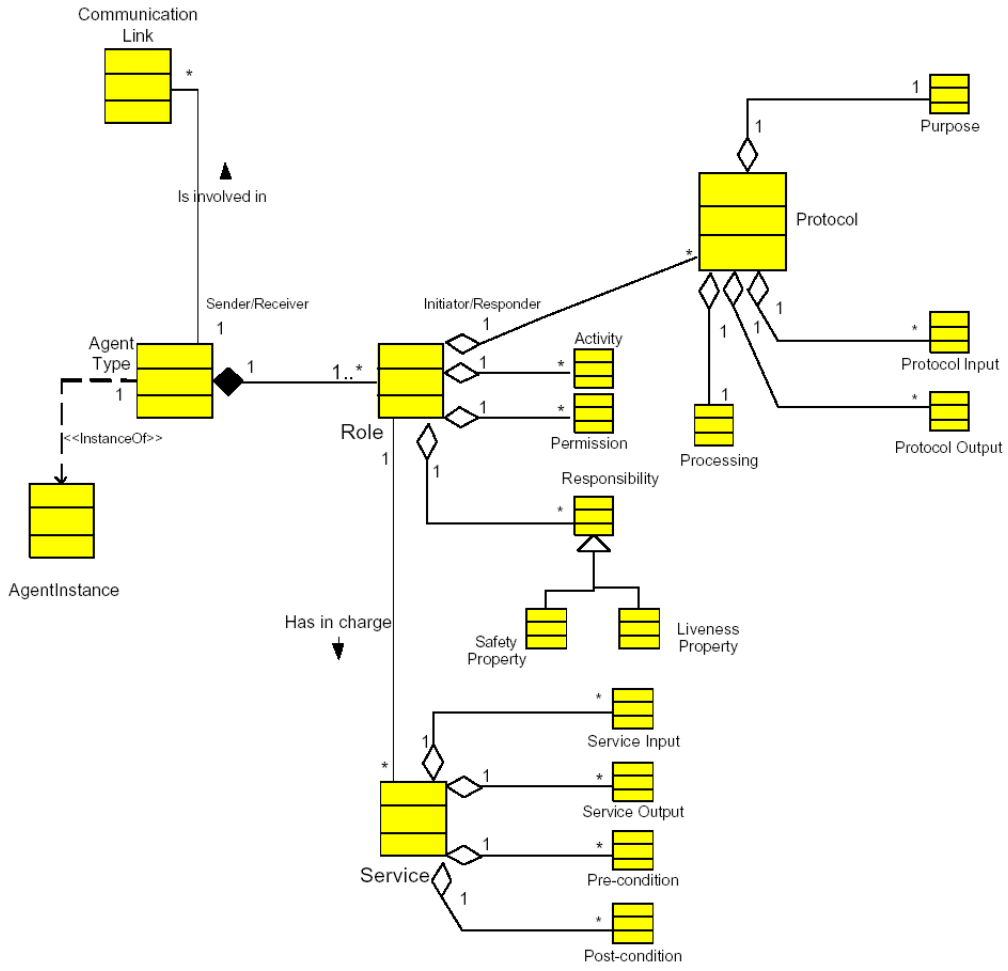


FIG. 3.5 – Le méta-modèle de la méthodologie GAIA

Un rôle est défini par quatre attributs : 1) *les responsabilités*, 2) *les permissions*, 3) *les activités* et 4) *les protocoles*. Une *responsabilité* définit la fonctionnalité du rôle, elle est décrite par des propriétés d'animation et des propriétés de sécurité. Les propriétés d'animation décrivent le cycle de vie du rôle à travers des expressions d'animation. Les propriétés de sécurité sont des conditions que doit vérifier certaines variables pour garantir la survie du rôle. Ces propriétés sont exprimées par une liste de prédicats.

Les *permissions* sont des droits accordés à chaque rôle pour manipuler des ressources. Ces dernières sont les informations et les connaissances que possède l'agent. Les droits sont de trois types : des droits de lecture, de modification et de génération. Ces règles définissent les besoins du rôle en ressources pour fonctionner correctement.

Une *activité* est l'équivalent d'une méthode dans l'approche orientée objet. Elle correspond à une unité d'action que l'agent exécute, sans qu'il n'ait recourt à une interaction avec un autre agent. Les *protocoles* sont considérés comme des activités qui nécessitent de l'interaction avec les autres agents. Des opérateurs spécifiques sont utilisés pour exprimer la dynamique de l'activité du rôle. Le protocole est décrit par les attributs suivants : l'objectif visé, le(s) rôle(s) initiateur(s)<, le(s) rôle(s) participant(s), les paramètres d'entrée, les résultats et une description du traitement.

3.2.3.2 Démarche de développement

Le processus de développement de *GAIA* [WJK00, ZJW03] comprend une phase d'analyse et une phase de conception (voir figure 3.6). Selon Cernuzzi et al. [CCZ05], *GAIA* applique le modèle de développement en cascade.

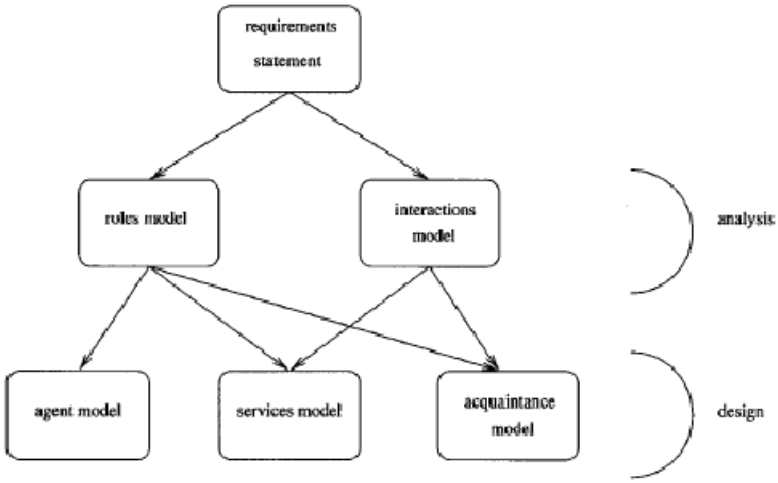


Fig. 3.6 – Les modèles de la méthodologie GAIA [WJK00]

Phase d'Analyse :
La phase d'analyse regroupe le modèle de rôles et le modèle d'interactions. Le *modèle de rôles* énumère les rôles et donne leurs caractéristiques (i.e. les responsabilités, les permissions, les activités et les protocoles). La forme générale du rôle est *NomRole = expression*, où *expression*

est un enchaînement entre les activités et les protocoles assurés par le rôle. Les activités et les protocoles sont les composants élémentaires de ces expressions.

Role Schema: CoffeeFiller

Description:
This role involves ensuring that the coffee pot is kept filled, and informing the workers when fresh coffee has been brewed.

Protocols and Activities:
Fill, InformWorkers, CheckStock, AwaitEmpty

Permissions:

reads	supplied *coffeeMaker*	// *name of coffee maker*
	coffeeStatus	// *full or empty*
changes	*coffeeStock*	// *stock level of coffee*

Responsibilities
Liveness:
CoffeeFiller = (Fill. InformWorkers. CheckStock. AwaitEmpty)$^{\omega}$
safety:
- *coffeeStock* > 0

Fig. 3.7 – Exemple de modèle de rôle [WJK00]

Le *modèle d'interactions* décrit les dépendances et les relations entre les différents rôles du système. Il regroupe un ensemble de définitions de protocoles pour chaque type d'interaction entre les rôles. Un protocole est vu comme un pattern d'interactions, avec une définition formelle et abstraite de toute séquence particulière d'exécution. La séquence des messages échangés dans un protocole n'est pas spécifiée dans la phase d'analyse.

Phase de Conception :
La phase de conception transforme les modèles d'analyse (modèle de rôles et modèle d'interactions) en modèles plus spécifiques. Elle produit un modèle d'agents, un modèle de services et un modèle d'accointances.

Le *modèle d'agent* définit les différents types d'agents existants dans le système et les instances correspondantes. Ce modèle est une arborescence dont les racines correspondent aux différents types d'agents et les feuilles correspondent aux rôles. Dans la figure 3.8, les rôles *CustomerHandler* et *QuoteManager* sont joués par l'agent *CustomerDivisionAgent*. Les cardinalités du côté de l'agent indiquent le nombre d'agents pouvant jouer le rôle.

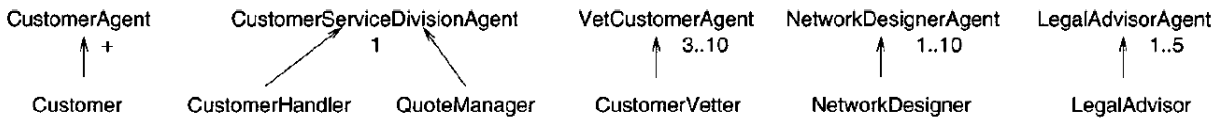


Fig. 3.8 – Un exemple de modèle d'agent

Le *modèle de services* énumère les différents services (les fonctions) assurés par chaque rôle. Dans l'approche orienté objet, un service correspond à une méthode. Chaque activité identifiée

dans le modèle de rôle se transformera en un service. Chaque protocole donnera lieu, au moins, à un service. Chaque service a un ensemble de propriétés : les paramètres d'entrées, les résultats, les pré-conditions et les post-conditions. Les paramètres d'entrées et les résultats seront déduits à partir de la description des protocoles de la phase d'analyse. Les pré et post-conditions sont les contraintes du service, extraites des propriétés de sécurités du rôle.

Le *modèle d'accointances* représente les liens de communication entre les différents types d'agents. C'est un graphe, dont les noeuds correspondent aux types d'agents et les arcs expriment les liens de communication. Ce modèle est déduit des précédents modèles de rôles, de protocoles et d'agents.

3.2.3.3 Passage à l'implémentation

L'approche de modélisation préconisée dans *GAIA* se base sur une spécification assez abstraite du système multi-agents. Elle fait abstraction de l'architecture interne de l'agent. Cette abstraction offre plus de pérennité aux modèles conceptuels. Ainsi, le développeur peut les raffiner selon la plate-forme d'implémentation.

Par contre, nous ne pouvons pas modéliser avec GAIA, par exemple les croyances et les intentions d'un agent BDI ; ce qui demande un effort supplémentaire de la part du développeur. Par ailleurs, il n'existe pas d'outils qui supportent les différentes phases de GAIA. Par conséquent, nous obtenons à la fin de la phase conceptuelle des diagrammes graphiques et textuels des agents, qui sont difficiles à transformer en une implémentation. De plus, *GAIA* ne propose pas une démarche pour aller d'un modèle conceptuel à un modèle d'implémentation du système.

Moraitis et al. [MPS02] proposent une démarche qui permet de passer d'une modélisation *GAIA* à une implémentation avec la plate-forme JADE [BCTR04]. Ils donnent des recommandations de modélisation pour guider le développeur dans la phase d'implémentation. Ces recommandations se résument en les points suivants :

1. Définir tous les messages *ACL* en utilisant les modèles de protocoles et d'interactions de *GAIA*.
2. Conception de structures de données demandées et des modules qui seront utilisés par les agents par le biais de modèles de rôles et d'agents.
3. Décider des choix d'implémentation pour les propriétés de sécurité de chaque rôle.
4. Définir le comportement sous JADE en commençant par les comportements du bas niveau en utilisant les diverses classes de comportements fournies par JADE. Pour cette étape il faut utiliser les modèles de rôles. Les comportements qui sont activés à la réception d'un type message spécifique doivent ajouter un comportement de réception ou la spécification

d'un template de message au démarrage de leur action. Les activités *GAIA* qui s'exécutent en séquentiel, sans interaction ou appel à des protocoles, peuvent être regroupés dans une seule activité. Cependant, pour une meilleure réutilisabilité et clarté de programmation, nous croyons qu'un développeur pourrait choisir de les implémenter comme des méthodes séparées ou comme des actions dans un automate.

5. Les rôles définis dans GAIA se transforme en code réutilisable sous JADE.
6. La méthode d'initialisation de l'agent appelle toutes les méthodes, équivalentes aux activités dans GAIA, exécutées au début de l'activité de l'agent, par exemple *RegisterDF*.
7. Ajouter tous les autres comportements dans le scheduler de l'agent.

Ces informations sont d'une grande utilité pour le développeur sous JADE. Par contre, les recommandations sont informelles. Par conséquent, ces règles sont utilisées par le développeur manuellement. Par ailleurs, la modélisation avec GAIA n'est pas outillée, ce qui ne favorise pas l'automatisation de l'application de ces règles.

3.2.4 INGENIAS

3.2.4.1 Méta-modèles

Dans *INGENIAS*, l'approche générale pour spécifier un système multi-agents consiste à diviser le problème en plusieurs aspects plus concrets qui forment les différentes vues du système. Chaque type de vue est décrit en utilisant le langage de méta-modélisation GOPRR [TR03]. Ainsi, une vue représente, selon cette méthodologie, un modèle du système en cours de développement.

Les méta-modèles sont appliqués pendant l'analyse en recherchant une première version du SMA. Dans ce sens, un méta-modèle agit en tant que directive pour l'analyste, en indiquant quelles entités doivent être définies. Pendant la conception, le méta-modèle est raffiné, en identifiant de nouveaux composants et rapports parmi eux, pour atteindre le niveau de détail approprié. En parallèle, les composants du méta-modèle sont traduits en entités de calcul, comme les machines à états, les gestionnaires de sessions, ou moteurs de règles de production. A la fin du processus on obtient une spécification du système multi-agents avec une conception qui est basée sur des entités concrètes et implémentables.

INGENIAS identifie cinq méta-modèles qui décrivent les vues du système. Chaque vue, étant une instance d'un méta-modèle, est nommée *modèle*. Le concepteur doit utiliser les modèles suivants [SF02] :

– **Modèle d'agent :** Décrit pour chaque agent ses tâches, ses buts, son état mental et les rôles qu'il joue. D'ailleurs, ces différents modèles sont employés pour décrire les états

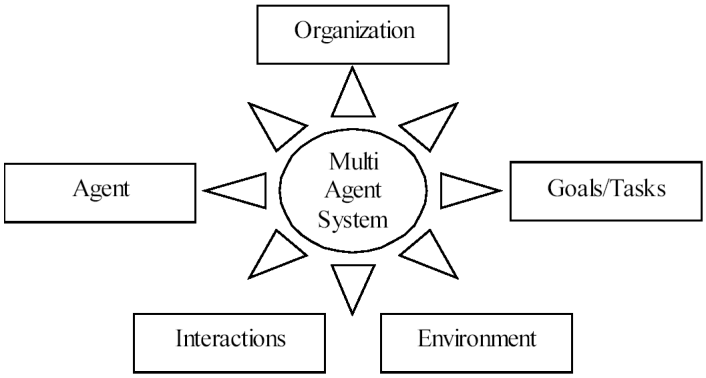


FIG. 3.9 – Les cinq vues qui structurent l'approche *INGENIAS* [PGS03]

intermédiaires de l'agent. Ces états sont présentés en utilisant des buts, des faits, des tâches, ou n'importe quelle autre entité de système qui aide à la description de son état. De cette façon, un modèle d'agent pourrait représenter dans quel état devrait être un agent qui commence une négociation.

- **Modèle d'interaction :** Décrit comment l'interaction s'établit entre les agents. Chaque déclaration d'interaction inclut les acteurs impliqués, les buts poursuivis par l'interaction, et une description du protocole suivi par interaction. *INGENIAS* emploie les diagrammes de collaboration d'*UML* et les diagrammes de *GRASIAS* pour décrire l'interaction. Cependant, il n'y a pas de contrainte sur le formalisme à utiliser pour décrire le protocole.
- **Modèle de tâches et de buts :** Décrit les rapports qui existent entre les buts et les tâches, d'un point de vue structurel. Il est également employé pour définir les entrées et les sorties des tâches, et quels sont leurs effets sur l'environnement ou sur l'état mental de l'agent.
- **Modèle d'organisation :** Décrit comment des composants de système multi-agents (agents, rôles, ressources, et applications) sont groupés ensemble, quelles sont les tâches qui sont exécutées en commun, quels sont les buts qu'ils partagent, et quelles sont les contraintes qui existent dans l'interaction entre des agents. Ces contraintes sont exprimées sous la forme de rapports de serveur de subordination et de client.
- **Modèle d'environnement :** Définit les termes de la réception des éléments de l'environnement, chez l'agent. Il identifie les ressources du système et leur gestion.

3.2.4.2 Démarche développement

INGENIAS [SF02] applique explicitement le processus *USDP* (*Unified Software Development Process*). C'est un modèle de processus itératif identifiant deux dimensions pour le processus se développement : le temps (les phases du cycle de vie), et le contenu (les modèles et

d'autres artefacts). Ainsi, le modèle de processus qu'il adopte est le prétendu processus amélioration par étapes, une forme spécifique du modèle de processus par incrémentation [CCZ05]. *INGENIAS* couvre les phases d'analyse, de conception, de codage et d'implémentation. Ceci suggère qu'*INGENIAS*, comme MASSIVE, pourrait être une méthodologie très pertinente pour le développement rapide du système d'agent, avec l'avantage supplémentaire de fournir les notations standard et les outils prêts à employer. *INGENIAS* adopte la notation *AUML* tout au long du processus de développement.

3.2.4.3 Passage à l'implémentation

IDK représente le kit de développement d'Ingenias. Il a deux parties :

1. l'editeur INGENIAS : Avec lui, nous générons les specifications du SMA en utilisant des concepts génériques d'agent. L'éditeur d'INGENIAS est l'outil de développement principal d'INGENIAS. C'est le remplacement de Rose Rational ou d'autres outils basés sur UML pour ces chercheurs qui travaillent avec des agents de logiciel. Actuel, il permet de produire une représentation multi-niveau des systèmes multi-agents. Cet éditeur est le générateur de spécifications plus tard employées par des générateurs de code. La version en cours (IDK 2.5) supporte les points suivants :
   - une version alpha des diagrammes de protocole d'AUML (de version),
   - faciliter la création/suppression/modification des diagrammes,
   - structurer les diagrammes dans des packages,
   - faciliter la génération de code,
2. les outils INGENIAS et les générateurs de code d'INGENIAS. C'est un framework qui facilite le parcours de la spécification et la productions du résultat. Ce framework a été intégré avec l'éditeur d'INGENIAS de tels sorte que les réalisateurs puissent :
   - ajouter automatiquement de nouveaux outils dans l'IDE. Ces outils traitent les diagrammes directement existants dans l'outil à travers un API qui est fourni directement au programme d'extension.
   - ajouter automatiquement de nouveaux générateurs de code dans l'IDE. Un générateur de code parcours les différents diagrammes et produit d'une partie du code qui utilise les données de ces diagrammes. Les générateurs de code sont une solution faisable pour surmonter l'intervalle qui existe entre la spécification et l'implémentation.

L'IDK est très important dans le processus de développement d'INGENIAS puisqu'il permet le développement rapide d'applications, le découplage des spécifications de l'implémentation, et la vérification de la spécification suivant les besoins d'implémentation.

L'interaction peut être spécifiée avec différentes notations : avec un diagramme de collaboration d'UML, avec un diagramme d'interaction de GRASIA (*GRupo de Agentes Software :*

Ingenieria y Aplicaciones), ou avec un diagramme de protocole d'AUML. Actuellement, le code ne peut pas être généré à partir de diagramme AUML.

3.2.5 PASSI (Process for Agent Societies Specification and Implementation)

3.2.5.1 Méta-modèle

Le figure 3.10 décrit le méta-modèle de *PASSI*. Les agents *PASSI* sont conformes à la spécification *FIPA*. Par conséquent, les agents sont implémentés avec une plate-forme compatible *FIPA* (la relation avec l'agent de *FIPA-Platform*) ; ses tâches et ses comportements ont une correspondance directe dans les plate-formes compatibles *FIPA*.

Chaque agent est censé satisfaire ses besoins fonctionnels (i.e. la classe *Requirement*). Les besoins sont exprimés comme cas d'utilisation en phase de description du domaine, et ils sont affectés aux agents dans la phase d'identification d'agent. Un agent peut avoir quelques ressources qu'il peut utiliser pour réaliser ses buts, offrir des fonctionnalités liées à ses besoins ou les partager avec d'autres agents.

L'agent se compose de rôles qui visent à atteindre un but spécifique (la relation entre *Role* et *Goal*), et fournissent un service à la société d'agents. L'activité des rôles est décrite dans des scénarios, représentés par des diagrammes de séquence dans la phase d'identification de rôles.

Le rôle que joue l'agent peut être l'initiateur ou le participant à une communication. Chaque communication est composée de messages conformes au langage *FIPA-ACL*. Chaque message se rapporte à un acte communicatif (ou performatif). Le message est caractérisé par un nom, qui permet de l'identifier durant la conception. Le message contient aussi la connaissance échangée qui est une partie de l'ontologie du domaine, et le langage utilisé pour décrire son contenu. Dans *PASSI*, l'ontologie est formée de concepts, d'attributs et d'actions. L'ontologie du domaine est décrite dans le modèle de spécification des besoins du système.

3.2.5.2 Processus de développement

PASSI [CP02] est une méthodologie qui couvre toutes les phases du développement logiciel (figure 3.11) : la spécification des besoins, l'analyse, la conception, l'implémentation et le déploiement des agents.

La phase de la définition des besoins du système comprend quatre diagrammes : le diagramme de description des besoins du domaine, les diagrammes d'identification des agents,

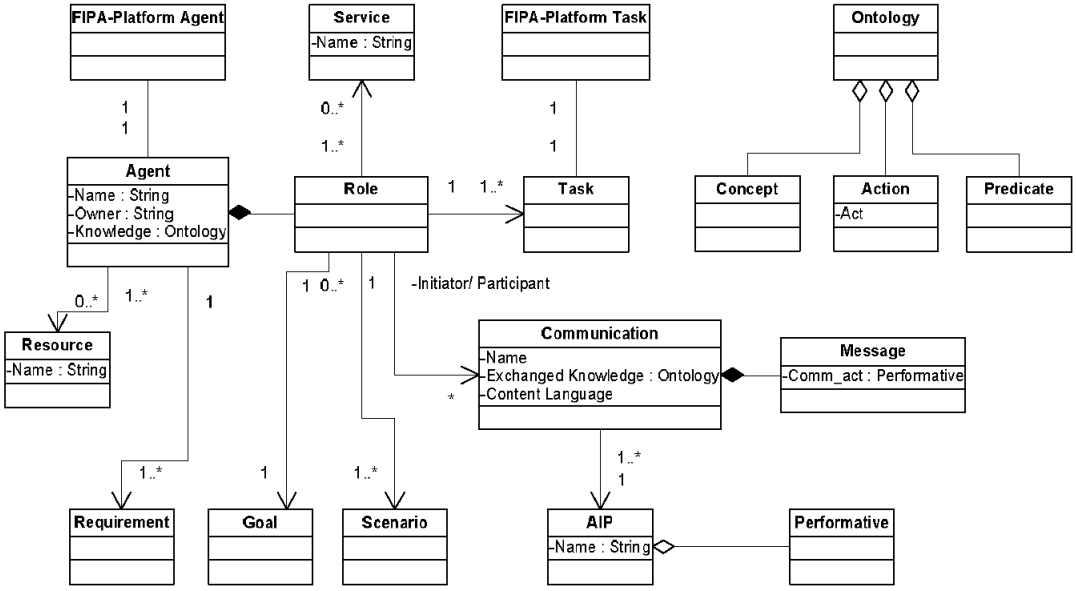


FIG. 3.10 – Le méta-modèle de la méthodologie *PASSI* [Cos03b]

diagramme d'identification des rôles et le diagramme de spécification des tâches. On commence par identifier les cas d'utilisation du système. Puis, on regroupe les fonctionnalités dans des packages qui seront associés aux agents. En se basant sur les besoins du système et des scénarios possibles, on peut identifier les rôles joués par les agents. Ensuite, pour chaque scénario on dessine un diagramme de séquence décrivant l'interaction entre les rôles précédemment identifiés. A partir du diagramme des rôles, on identifie les tâches de chaque agent nécessaire à l'exécution de ces rôles et à l'obtention des responsabilités qui lui sont confiées. Enfin, on décrit les relations de communication entre les tâches des différents agents et du flux de contrôle entre les tâches du même agent.

La phase de définition de la société d'agents est composée de quatre sous phases : la description de l'ontologie du domaine, la description de l'ontologie de communication, la description de rôles, et la description des protocoles.

3.2.5.3 Passage à l'implémentation

La phase d'implémentation comprend quatre différents modèles : les deux premiers modèles définissent la structure du système multi-agents et la structure de chaque agent, et les deux autres modèles décrivent le comportement du système multi-agents et le comportement de chaque agent.

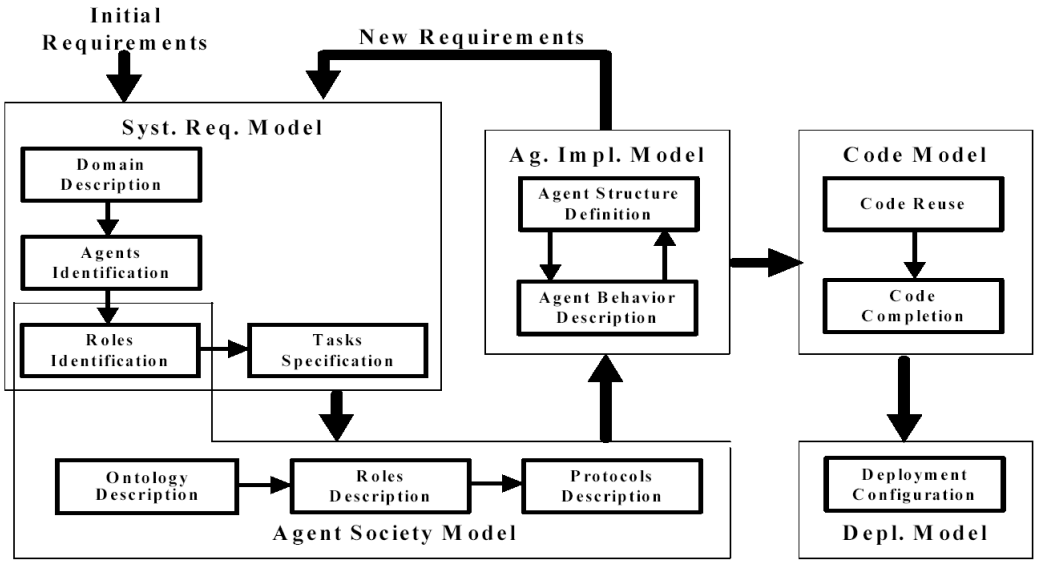


FIG. 3.11 – Le processus de développement de *PASSI*

Le modèle qui définit la structure du système multi-agents utilise un diagramme de classes. Chaque agent est décrit par une classe stéréotypée. La partie des attributs permet de décrire les connaissances de l'agent faisant référence aux entités définies au moment de la description de l'ontologie du domaine. La partie des opérations contient les principales tâches de l'agent. Les relations entre les classes stéréotypées d'agents indique le flux d'informations échangées (i.e. communication).

La structure de chaque agent est ensuite détaillée dans un diagramme de classes. Ce diagramme définit la structure de l'agent et de ses tâches. Une classe permet de représenter un agent ou une tâche. Ces classes sont une spécialisation des classes *Agent* et *Task* de la plateforme *FIPA-OS* [PBH00]. Les attributs et les méthodes de ces classes permettrons de construire l'implémentation réelle (i.e. le code) du système. Il est possible d'automatiser la génération du code de ces diagrammes par des outils commerciaux.

Le comportement du système est décrit par un diagramme d'activité, qui représente le flux d'événements entre les principales agent/tâches. Chaque colonne correspond à une agent et une tâche particulière. Les méthodes d'une tâche sont représentées par des activités. Dans ce contexte, une transition représente soit un événement (réception d'un message ou la fin d'une tâche), soit une invocation de méthode. Si la transition concerne une conversation, alors son label donne le nom du performatif du message et son contenu.

Selon Cossentino et Potts [CP02], la description du comportement de l'agent est tout à fait commune car elle comporte l'implémentation des méthodes. Le développeur peut les décrire dans

la voie la plus appropriée, par exemple, en utilisant des diagrammes de flux, des diagrammes d'état ou des descriptions textuelles semi-formelles.

En effet, *PASSI* utilise la notation *UML* pour la spécification de ses diagrammes. Un ensemble de stéréotypes sont introduits pour enrichir les diagrammes avec des aspects multi-agents, tels que le stéréotype *«Agent»* pour l'élément de modélisation *Package* d'*UML*. *PASSI* recommande le développeur d'utiliser la notation *AUML* pour décrire les protocoles d'interaction.

Afin de faciliter le passage à l'implémentation, *PASSI* se base sur les standards FIPA, tels que le langage de communication *FIPA-ACL* et les protocoles d'interaction de FIPA. *PASSI* préconise pour l'implémentation de ses applications, d'utiliser une plate-forme compatible FIPA, tels que *FIPA-OS* ou *JADE*.

La modélisation avec *PASSI* est supportée par l'outil *PTK* (*PASSI Toolkit*) [CCS04]. *PTK* est un *add-in* du *CASE Tool Rational Rose*, qui offre plus de robustesse et de cohérence à la conception, et une assistance à l'utilisateur non spécialise en multi-agents pour développer son application.

Dans *PTK*, les différents diagrammes qui constituent le résultat final de la conception sont obtenu progressivement, avec une interaction entre le concepteur et *PTK*. *PTK* permet de générer des diagrammes à partir des diagrammes précédents et des choix de conceptuels du développeur. Par exemple, il permet de spécifier les communications (i.e. les messages) via une interface qui contient l'ensemble des informations (i.e. le nom du protocole, l'ontologie, le langage de contenu, etc.) que le concepteur doit fournir. Il peut générer aussi un document XML de l'ontologie du domaine.

PTK se limite à l'utilisation des protocoles d'interaction de FIPA pour spécifier les communications. Par ailleurs, *PTK* ne supporte pas la phase de définition des protocoles d'interaction, qui doit être gérée par le concepteur en utilisant la notation AUML.

Bien que l'outil *PTK* apporte une assistance au concepteur durant le processus de développement, il ne propose pas des solutions pour générer le code des diagrammes d'implémentation, qui décrivent la structure et le comportement des agents, avec une plate-forme donnée.

3.2.6 ROADMAP

La méthodologie *ROADMAP* [JPS02] est une extension et une amélioration de la méthodologie *GAIA*. *ROADMAP* propose une démarche pour développer des applications caractérisées par une évolution dynamique de la structure et du comportement des agents.

3.2.6.1 Le méta-modèle

Le méta-modèle de ROADMAP reprend les concepts de la méthodologie *GAIA*. Nous trouvons des rôles simples, comme ceux définis dans *GAIA*, et des rôles composites qui utilisent le mécanisme d'agrégation (voir figure 3.12), pour donner différentes vues du même système selon le niveau de détail demandé. Un système en entier peut être perçu comme un seul rôle. Chaque rôle a ses propres fonctionnalités et ses objectifs. Un nouveau concept introduit celui de zone d'environnement qui est une modélisation d'une partie l'environnement dans lequel peuvent évoluer des agents par le bias de ses roles.

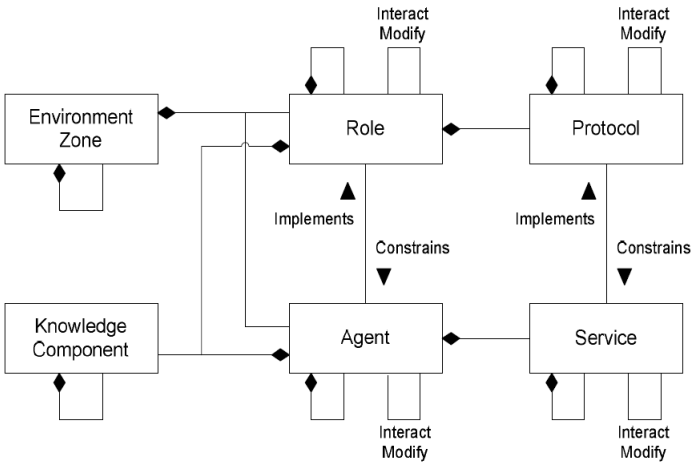


FIG. 3.12 – Le méta-modèle de la méthodologie *ROADMAP*

3.2.6.2 Le processus de développement

ROADMAP suit également une démarche de développement en cascade [CCZ05]. Elle couvre les trois premières phases du développement (voir figure 3.13) : la spécification de besoins, l'analyse et la conception.

Les cas d'utilisation :
Le premier modèle de ROADMAP décrit les cas d'utilisations, qui donne une première vue de l'application à travers les interactions entre l'utilisateur et les agents.

Modèle de l'environnement :
Le modèle de l'environnement est déduit du modèle de cas d'utilisation. Il contient une arborescence des zones ainsi que leurs descriptions qui regroupent les objets statiques, les objets, les contraintes, les sources d'incertitude, et les suppositions sur les zones. Les objets statiques sont des entités perçues par les agents, mais avec qui ils n'interagissent pas. Par contre les objets

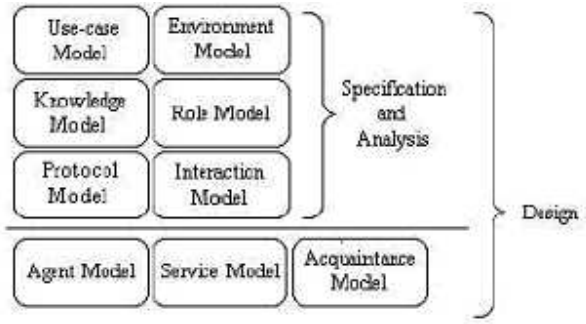


FIG. 3.13 – Les modèles de la méthodologie *ROADMAP*

sont des entités avec qui les agents interagissent. Les concepts d'héritage et d'agrégation du paradigme objet sont utilisés pour décrire la hiérarchie des zones et de leurs objets.

Modèle de connaissances :
Le modèle de connaissances est une description des informations du domaine. Il consiste en une hiérarchie des composants de connaissances et de leurs descriptions. Les connaissances utilisées dans le modèle d'environnement et celui des cas d'utilisation sont décomposées en composants cohérents selon leurs appartenance aux zones. Ensuite, le cycle de vie de chaque composant est analysé ainsi que les relations de dépendance avec les autres composants. Le modèle de connaissances est le maillon qui relie les deux modèles de cas d'utilisation et d'environnement avec le modèle de rôles.

Modèle de rôles :
Le modèle de rôles de *GAIA* est étendu par une arborescence des rôles et une description de chaque rôle dans chaque hiérarchie. Dire qu'un sous-rôle participe à un une hiérarchie du super-rôle, signifie que les responsabilités du sous-rôle interagissent pour achever les responsabilités du super-rôle. Les actions du sous-rôle interagissent pour faire émerger les actions du super-rôle.

Modèles du niveau conceptuel :
Dans cette phase, les rôles sont associés aux agents. Chaque rôle possède une référence à l'agent pour pouvoir accéder à ses informations. Chaque protocole ou activité du rôle correspond à un ou plusieurs services. Les modèles d'accointances et modèles de services sont identiques aux modèles définis dans *GAIA*.

3.2.6.3 Passage à l'implémentation

Les modèles produits dans la phase de conception ne diffèrent pas des modèles conceptuels de GAIA. La seule différence, est l'ajout du concept Rôle dans *ROADMAP*. Similaire à la

méthodologie *GAIA*, *ROADMAP* ne propose pas des outils qui aident le développeur dans sa représentation des différents modèles *ROADMAP*. Par conséquent, le développeur ne peut pas exploiter ou réutiliser les modèles *ROADMAP* pour l'implémentation du système multi-agents. De plus, *ROADMAP* ne propose pas de solutions logicielles ou un choix de plate-forme qui supportent la phase d'implémentation.

3.2.7 TROPOS

Tropos est une méthodologie de développement orientée agent qui est basée sur le concept de besoins, adoptés de la méthodologie *I** [Yu95]. *Tropos* couvre tout le cycle de développement, depuis la définition des besoins non fonctionnels jusqu'à l'implémentation effective de l'application.

La méthodologie *Tropos* modélise les aspects internes de l'agent : les buts, les croyances et les plans. Cette approche s'interesse principalement à l'architecture des agents BDI. Pour modéliser également d'autres aspects des SMA, tels que l'interaction, *Tropos* utilise dans sa démarche les diagrammes de classe, les diagrammes d'activité et de séquence d'UML et les diagrammes d'interaction d'AUML.

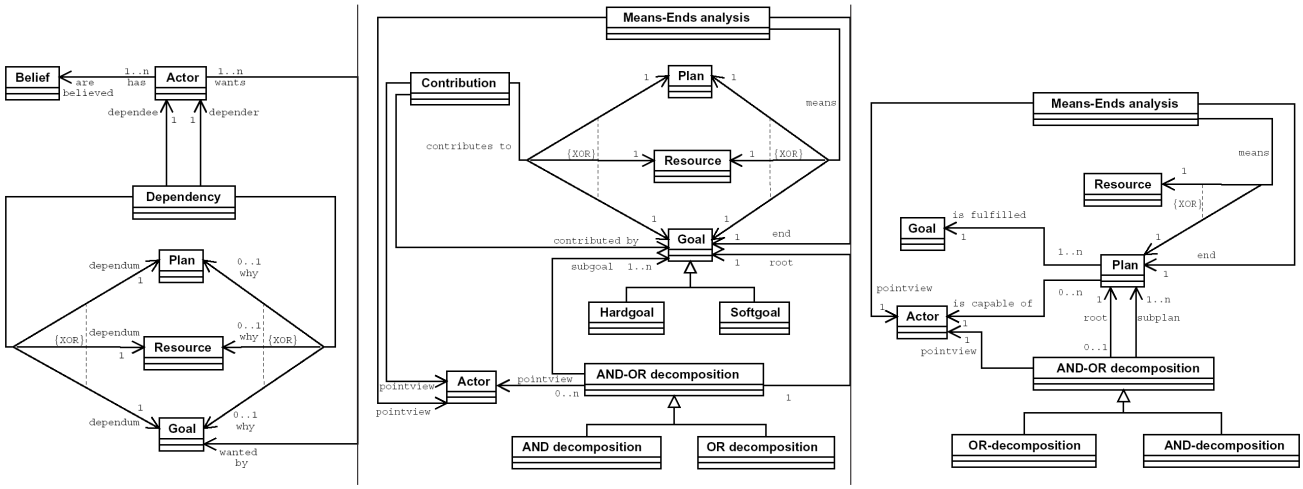
3.2.7.1 Méta-modèle

Le diagramme de classe de la figure 3.1, donne les éléments conceptuels utilisés dans Tropos. Les modèles de *Tropos* [BPG$^+$04] emploient les éléments de modélisation suivants :

- L'***Acteur*** est une entité qui a des buts et l'intentionality stratégiques dans le système ou l'arrangement d'organisation. Un acteur représente un agent physique ou logiciel aussi bien qu'un rôle ou une position.
- Le ***But*** décrit les intérêts stratégiques des acteurs. Il y a le but rigide et le but flexible.
- Le ***Plan*** représente, à un niveau abstrait, une manière de faire quelque chose. L'exécution du plan peut être un moyen pour satisfaire un but.
- Une ***Ressource*** est une entité physique ou informationnelle.
- La ***Dépendance*** est une relation entre deux acteurs, qui indique qu'un acteur dépend, pour certaines raisons, de l'autre afin d'atteindre un certain but, exécuter un certain plan, ou fournir une ressource. Le premier acteur source s'appelle le *depender* alors que le dernier s'appelle le *dependee*. L'objet sur lequel porte la dépendance s'appelle le *dependum*, qui peut être un but, une ressource ou tâche.
- La ***Contribution*** est un rapport entre les buts ou les plans représentant comment et combien les buts ou les plans peuvent contribuer, positivement ou négativement, dans la réalisation du but.

- La ***Décomposition*** est un rapport entre les buts ou les plans représentant une décompostion ET/OU en sous-buts ou sous-plans.
- Les ***Compétences*** sont les capacités d'un acteur de définir, de choisir et d'exécuter un plan pour l'accomplissement d'un but, selon certains états du monde et en présence d'un événement spécifique.
- La ***Croyance*** est la connaissance de l'acteur du monde.

TAB. 3.1 – (a) Le méta-modèle de l'acteur (b) Le méta-modèle du but (c) Le méta-modèle du plan



3.2.7.2 Démarche de développement

TROPOS suit une approche incrémentale par raffinement itératif [CCZ05]. La démarche proposée par *TROPOS* se décompose en cinq phases [GMP02] :

- La détermination des besoins préalables qui consistent à identifier les principaux acteurs et leurs buts respectifs dans le système.
- La détermination des besoins finaux qui décrivent le système à construire en imaginant dans son environnement réel avec ses fonctionnalités.
- La conception architecturale décrivant l'architecture globale du système en terme de sous systèmes inter-connectés par des flux de données et de contrôle.
- La conception détaillée où chaque composant architectural est défini plus en détails : entrées, résultats et toutes autres informations.
- L'implémentation qui transforme le résultat de la conception en squelettes de codes pour la réalisation technique. Une correspondance est faite entre les modèles construits par *TROPOS* et une plate-forme multi-agents.

Nous présentons dans ce qui suit, les différents diagrammes utilisés dans *TROPOS*.

Diagramme d'Acteurs : Il décrit les acteurs, leurs buts et le réseau des rapports de dépendance entre les acteurs. Ce type de diagramme est employé pour définir les besoins et les rapports intentionnels des utilisateurs du système, ou pour montrer des intentions et des relations entre les acteurs du système intérieur (représentant l'architecture des besoins et de l'organisation).

Diagramme des Buts : Il montre la structure interne d'un acteur (ses buts, ses plans et ses ressources) et les rapports entre eux.

Diagramme des Compétences : Des compétences sont modélisées textuellement (par exemple par une liste de compétences pour chaque acteur) ou par les diagrammes de compétences. Le diagramme d'activité d'UML est employé pour modéliser les compétences (ou un ensemble de compétences corrélées) pour un agent spécifique. Les événements externes déclenchent l'état du diagramme de compétence ; les plans sont décrits par les noeuds du diagramme, les arcs des transitions modélisent les événements et les croyances sont décrites par des objets.

Diagramme de Plan : Chaque noeud de plan du diagramme de compétences peut être encore détaillé par un diagramme d'activités UML.

Diagramme d'Interaction d'Agent : Les diagrammes de séquences d'UML sont employés pour décrire l'interaction des agents.

3.2.7.3 Passage à l'implémentation

La phase d'implémentation se fait à l'aide de la plate-forme JACK [Gro04]. L'architecture des agents dans cette plate-forme est de type BDI. L'agent a des buts qu'il veut satisfaire à l'aide de ces plans et de ces croyances. Selon Bresciani et al. [BPG+04], les concepts *Tropos* de la phase de conception détaillée ont une correspondance directe avec les éléments de JACK :

- Agent : définit le comportement de l'agent en incluant ses compétences, les types de messages et d'événements auxquels il répond et les plans qu'il utilise.
- Compétence : peut inclure des plans, des événements, des croyances ainsi que d'autres compétences d'un agent.
- Croyance : est définit dans une base de données relationnelles contenant toutes les croyances d'un agent.
- Evénement : définit une condition permettant de déclencher une action de l'agent. Cela concerne les événements internes et externes de la phase de conception détaillée.
- Plan : définit une séquence d'instructions permettant d'atteindre un but.

Bien qu'il existe une grande similarité entre les concepts BDI définis avec *Tropos* et ceux de la plate-forme *JACK*, ce passage est plus complexe à mettre en ouvre avec de simples règles de correspondance.

3.3 Autres travaux

Carole Bernon et al. [BCG+04] proposent une unification de trois travaux méthodologiques : ADELFE, GAIA et PASSI. Leur méta-modèle regroupe les principaux concepts de chaque méthodologie. Le but de ce travail d'unification, est de trouver une définition commune des concepts multi-agents, à partir des différentes expériences menées en ingénierie des systèmes multi-agents. Selon les auteurs, ce travail est une première étape pour aller à une démarche de développement commune des systèmes multi-agents.

L'unification des méta-modèles et des méthodologies existants, est une chose difficile compte tenu des différentes architectures d'agents et de systèmes multi-agents existants dans la littérature [Occ03], [SBPL04] et [Tve01]. La distinction entre problème et domaine pose la question d'une méthode globale unifiée pour les systèmes multi-agents [Occ03]. La nature des problèmes résolus par les systèmes multi-agents présentent une grande diversité : Simulation, résolution, intégration, contrôle ... rendant difficile une même approche. Le domaine dans lequel est résolus le problème impose lui même une démarche particulière conduisant à des méthodes très dédiées.

Amor et. al. [AFV04] utilisent l'approche MDA [orm01] (*Model Driven Architecture*) pour faciliter le développement des applications multi-agents et plus particulièrement pour faciliter le passage de la conception à l'implémentation. Les auteurs proposent d'utiliser les modèles obtenus dans la dernière phase de la méthodologie *Tropos* comme des modèles PIM. Le modèle d'agent de *Malaca*[AFT03], qui est basé sur une approche modulaire, est utilisé comme modèle PSM. Les plate-formes qui supportent cette approche sont supposées être compatibles FIPA.

Parmi les principaux fondement de l'approche MDA, est la pérennité des modèles et leur réutilisation dans différentes plates-formes implémentation. Nous jugeons que la spécification MDA, est assez générale, non appropriée à une technologie donnée. Ce qui nous amène à l'adapter cette approche de développement au domaine des systèmes multi-agents.

3.4 Synthèse

Nous distinguons deux catégories de méthodologies : les méthodologies qui se basent sur un seul méta-modèle, et les méthodologies qui décomposent le système multi-agents en plusieurs vues, et dont chacun est décrit un ou plusieurs méta-modèles.

La majorité des méthodologies multi-agents, tels que GAIA, ROADMAP, ADELFE et Tropos, proposent un seul méta-modèle pour le développement du système multi-agents. Dans cette même perspective certains travaux essaient d'unifier la représentation du système multi-agents. Nous citons le travail de Bernon et al. [BCG+04] qui proposent une unification des méta-modèles des trois méthodologies PASSI, ADELFE et GAIA. Ce travail a été repris par le groupe de travail AOSE de Agent Link [CBP05] qui a étendu son analyse des méthodologies, en ajoutant INGENIAS, RICA et TROPOS. La proposition d'un méta-modèle des concepts de base du système multi-agents est intéressante si on se limite à un domaine d'application et à une problème particulier. Par contre nous ne pouvons pas décrire avec ce type de methodologies des systèmes multi-agents avec différentes architectures d'agents (i.e. adaptatif, BDI, réactif, etc.) ou différents modes d'interaction (i.e. par message ou par environnement).

La deuxième approche préconise une décomposition du système multi-agents en plusieurs vues. Voyelles [Dem95] est la première méthodologie qui propose une décompose du système multi-agents en cinq vues : agent, environnement, interaction, organisation et utilisateur. D'autres méthodologies telles que INGENIAS et MASSIVE, proposent d'autres décomposition multi-agents. Toutes ces méthodologies se ressemblent dans leur décompositions. Elles ont en commun les aspects : agent, environnement, interaction et organisation.

La conception du système multi-agents en plusieurs vues est une solution pour le problème de la diversité des problèmes et des domaines d'application [Occ03]. Ainsi le concepteur donne la définition de chaque aspects qui convient le mieux au domaine d'application auquel il s'intéresse et au problème qu'il essaie de résoudre avec son système multi-agents. Un autre avantage de cette architecture multi-vues est la possibilité d'utiliser différentes modélisation d'un même aspect dans un même système multi-agents. Nous pouvons par exemple concevoir un système multi-agents contenant des agents avec des architectures BDI, adaptatifs et reactifs, et dont l'interaction peut être en mode indirecte (i.e. par l'environnement) et en mode direct (i.e. par envoi de messages).

Il n'existe pas un consensus sur la représentation des concepts multi-agents. Ainsi, les plateformes multi-agents ont des représentations des concepts multi-agents différentes des méthodologies. Par consequent, il existe une décalage entre la représentation méthodologique et la représentation d'implementation du système multi-agents. Cela nécessite un effort supplémen-

taire de la part du développeur qui utilise une méthodologie pour l'analyse et la conception de son système, et une plate-forme pour son implémentation.

Pour facilité le passage de la conception vers l'implémentation certaines méthodologies se sont conformés aux standards tels que FIPA. Par exemple *PASSI* supporte les architectures et les services proposés par FIPA, et elle recommande l'implémentation du système avec une plate-forme compatible *FIPA* tels que *JADE* et *FIPA-OS*. Une limité de cette approche est la non pérennité des modèles conceptuels de *PASSI*. Une autre limité, il n'existe pas une démarche complète qui assure le passage au niveau implémentation en se basant sur les modèles conceptuels.

D'autres méthodologies se caractérisent par leur représentation assez abstraite du système, telles que *AALAADIN*, *GAIA*, *ROADMAP* et *VOYELLES*. Une telle représentation facilite la réutilisation du modèle conceptuel pour obtenir des implémentations avec différentes plate-formes. La limite de ces méthodologies est l'absence d'une démarche qui guide le développeur dans l'implémentation de son système avec une plate-forme donnée.

Toutes ces méthodologies n'utilisent pas le savoir-faire qui permet d'utiliser les modèles conceptuels pour obtenir une implémentation dans une plate-forme donnée. Par conséquent, le passage à l'implémentation dans ces méthodologies reste manuel et informel.

Les méthodologies présentent un ensemble de limites résumées en les points suivants :
- l'absence d'une démarche qui assure le passage du niveau conceptuel vers le niveau d'implémentation,
- la non pérennité des modèles,
- le passage de la conception à l'implémentation est coûteux en temps et en efforts de développement.

3.5 Conclusion

Ce chapitre nous a permis de présenter quelques méthodologies multi-agents, dans le but d'étudier leurs apports dans le processus de développement des systèmes multi-agents et particulièrement dans la phase d'implémentation. Nous avons identifié une difficulté dans les méthodologies actuelles pour assurer le passage du niveau conceptuel au niveau d'implémentation. De plus, nous trouvons important de pouvoir modéliser des systèmes multi-agents avec de différentes architectures agents et multi-agents, pour répondre à des besoins liés au contexte d'application et au problème à résoudre.

Pour remédier à ces problèmes, nous proposons une démarche de développement qui se base sur l'approche *MDA*. Le chapitre 6 présente en détails notre démarche *MDAD*.

Chapitre 4

Protocoles d'interaction réutilisables

4.1 Introduction

Le développement d'un système multi-agents nécessite souvent le développement d'agents dotés d'un composant d'interaction. En effet, différentes représentations ont été proposées pour faciliter la conception des interactions des agents [Hug01] [Maz01]. Cependant, l'implémentation réutilisable des protocoles d'interaction reste un problème assez difficile [EC04] [TWD02].

Ce chapitre décrit notre représentation des protocoles d'interaction, proposée dont l'objectif de faciliter leur réutilisation. Dans un premier temps nous donnons deux exemples de situations d'interaction qui montrent le besoin d'une représentation réutilisable des protocoles, et d'une démarche pour leur intégration dynamique au sein des agents. Ensuite, nous donnons notre démarche d'analyse appliquée à deux protocoles d'interaction de FIPA : le *Contract Net* et l'*Enchère Anglaise*. Cette analyse nous permis de définir une ontologie des protocoles d'interaction, présentée dans la section 4.5.

4.2 Besoin d'une représentation réutilisable

Pour illustrer l'intérêt de la réutilisation des protocoles d'interaction, nous proposons le scénario suivant : un agent $Agent_i$, qui a le rôle initiateur de *FIPA-Contract-Net* [FIP02a] (cf. 3.2), diffuse un appel à proposition pour une tâche donnée. Il reçoit une seule proposition de l'agent $Agent_j$. De nouveau, $Agent_i$ relance un autre appel à propositions et il ne reçoit à nouveau qu'une seule proposition de $Agent_j$. Si $Agent_i$ était adaptatif, il remplacerait le

protocole d'interaction *FIPA-Contract-Net* par le protocole d'interaction *Request* [FIP02b]. Ceci optimisera le nombre de messages dans le système.

Nous pouvons aussi imaginer ce deuxième scénario : un agent $Agent_i$ qui ne connaît que le protocole *Contract net*, reçoit un appel à participation pour une interaction avec le protocole *Request*. Il répondra alors par un message *Not_ understand* qui met fin à cette interaction. Si $Agent_i$ était adaptatif, il aurait pu récupérer et exécuter le rôle participant du protocole *Request*.

Ces deux exemples montrent bien le besoin d'une représentation réutilisable des protocoles d'interaction qui offrirait plus de flexibilité aux comportements de l'agent et lui permettrait d'intégrer facilement et de façon dynamique de nouveaux protocoles.

4.3 Une représentation componentielle

L'approche la plus réaliste pour modifier, maintenir ou faire évoluer dynamiquement un système, est de pouvoir lui substituer, supprimer et agréger des composants. Cette approche est soutenue par la communauté des systèmes à composants.

La notion de composant logiciel n'a pas encore reçue une définition aussi complète et communément acceptée que celle d'objet. Cependant, la communauté scientifique s'accorde sur un certain nombre de caractéristiques propres à l'approche composant. De manière générale, un composant a un état et un comportement, représentés respectivement par des attributs et des méthodes. Un composant peut être réutilisé et intégré dans plusieurs applications. Pour cela, il doit être suffisamment autonome et doit comporter toutes les informations lui permettant d'être assemblé avec d'autres composants. L'utilisation d'un composant se fait par le biais de ses interfaces requises et ses interfaces offertes.

Pour une représentation réutilisable du protocole d'interaction, nous préconisons de représenter chaque rôle du protocole par un composant. Par analogie au concept de composant, les rôles d'interaction possèdent un ensemble de paramètres d'entrée, avec d'éventuels paramètres de sortie. Le rôle d'interaction possède un comportement (i.e. envoie et réception de messages et des actions décisionnelles) et un état correspondant à l'état de l'interaction. Les motivations de ce choix sont :

- avoir une spécification et une implémentation génériques des rôles d'interaction, indépendante du contexte applicatif,
- faciliter l'instanciation et l'exécution des rôles d'interaction,
- faciliter la mise à jour ou la modification du comportement interactif de l'agent.

Notre démarche consiste à analyser en premier lieu les protocoles d'interaction dans le but d'identifier les paramètres d'entrée nécessaires à leur instanciation. A partir de cette analyse nous essayerons de définir une ontologie qui unifie et réifie ces paramètres. Cette ontologie a l'avantage d'offrir une sémantique commune pour décrire les rôles d'interaction de telle manière à ce que tous les agents qui utilisent cette ontologie peuvent adopter facilement de nouveaux protocoles d'interaction et adapter leur comportement lorsque cela est nécessaire. Pour cela, nous nous sommes fixé deux hypothèses.

Hypothèse 1 : *Les protocoles d'interaction de FIPA sont à l'heure actuelle les seuls protocoles standardisés dans le domaine des systèmes multi-agents. Nous nous intéressons dans notre étude à ces protocoles.*

Hypothèse 2 : *Nous utilisons le langage de communication FIPA-ACL, car c'est le langage standardisé et utilisé pour la spécification des protocoles FIPA.*

4.4 Analyse des protocoles d'interaction de FIPA

Nous analysons dans cette section deux protocoles d'interaction : *FIPA-Contract-Net* et *FIPA-English-Auction*. Le but de cette analyse est d'élaborer une représentation générique de chaque rôle d'interaction. Nous essayons d'identifier pour chaque rôle d'interaction : ses paramètres d'entrée, ses paramètres résultat et ses états finaux. De plus, nous donnons pour chaque protocole d'interaction, l'objet sur lequel porte l'interaction.

4.4.1 Analyse du protocole Contract Net

Le protocole *Contract Net* est le protocole d'interaction le plus utilisé dans les systèmes multi-agents. Il repose sur un mécanisme d'allocation de tâches régi par le protocole d'appel d'offres qui est utilisé dans les organisations humaines. Ce protocole est basé sur le rôle d'initiateur (appelé *Manager*) et le rôle participant (voir figure 4.1) que nous détaillons dans la suite de cette section.

4.4.1.1 Le rôle initiateur du Contract Net

La figure 4.2 est un automate qui modélise l'activité du rôle initiateur. L'initiateur commence par envoyer une appel à proposition (*cfp*[1]) qui contient la description de l'action à exécuter et

[1]Call For Proposal

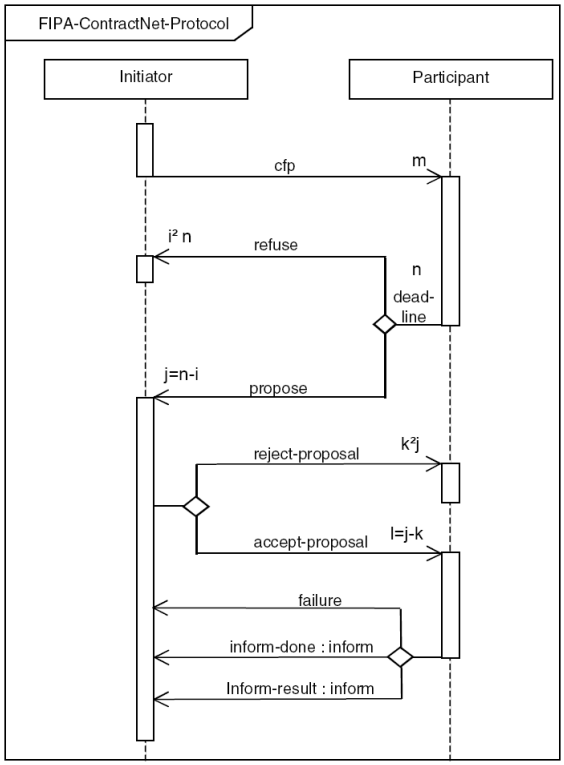


FIG. 4.1 – Le diagramme du protocole *FIPA-Contract-Net*

une expression de pré-condition contenant un paramètre de proposition. Dans une extension de ce protocole, on ajoute un *Timeout* pour éviter des attentes infinies de l'initiateur. Le *Timeout* est indiqué dans l'attribut *reply-by* du message *Cfp*.

Voici un exemple de message *cfp* issue de [FIP02a]. Dans ce message l'$agent_i$ demande à l'$agent_j$ de lui faire une proposition pour la vente de 50 prunes.

```
(cfp
    :sender (agent-identifier : name i)
    :receiver (set(agent-identifier :name j))
    :content
            "((action (agent-identifier :name j)
            (sell plum 50))
            (any ?x (and (= (price plum) ?x) (< ?x 10))))"
    :ontology fruit-market
    :langage fipa-sl)
```

Nous considérons que les actions de communication (actions d'envoi ou de réception de messages) sont génériques, elles sont supportées par l'agent qui joue le rôle. La gestion de communication est une compétence de base de l'agent, supportée par la majorité des plates-formes multi-agents. Les données nécessaires à la préparation du message *Cfp* sont : (i) la description de l'action à exécuter, (ii) la pré-condition et (iii) le(s) paramètre(s) de proposition.

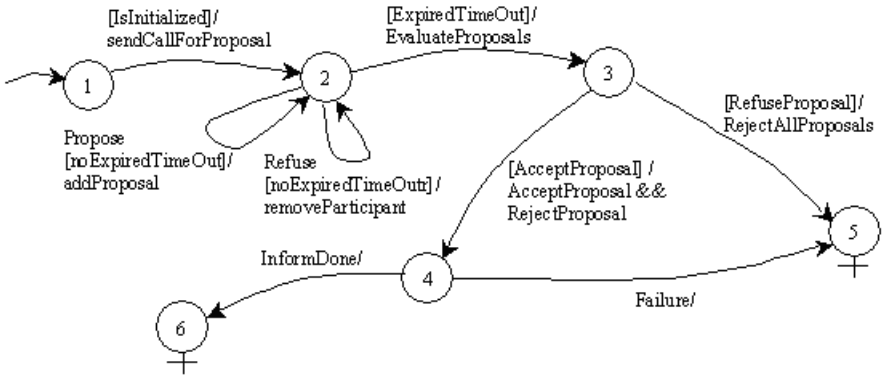


FIG. 4.2 – Le rôle initiateur du protocole *FIPA Contract Net*

Après la réception de toutes les propositions, l'initiateur doit les évaluer pour en choisir une. En cas d'utilisation de *Timeout*, l'évaluation a lieu juste après son expiration, même si l'initiateur n'a pas encore reçu toutes les réponses, qui sont au nombre des participants moins les réponses *Refuse* reçues. Toute proposition reçue, après l'expiration du *Timeout*, sera automatiquement rejetée en envoyant un message *Reject-proposal*. Dans certains cas, cette évaluation peut aboutir à un rejet total de toutes les propositions. L'évaluation des propositions est une action décisionnelle propre à chaque agent. De ce fait, elle doit être considérée comme un paramètre du rôle. Cette action décisionnelle est considérée comme une boîte noire qui prend comme paramètre une liste de propositions et qui peut retourner la proposition sélectionnée.

Dans le cas où une proposition est sélectionnée, l'initiateur envoie un message *Accept-proposal* au participant dont la proposition est acceptée et un message *Reject-proposal* aux autres participants. A la fin de l'interaction, l'initiateur reçoit de la part du participant un *Inform-done* pour confirmer l'exécution de l'action. Il peut recevoir à la place du *Inform-done* un *Inform-result* qui contient le résultat retourné suite à l'exécution de l'action.

L'activité du rôle peut se terminer par :
- un *échec* dans les cas suivants : (i) l'initiateur ne reçoit aucune proposition, (ii) aucune proposition n'est acceptée, ou (iii) l'initiateur reçoit un message *Failure*,
- un *succès*, après la réception du message *Inform-done* ou *Inform-result*.

D'après cette analyse nous pouvons dire que toutes les actions et les décisions de ce rôle peuvent être décrite de façon générique. Cependant, nous constatons que l'exécution de ce rôle

nécessite : (i) la description de l'**action à réaliser**, (ii) la **liste des agents participants** et (iii) le **processus d'évaluation des propositions** `evaluateProposals`.

4.4.1.2 Le rôle participant du Contract Net

La figure 4.3 est un automate qui modélise l'activité du rôle participant. A la réception du *Cfp*, le participant évalue son intérêt pour l'action en fonction de ses ressources et de ses compétences. Si l'action est intéressante, le participant formule une proposition en se basant sur la description de l'action et l'expression de pré-condition, qui est donnée dans le message *Cfp*. Le participant donne une valeur au paramètre de proposition. Voici un exemple de message *propose* issue de [FIP02a]. Dans ce message $agent_j$ propose à $agent_i$ de lui vendre 50 prunes à 5$.

(cfp
:sender (agent-identifier : name j)
:receiver (agent-identifier :name i)
:content
"((action j (sell plum 50))
(=(any ?x (and (= (price plum) ?x) (< ?x 10))) 5))"
:ontology fruit-market
:in-reply-to proposal2
:langage fipa-sl)

Le participant peut aussi refuser de faire une proposition. Pour ce faire, il envoie un message *Refuse* à l'initiateur. L'action décisionnelle de faire une proposition `Decide_to_Propose`, est considérée comme interne à l'agent. Elle prend comme paramètres la description de l'action à exécuter ainsi que l'expression de la pré-condition et peut retourner comme résultat une proposition.

A la réception d'un *Accept-proposal*, le participant s'engagera à exécuter l'action décrite dans le *Cfp* sous les conditions définies dans l'expression de pré-conditions et avec la proposition qu'il vient de soumettre. L'exécution de l'action dépend des compétences et des ressources de l'agent, elle ne fait pas partie de la description du rôle d'interaction. Si le participant parvient à exécuter l'action, alors il envoie un message *Inform-done* ou *Inform-result* contenant le résultat de l'exécution de l'action. Si le participant échoue dans l'exécution de l'action, alors il envoie un message *Failure* contenant la raison de l'échec. Dans le cas où le participant reçoit un message *Reject-proposal*, il quitte l'interaction.

L'activité du rôle peut se terminer par un :

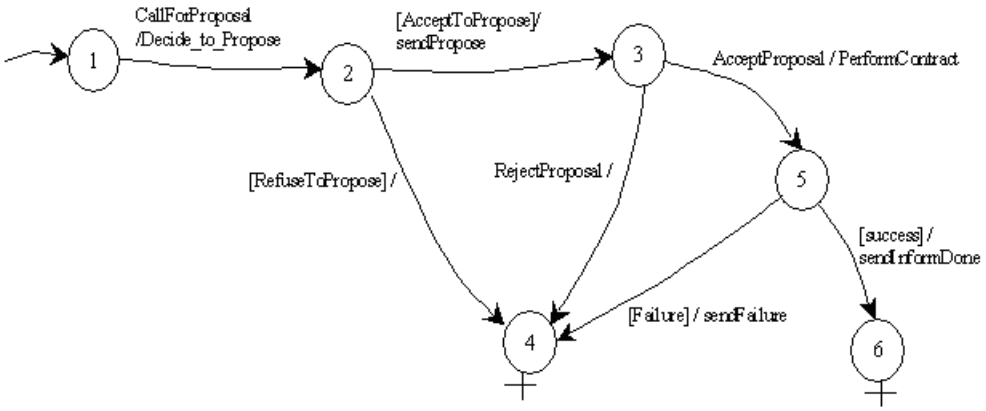


FIG. 4.3 – Participant du protocole FIPA Contract Net

– *échec*, à la réception d'un message *Reject-proposal*, ou l'envoie d'un message *Failure*.
– *succès*, après l'envoie du message *Inform-done* ou *Inform-result*.

En résumé, l'activité de ce rôle peut être décrite de façon générique. Cependant, son exécution nécessite la spécification du **processus de construction de la proposition**.

4.4.2 Analyse du protocole Enchère Anglaise

Une description du protocole d'enchère anglaise est donnée par le diagramme de séquences d'AUML [HOB04] dans la figure 4.4. Le protocole d'enchère anglaise utilise deux rôles : le rôle initiateur (ou le commissaire-priseur) et le rôle participant (ou l'enchérisseur).

4.4.2.1 Le rôle initiateur de l'Enchère Anglaise

Au début de l'enchère, l'initiateur envoie un message *Inform-start-auction* à tous les participants. Ensuite, le commissaire-priseur envoie un premier *Cfp* contenant une proposition avec un prix initial. Le corps du message *Cfp* contient une description du produit et une proposition de prix. Voici un exemple de message *Cfp* pour la vente en enchère d'une voiture avec un prix initial de 5.000 euro.

(cfp
:sender A
:receiver B
:language Prolog
:ontology voiture

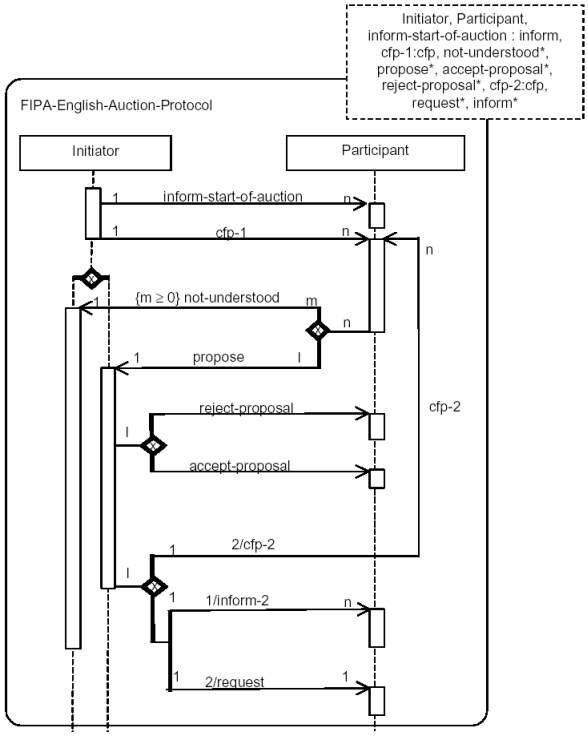


FIG. 4.4 – Le protocole *FIPA english Auction* décrit par un diagramme de séquence d'AUML

:content prix(clio02,5000 EUR))
:conversation-id conv01
:reply-by 1 min)

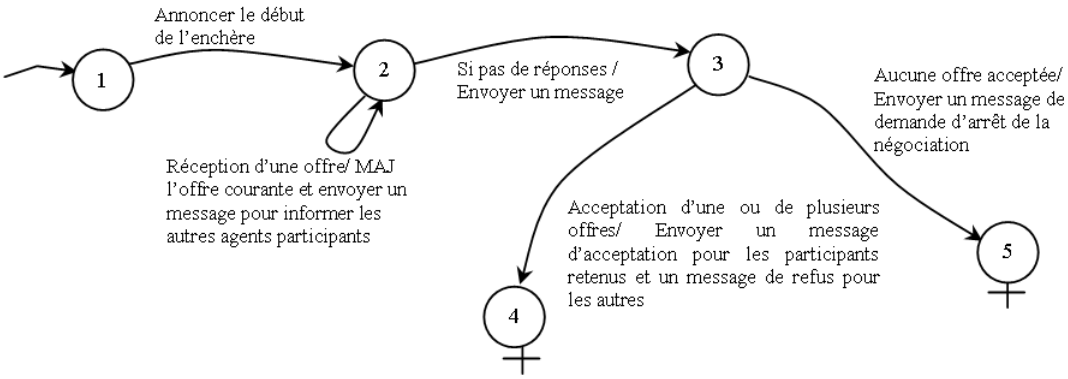


FIG. 4.5 – Le rôle *Initiateur-Enchère-Anglaise*

Après la réception des propositions des enchérisseurs, l'initiateur envoie un message *Accept-proposal* au premier participant qui a répondu au *Cfp*, et un message *Reject-proposal* aux autres

participants. Le message *Reject-proposal* contient le nom du participant dont la proposition est acceptée. Cette information est bien utile pour les participants pour pouvoir raisonner sur le comportement de leurs adversaires et sur les actions à entreprendre dans le prochain tour. L'action d'évaluation des propositions est bien générique, elle ne demande pas une décision propre de l'agent initiateur. Ensuite, l'initiateur envoie un nouveau *Cfp* contenant le nouveau prix qui est une incrémentation du précédent prix d'un pas, qu'il a défini au début de l'enchère. L'action de construction d'une nouvelle proposition est bien générique, il suffit juste de donner le *pas d'incrémentation*. Ce processus est réitéré tant qu'il existe au moins un participant qui valide les propositions de l'initiateur.

Dans le cas où l'initiateur ne reçoit aucune proposition pour un *Cfp* donné, alors il compare le prix de la dernière proposition acceptée avec la valeur de réserve. Cette valeur est initialement fixée par l'initiateur qui indique le prix minimal de vente. C'est une donnée privée de l'initiateur. Si la dernière proposition acceptée est supérieure à cette valeur de réserve, alors l'initiateur envoie un message *Inform* à l'ensemble des participants indiquant la fin de l'enchère, contenant la proposition (nom du participant, le prix de vente) finale. Sinon l'initiateur envoie un message *Reject-proposal* à tous les participants.

Les états finaux de ce rôle sont :
- *échec* quand la dernière proposition acceptée est inférieure à la valeur de réserve ou la non réception de propositions pour le premier *Cfp*,
- *succès*, quand la dernière proposition acceptée est supérieure à la valeur de réserve.

Similaire aux précédents rôles d'interaction, ce rôle peut décrit de façon générique. L'exécution de ce rôle demande un ensemble de paramètres d'entrée :
- la description du **produit** mis en enchère,
- le **Time Out**,
- le **taux d'incrémentation** à chaque tour,
- le **premier prix** donné au premier tour,
- la **prix minimal** de vente du produit.

4.4.2.2 Le rôle participant de l'Enchère Anglaise

Dès la réception du premier *Cfp*, le participant évalue le prix proposé et décide de son acceptation. Cette évaluation est une action décisionnelle propre à l'agent qui joue ce rôle. Si le participant accepte le prix proposé par le commissaire priseur, alors il envoie un message de *Propose* contenant le prix déjà donné dans le *Cfp*, sinon il s'abstient. Suivant la réponse de l'initiateur, soit un *Accept-proposal* ou un *Reject-proposal*, le participant met à jour ses attributs internes, dont il se sert pour ses futures décisions.

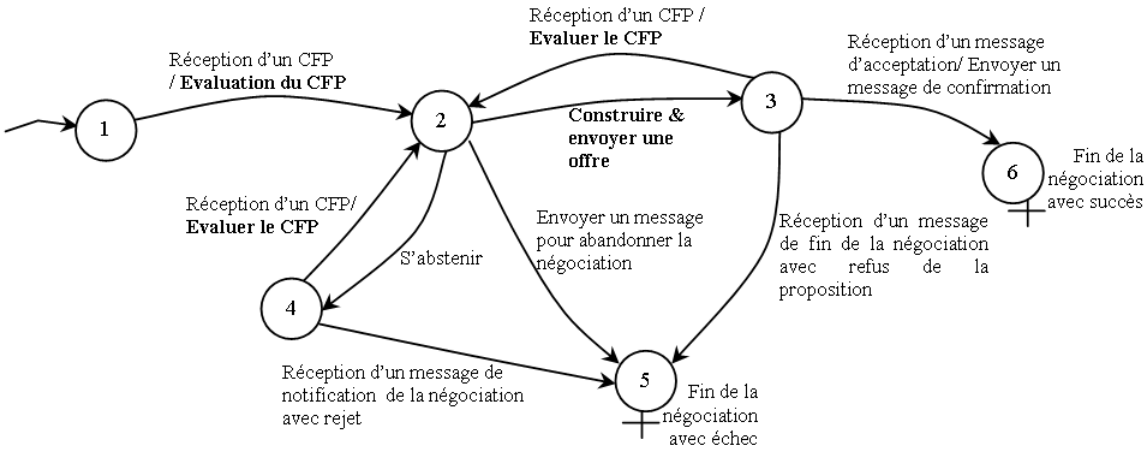


FIG. 4.6 – Le rôle *participant-Enchère-Anglaise*

En résumé, l'activité ce rôle peut être décrite de façon générique. Cependant, son exécution nécessite la spécification du **processus de proposition**.

4.4.3 Synthèse de l'analyse des protocoles d'interaction

L'analyse de ces deux protocoles montre que les protocoles d'interaction ont des comportements génériques, indépendants du contexte d'application. Les actions d'interaction du rôle sont de trois types : (1) les actions de communication (i.e. envoi de message), (2) des actions qui manipulent les données internes du rôle, par exemple, l'ajout de la proposition reçue à l'ensemble des propositions, (3) les actions décisionnelles qui permettent de faire des choix aux cours de l'interaction. La plupart de ces actions sont génériques, à l'exception des actions décisionnelles, qui demandent des compétences propres à l'agent jouant le rôle. Parmi ces actions nous citons l'action d'évaluation de propositions du rôle *Initiateur-FIPA-Contract-Net* et l'action de proposition dans le rôle*Participant-FIPA-Contract-Net.*

En plus des actions décisionnelles, certaines données sont indispensables pour pouvoir spécialiser un rôle d'interaction. Parmi ces données, la description du produit à vendre pour le rôle *initiateur-FIPA-Enchère-Anglaise* ou l'action à exécuter pour le rôle *Initiateur-FIPA-Contract-Net.*

Nous notons que certaines actions décisionnelles, sont communes à plusieurs rôles d'interaction. Par exemple, l'action de proposition est utilisée par le rôle *Participant-FIPA-Contract-net* et le rôle *participant-FIPA-Enchère-anglaise.* De même pour certaines données, par exemple les

deux rôles *Initiateur-FIPA-Contract-Net* et *Initiateur-FIPA-Request* demandent la description de l'action à exécuter.

Sur la base de l'analyse des principaux protocoles d'interaction de *INAF*, nous avons identifié un ensemble de paramètres d'entrée pour ces protocoles, qui sont résumés dans le tableau 4.1. Les concepts utilisés dans ce tableau ont permis de définir une ontologie des protocoles d'interaction que nous présentons dans la section suivante.

4.5 Vers une ontologie des protocoles d'interaction

Une ontologie est une spécification formelle et explicite, des concepts d'un domaine d'intérêt et leurs relations. Guarino [Gua98] donne la définition suivante : *"in AI, an ontology refers to an engineering artifact, constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words. This set of assumptions has usually the form of a first-order logical theory, where vocabulary words appear as unary or binary predicate names, respectively called concepts and relations. In the simplest case, an ontology describes a hierarchy of concepts related by subsumption relationships ; in more sophisticated cases, suitable axioms are added in order to express other relationships between concepts and to constrain their intended interpretation."*

Nous proposons une ontologie pour décrire les protocoles d'interaction, qui offre une terminologie commune de l'interaction, partagée par tous les agents. Par conséquent, l'agent peut réutiliser facilement les protocoles d'interaction. Les concepts de cette ontologie sont principalement une réification des paramètres d'entrée des rôles d'interaction. Nous avons aussi unifié certains concepts, tels que le concept de stratégie utilisé dans la majorité des rôles.

4.5.1 Des travaux sur les ontologies de l'interaction

Pour justifier nos choix conceptuels, nous présentons dans cette section les principaux travaux sur les ontologies des protocoles d'interaction.

4.5.1.1 Une ontologie pour la négociation

Tamma et al. [TWD02] proposent une ontologie pour la négociation (voir figure 4.7). Les auteurs considèrent qu'un protocole de négociation est défini par un ensemble de *règles* explicites régissant l'interaction. L'ontologie proposée reprend la taxonomie des règles de négociation de

TAB. 4.1 – Les paramètres des principaux protocoles d'interaction de *FIPA*

Protocole d'interaction	Rôle d'interaction	Paramètres
FIPA Contract Net	Initiateur	Action Timeout Liste des participants Stratégie d'évaluation
	Participant	Stratégie de proposition
FIPA Iterated Contract Net	Initiateur	Action Timeout Liste des participants Stratégie d'évaluation Stratégie de réitération
	Participant	Stratégie de proposition
FIPA English Auction	Initiateur	Produit Timeout Stratégie d'évaluation d'enchère
	Participant	Stratégie d'enchère
FIPA Dutch Auction	Initiateur	Produit Timeout Stratégie d'évaluation d'enchères
	Participant	Stratégie d'enchère
FIPA Request	Initiateur	Action
	Participant	Stratégie de proposition
FIPA Request When	Initiateur	Action Condition d'exécution
	Participant	Stratégie de proposition
FIPA Query	Initiateur	Requête
	Participant	Stratégie de proposition
FIPA Propose	Initiateur	Action
	Participant	Stratégie d'évaluation

Bartolini et al. [BPJ02]. Dans cette ontologie, un *protocole* de négociation possède un *processus* générique pouvant être spécialisé par le biais d'un ensemble de propriétés (la dépendance, le nombre de phases, etc.). Le protocole fait intervenir un ensemble d'agents participants, décrit par le concept *Party*. les rôles chargés de contrôler le déroulement de la négociation sont également décrit dans l'ontologie. Le protocole concerne un *objet de négociation* qui décrit l'objet matériel ou immatériel transféré après accord entre les agents négociants. Le concept *Offer* définit les propriétés et les contraintes que doivent vérifier les offres faites par les agents participants.

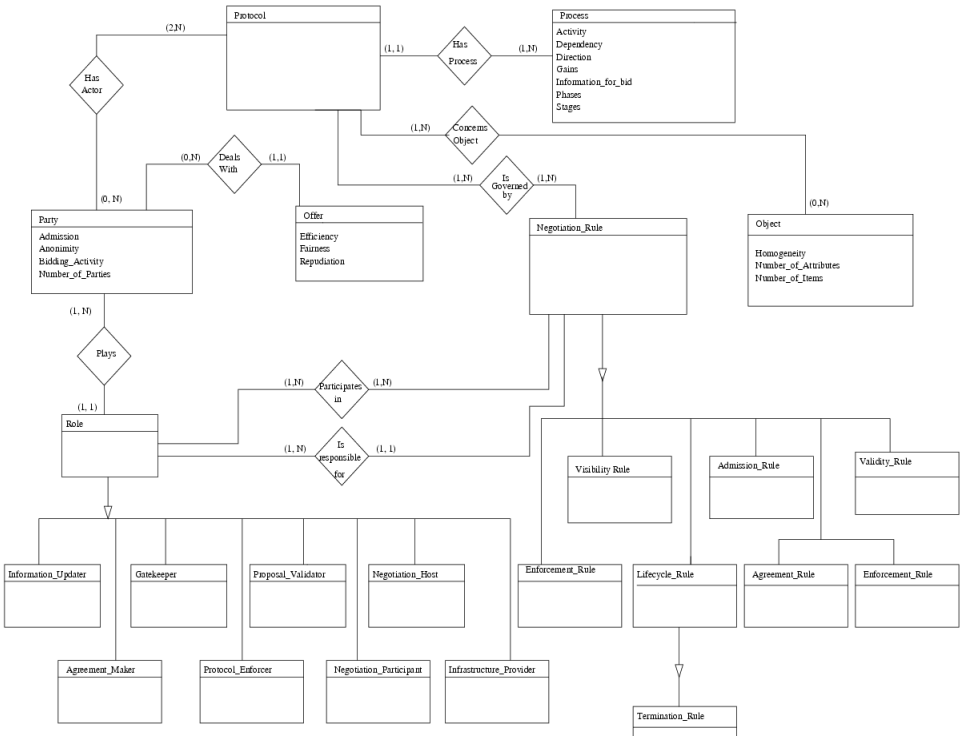


FIG. 4.7 – L'ontologie de négociation : issue de [TWD02]

Nous partageons le point de vue des auteurs sur la nécessité de définir une ontologie des protocoles, pour offrir plus de flexibilité aux agents dans le choix du protocole à utiliser pour leur négociation. Cependant, et contrairement à leur approche de représentation de l'ontologie, qui s'intéresse principalement au processus d'exécution de la négociation, nous utilisons l'ontologie pour représenter les concepts, qui serviront de paramètres pour instancier des rôles d'interaction, modélisés sous forme de composants. Par ailleurs, Tamma et ses collègues se sont intéressés seulement aux protocoles de négociation, alors que nous essayons, à travers notre ontologie, de représenter les protocoles d'interaction, qui incluent les protocoles de négociations.

4.5.1.2 Une ontologie des protocoles d'interaction avec DAML

Toivonen et Helin [TH03] proposent une représentation en DAML des protocoles d'interaction. Ils s'inspirent du travail de Freire et Botelho [FB03] (voir la section 2.8.4 du chapitre 2). La figure 4.8 présente les principaux concepts de l'ontologie du protocole d'interaction.

Les actions de l'agent sont divisées en actions générales et en actions du domaine. Ces dernières peuvent être distribuées dans des répertoires externes, que les agents peuvent télécharger (i.e. leur description), pour modifier leurs comportements.

Le protocole d'interaction contient les informations sur l'initiateur et le ou les participants. Chaque protocole possède un composant, qui définit le processus d'interaction du protocole (i.e. le concept *Progress*). Ce composant contient un ensemble d'états (i.e. *State*) qui constituent le flux du protocole. Une propriété *timeout* est associée à chaque *State*. Chaque état peut avoir plusieurs points de décisions, appelés *Choices*, qui donnent accès à plusieurs options. Les messages reçus ou envoyés sont indiqués au niveau des options. Un message contient un identifiant, l'expéditeur et le destinataire.

Selon ses auteurs, cette ontologie est assez générale, pouvant être étendue pour supporter toute communication de haut niveau, tels que la communication inter-agents par *FIPA-ACL* ou entre des services web par *WSCL*. Ils donnent une extension de leur ontologie pour la représentation des protocoles de FIPA. La principale extension de cette ontologie concerne le concept *Message*, qui permet de représenter les différents types de messages de *FIPA-ACL*.

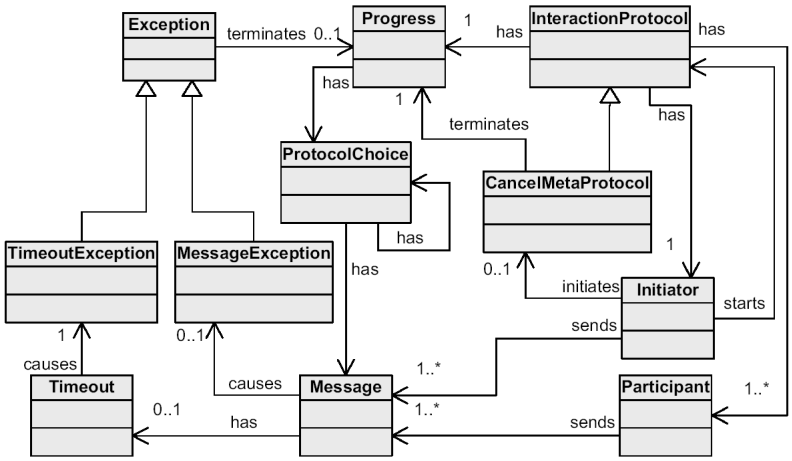


FIG. 4.8 – Une ontologie pour les protocoles d'interaction

Similaire au travail de Tamma et al. [TWD02], cette ontologie s'intéresse à la représentation du processus d'interaction.

4.5.1.3 Des ontologies statiques et dynamiques pour les protocoles d'interaction

Cranefield et Purvis [CP03] proposent une ontologie propre à chaque protocole d'interaction. Chaque ontologie regroupe les concepts et les actions que le protocole utilise. Les éléments de l'ontologie sont les types des messages, et les concepts que les messages utilisent. La figure 4.9 décrit une partie de l'ontologie relative au protocole *Request*.

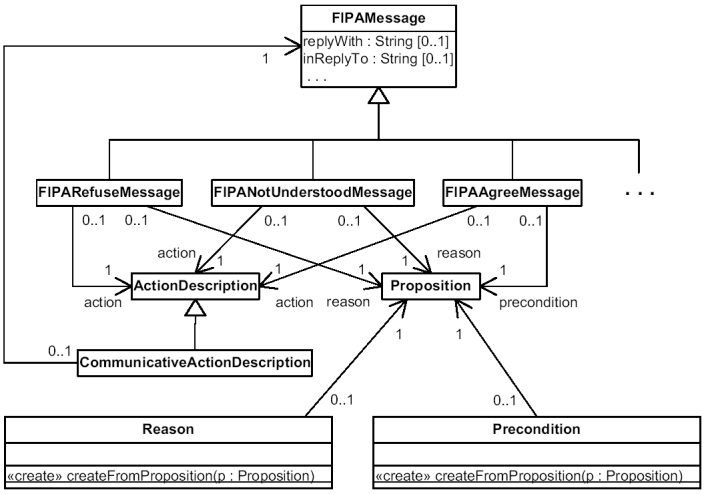


FIG. 4.9 – Une partie de l'ontologie relative au protocole *Request*

Les auteurs font la distinction entre les actions génériques présentes dans tous les protocoles d'interaction et les actions spécifiques à chaque protocole. Pour définir les actions de l'ontologie d'un protocole donné, les auteurs proposent deux approches : *statique* et *dynamique*.

L'approche *statique* représente chaque rôle par une classe stéréotypée *«Role»* qui encapsule, sous forme de méthodes, toutes les actions nécessaires à son exécution (voir figure 4.10). Pour pouvoir exécuter le rôle, il faut associer à chaque méthode du rôle une opération de l'agent. Cette représentation peut être vue comme une interface de programmation entre la spécification abstraite du rôle et les opérations internes de l'agent. Le mapping (opération, action) doit se faire d'une manière statique. Cela est une limite pour cette approche, car les agents ne peuvent pas changer dynamiquement de rôles d'interaction.

L'approche *dynamique* réifie toutes les opérations du rôle d'interaction (voir figure 4.11). Une opération est une classe stéréotypée, qui hérite de la classe abstraite *Operation*. Chaque opération est associée à un rôle, et est liée à deux contextes, relatifs aux états locaux du monde avant et après l'exécution de l'opération. Les paramètres d'entrée et de sortie sont décrits par des associations, avec les étiquettes *in* et *out*.

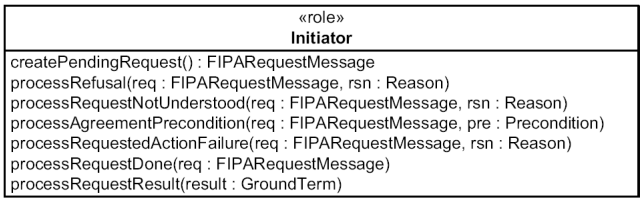


FIG. 4.10 – L'approche *statique* : spécification du rôle *initiateur FIPA-Request*

Pour exécuter une opération d'un rôle donné, l'agent invoque la méthode `execute` de la classe *OpExecutor*. Les arguments de l'opération sont stockés dans un objet de la classe *Binding*, qui fait le *mapping* entre les types des arguments d'une opération et les objets. Les résultats de l'opération sont retournés dans un objet *Binding*.

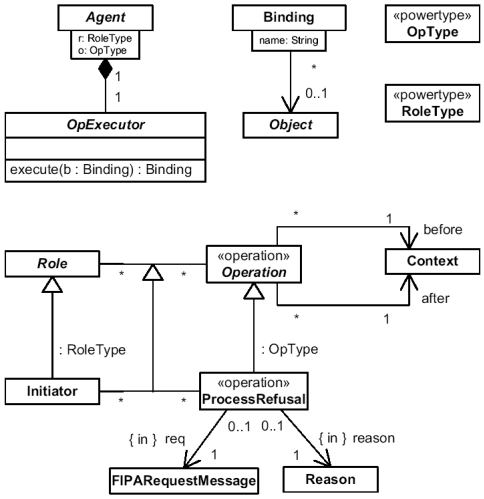


FIG. 4.11 – L'approche *dynamique* : réification des opérations du rôle

Cette approche donne plus de sémantique aux actions du rôle, grâce aux relations entre l'opération et les concepts qu'elle prend comme arguments. Ceci facilite la tâche de l'agent à faire l'appariement entre ses actions internes et les opérations du rôle. Par contre, l'approche dynamique est un peu contraignante, car elle augmente énormément le nombre de concepts de l'ontologie d'un protocole. Par exemple, l'ontologie du rôle initiateur du protocole *FIPA-Request* compte 15 concepts.

4.5.1.4 Synthèse

Selon les travaux, cité avant, nous constatons que la meilleure solution pour faciliter la réutilisation des protocoles d'interaction et de les décrire d'une manière explicite, par exemple en XML ou en DAML dont le schéma correspond à l'ontologie du protocole. Les ontologies présentées dans cette section ont en commun le concept de role, qui constitue un concept important pour une représentation réutilisable des protocoles d'interaction.

Les ontologies [TWD02] et [TH03] sont bien adaptées pour la conception des protocoles d'interaction. La définition d'un protocole d'interaction nécessité une description de l'ensemble de ses concepts. Ces ontologies fournissent les concepts nécessaires à la description du processus d'interaction, tels que le message, l'action et la règle. Cependant, elles ne se préoccupent pas de la réutilisation dynamique des protocoles d'interaction.

L'ontologie [CP03] réifie les informations et les actions du protocole pour permettre leur appariement avec les actions et les données spécifiques de l'agent, facilitant ainsi la réutilisation du protocole. Dans ce travail, les auteurs définissent une ontologie propre à chaque protocole d'interaction. Par exemple l'ontologie du protocole *FIPA-Request* compte quinze concepts. Le nombre des concepts de l'ontologie dépend de l'activité interactive du protocole. La limite de cette approche est le grand nombre de concepts de l'ontologie, qui est un peu contraignant pour la spécialisation des protocoles, surtout si l'agent prévoit d'utiliser différents protocoles d'interaction.

4.5.2 ONTOPRO : une ONTOlogie pour les PROtocoles d'interaction

Notre approche pour la définition de l'ontologie peut être vue comme une combinaison de l'approche statique et l'approche dynamique pour la représentation des actions, proposées par Cranefield et al [CP03]. Les deux principaux concepts de notre ontologie sont : le service et la stratégie. On retrouve ces deux concepts dans la définition donnée par Jennings et al. [JFL$^+$00] pour la négociation. La négociation possède trois aspects [JFL$^+$00] : le protocole de négociation, l'objet de négociation et les modèles décisionnels des agents. Dans notre ontologie, l'objet de négociation correspond au concept *Service* et les modèles décisionnels sont les stratégies décisionnelles des agents (voir figure 4.12).

4.5.2.1 Service

Les actes de communication de FIPA-ACL expriment :

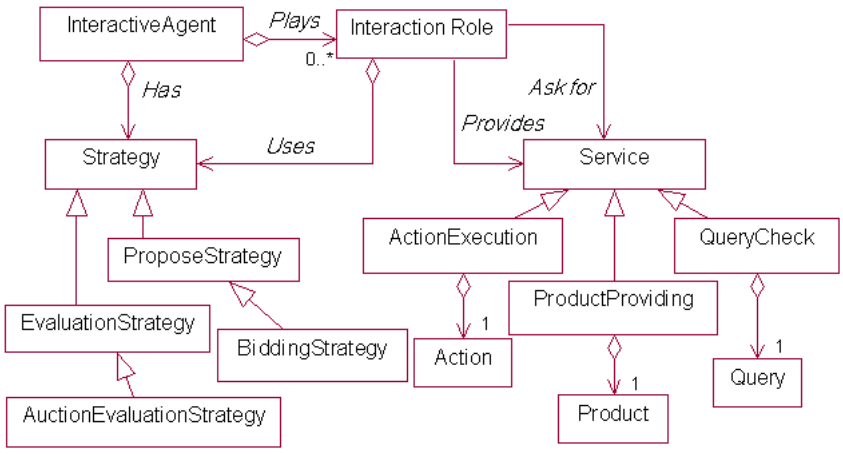


FIG. 4.12 – les principaux concepts de ONTOPRO

- des demandes : *Cfp*, *Request*, *Query*, *Subscribe*,
- des propositions : *Propose*, *Agree* ou
- des informations : *Inform-done*, *Refuse*, *Reject-proposal*, *Accept-proposal*, *Failure*, *Confirm*, *Disonfirm*.

Ces actes de communication peuvent concerner l'exécution d'une action, la vente d'un produit, la vérification d'un prédicat, etc. Puisque, le protocole d'interaction est une séquence d'actes de communication, alors nous appelons le sujet sur lequel porte l'interaction l'*objet d'interaction*. L'objet d'interaction est toujours défini par le rôle initiateur. Le tableau 4.2 donne l'objet d'interaction de chaque protocole d'interaction de FIPA.

TAB. 4.2 – Les objets d'interaction dans les protocoles d'interaction de FIPA

Protocole d'interaction	**L'objet d'interaction**	**Les inputs du rôle initiateur**
FIPA-Contract-Net	Exécuter une tâche	Une tâche
FIPA-Request	Exécuter une tâche	Une tâche
FIPA-Query	Vérifier une proposition	Une proposition
FIPA-Request-when	Exécuter une tâche avec des conditions	Une tâche, des conditions
FIPA-English-Auction	Acheter un produit	Un produit
FIPA-Dutch-Auction	Acheter un produit	Un produit
FIPA-Propose	Exécuter une tâche	Une tâche

Vu cette diversité dans les objets d'interaction, nous avons choisi le concept de *Service* comme une solution d'unification. Dans les protocoles *Contract Net*, *Request* et *Propose*, le service est l'exécution d'une action, avec d'éventuelles conditions, spécifiée par l'initiateur du protocole. Dans les protocoles d'enchère, le service est l'achat du produit. Nous présentons dans

ce qui suit les représentations existantes du concept de service dans le domaine des systèmes multi-agents.

Dans FIPA [fIPA04], le service est décrit par : un nom, un type, un ensemble de protocoles d'interaction supportés par le service, une liste d'ontologies supportées par le service, les langages de description supportés par le service et le nom de l'agent qui possède le service et la liste des propriétés du service (voir figure 4.13).

Frame Ontology	service-description fipa-agent-management			
Parameter	**Description**	**Presence**	**Type**	**Reserved Values**
name	The name of the service.	Optional	string	
type	The type of the service.	Optional	string	fipa-df fipa-ams
protocols	A list of interaction protocols supported by the service.	Optional	Set of string	
ontologies	A list of ontologies supported by the service.	Optional	Set of string	fipa-agent-management
languages	A list of content languages supported by the service.	Optional	Set of string	
ownership	The owner of the service	Optional	string	
properties	A list of properties that discriminate the service.	Optional	Set of property	

FIG. 4.13 – La description d'un service dans FIPA [fIPA04]

Dans Gaia [ZJW03], un service est défini comme un bloc cohérent d'activités auquel un agent s'engage. Un service peut être un produit à vendre ou à acheter, une action à exécuter, une information à donner, etc [WJK00].

Dans leur modèle de négociation, Jennings et al. [JFN+00] définissent le service comme une activité manuelle ou automatique que l'agent doit assurer. *"Here we use the term service to denote activities (manual or automated) that an agent can manage. A service corresponds to a conceptual unit of problem solving activity in the business process. Examples of services include designing an artefact, providing an insurance quote for a customer, or reviewing a paper for a scientific journal"* [JFN+00]. Le service est ainsi considéré comme une fonction qui prend des paramètres d'entrée, entreprend des traitements, et produit un ensemble de résultats. Un simple service est une tâche qui permet de résoudre un problème donné. L'exécution de cette tâche peut être manuelle ou automatisée. Ces unités atomiques (i.e. tâches) peuvent être composées pour former des services plus complexes, en leur ajoutant des contraintes d'ordonnancement et des contraintes de contrôle [JFN+00].

A partir de ces définitions, nous remarquons que le concept de service est quasiment décrit de la même manière. Ainsi, dans notre ontologie, chaque service possède un nom et des attributs descriptifs. Nous avons ajouté à l'ontologie trois services prédéfinies, utilisés par les protocoles de FIPA : *ActionExecution*, *ProductProviding* and *QueryCheck*. Dans les protocoles FIPA, les

services sont spécifiés par le rôle initiateur. D'autres concepts sont utilisés pour la représentation d'un service, tels que : le produit, l'action et la requête.

4.5.2.2 Stratégie

Certaines actions décisionnelles sont utilisées par plusieurs rôles d'interaction, tels que l'action d'évaluation de propositions qu'on trouve dans le rôle initiateur de *FIPA-Contract-Net*, le rôle *Participant-Propose* et le rôle *Initiateur-Contract-Net*. Cette réutilisation nous a amené à réifier l'ensemble des actions décisionnelles, que nous appelons *des stratégies*. Le concept de stratégie fait partie des principaux éléments pour la modélisation de la négociation en multi-agents. Plusieurs travaux se sont intéressés à la modélisation des stratégies dans la négociation [San99] [Kra01]. Tamma et al. [TWD02] définissent une stratégie : *"An agent's key task is to employ a negotiation strategy that maximises its welfare : the strategy is essentially the agent's program, which defines how it behaves during negotiation"*.

Cependant, nous ne retrouvons pas le concept de stratégie dans les travaux sur les ontologies de l'interaction, que nous avons présenté dans la section 4.5.1. Toivonen et Helin [TH03] parlent de *ProtocolChoice* qui correspond à un point de choix entre plusieurs performatives que le rôle peut envoyer à un moment donné de l'interaction. Mais un *ProtocolChoice* ne fait pas automatiquement appel à une stratégie. Par exemple le *ProtocolChoice* des deux performatifs *Accept-Proposal* et *Reject-Proposal* du rôle *Initiateur-Enchère-Anglaise* est associé à une action générique qui consiste à accepter la première proposition reçue et de rejeter le reste.

Par contre, le rôle *participant-Enchère-Anglaise* possède une action décisionnelle permettant de définir une proposition, suite à la réception du *Cfp*. Pour exécuter cette action le participant fait appel à une stratégie de l'agent, que nous appelons *Stratégie de construction de proposition*. Cette stratégie se base sur les informations relatives à l'état courant de l'interaction (i.e. la dernière proposition acceptée, la liste des participants, le nombre d'itérations), ainsi que sur des données privées de l'agent (i.e. ses compétences, ses préférences, ses croyances, son historique d'interaction, etc).

L'analyse des protocoles de FIPA, nous a permis de définir une taxinomie des stratégies. Nous distinguons deux grandes classes : (i) la *stratégie d'évaluation* permet de choisir une proposition de service parmi l'ensemble des propositions reçues, et (ii) la *stratégie de proposition* est utilisée par le rôle pour définir une proposition pour service donnée.

Sierra et al. [SFJ00] proposent un modèle de stratégie de proposition dans le contexte de la négociation. Une stratégie est une combinaison de plusieurs tactiques. Les auteurs définissent un ensemble de tactiques dont chacune est associée à un seul critère.

Sur la base de cette ontologie, nous avons pu spécifier les paramètres d'entrée des principaux protocoles d'interaction de FIPA [FIP02b]. Ainsi, les protocoles d'interaction peuvent être définis comme des composants réutilisables.

4.6 Conclusion

En effet, l'analyse des protocoles d'interaction de FIPA nous a permis de réifier les principaux concepts de l'interaction. Ainsi chaque rôle d'interaction est décrit comme un composant logiciel avec des paramètres d'entrée. Notre ontologie ONTOPRO donne une typologie des paramètres d'entrée des rôles d'interaction. Cette représentation componentielle facilite l'instanciation et la réutilisation des rôles d'interaction.

Nous allons décrire dans le chapitre suivant notre framework INAF pour le développement des agents interactifs. Ce framework est une extension de la plate-forme DIMA et utilise ONTOPRO.

Chapitre 5

INAF : un framework d'agents interactifs

5.1 Introduction

Les systèmes multi-agents offrent de nombreux avantages pour la conception et le contrôle des systèmes complexes, cependant leur développement reste difficile. Les difficultés de développement inhérentes aux systèmes multi-agents, sont la conséquence de la diversité et de la complexité des concepts d'agent et des mécanismes de coordination, d'interaction, d'organisation, etc. Cette complexité fait que l'utilisation de la plupart des outils et des plates-formes existants est difficile aux non-spécialistes en multi-agents.

Dans cette thèse nous nous intéressons aux systèmes multi-agents dont l'interaction se base sur les protocoles d'interaction. Le développement de ces systèmes nécessite une bonne maîtrise des protocoles d'interaction et de tous les détails de leur utilisation. Le développeur doit connaître le langage de communication à utiliser, tels que *FIPA-ACL*, l'enchaînement des messages et la spécification de leur contenu. De ce fait, l'implémentation des protocoles d'interaction est complexe, de même pour l'implémentation des agents qui les utilisent.

Pour palier ce problème, nous avons conçu et implémenté le framework *INAF* (*INteractive Agent Framwork*), qui offre une bibliothèque de protocoles d'interaction implémentés d'une manière générique, et propose une architecture d'agents interactifs et adaptatifs. Ces agents ont la capacité de changer dynamiquement de protocoles d'interaction. *INAF* est une extension de la plate-forme *DIMA* [GB99b] qui est caractérisée par une approche modulaire pour le développement des agents.

Ce chapitre est voué à la présentation d'*INAF*. Nous commençons par donner un aperçu de l'architecture générale d'*INAF*. Nous présentons ensuite, dans la section 5.3, l'architecture d'agents interactifs. La section 5.4 décrit l'implémentation des protocoles d'interaction de *FIPA* et leur réutilisation. Finalement, nous présentons l'implémentation des deux benchmarks : la gestion des emplois du temps et la gestion d'une chaîne logistique.

5.2 Aperçu du framework *INAF*

Le framework *INAF* est basé sur une architecture modulaire d'agents interactifs, une bibliothèque de composants réutilisables et une ontologie d'interaction (voir figure 5.1) :

- L'architecture d'agent fournit les composants de base et une architecture minimale d'agent interactif que le développeur réutilise pour développer ses agents. pour faciliter le développer des agents.
- La bibliothèque contient un ensemble de protocoles d'interactions dont les rôles sont implémentés sous forme de composants. Cette bibliothèque est gérée par un agent *Manager*.
- L'ontologie contient une spécification des concepts de l'ontologie d'interaction, en l'occurrence les des deux concepts service et stratégie. Cette ontologie permet de définir les paramètres d'entrée des rôles d'interaction de la partie `InteractionRoleLibrary`. Par exemple le rôle *participant* du *FIPA-English-Auction* a besoin d'une stratégie de proposition. Les concepts de cette ontologie sont aussi utilisés pour définir la structure des agents interactifs. Par exemple, un agent vendeur doit définir l'ensemble des services qu'il propose, ainsi que ses stratégies de vente, tels que la stratégie d'évaluation, s'il veut utiliser le rôle *initiateur* du *FIPA-Contract-Net*.

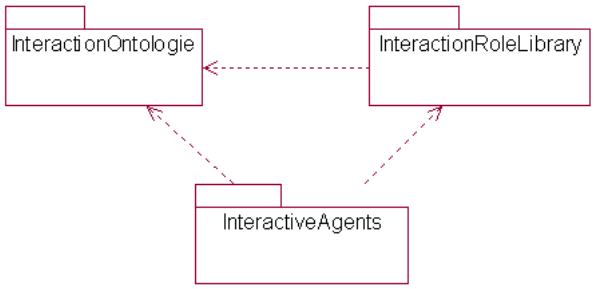


FIG. 5.1 – La structure d'INAF

Pour la construction d'un agent interactif, le développeur réutilise l'architecture de base de l'agent interactif de INAF, et les rôles d'interaction que son agent va jouer. L'initialisation de l'agent et des rôles d'interaction se base sur les concepts de l'ontologie d'interaction. Nous commençons par décrire l'architecture d'agent interactif.

5.3 Architecture d'agents interactifs

Similaire à *DIMA* [GB99b], *INAF* adopte la même approche de développement des agents, celle de la modularité. L'approche modulaire offre plusieurs avantages, tels que :

- avoir une granularité variable des agents,
- pouvoir définir des agents avec une structure adaptative ; chaque agent peut changer dynamiquement ses composants et leurs relations,
- la possibilité d'exploiter une bibliothèque de composants réutilisables.

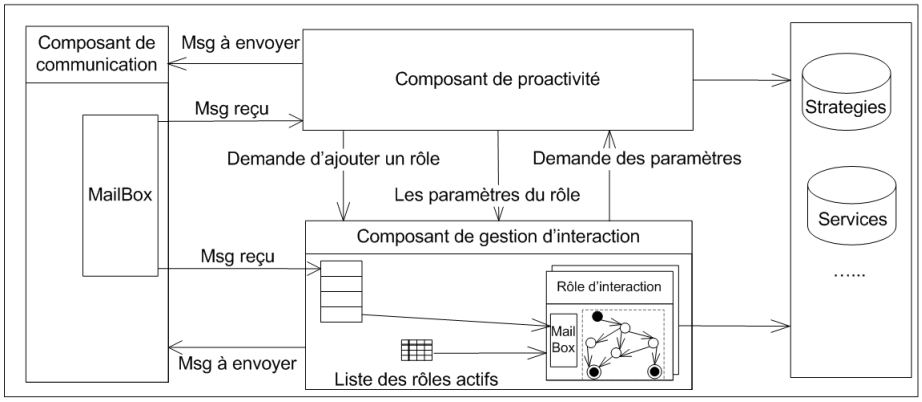


FIG. 5.2 – architecture de l'agent interactif

Ainsi, nous proposons une architecture modulaire d'agents interactifs. C'est une architecture minimale de l'agent interactif qui contient les trois principaux composants (figure 5.2) : communication, interaction et proactivité.

5.3.1 Composant de communication

Le composant de communication (voir figure 5.3) fournit les comportements de base de communication, qui sont l'acheminement des messages reçus vers les composants qui doivent les traiter, et l'envoi des messages reçus par les composants vers les agents destinataires. Les messages sont décrits avec le langage de communication de *FIPA-ACL* [FIP02a].

Dans INAF, la communication est asynchrone. En effet, le composant de communication encapsule la boîte aux lettres de l'agent. Il analyse les messages reçus pour déterminer le rôle d'interaction correspondant. Cette analyse est basée sur l'attribut *Conv_Id* contenu dans les messages *FIPA-ACL* et sur la liste des rôles actifs de l'agent. Le *Conv_Id* permet d'identifier une interaction parmi l'ensemble des interactions en cours d'exécution. Cet attribut est défini par l'initiateur de l'interaction, généralement il commence par l'identifiant de l'agent, suivi par

un numéro séquentiel. Si le *Conv_Id* du message reçu est initialisé, alors le message est transmis au composant d'interaction. Sinon, le message est traité par le composant de proactivité.

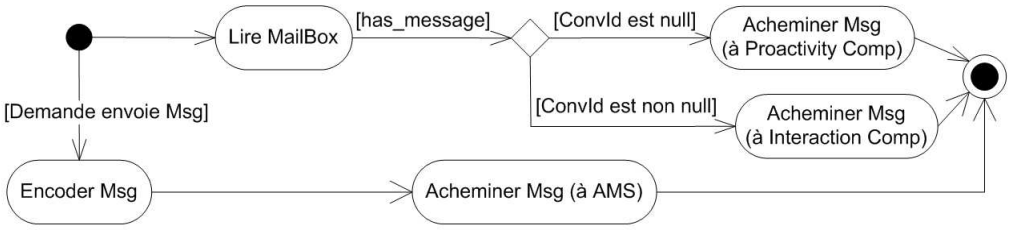


FIG. 5.3 – L'activité du composant de communication

5.3.2 Composant d'interaction

Ce composant gère les rôles d'interactions de l'agent en cours d'exécution. Dans le cas où l'agent joue un seul rôle par interaction, le *Conv_Id* est suffisant pour retrouver le rôle d'interaction. Cela n'est plus valable, si l'agent joue plusieurs rôles dans une même interaction, dans le cas où le protocole utilisé compte plus de deux rôles d'interaction. Pour ce faire, nous avons ajouté au *Conv_Id* le nom du rôle d'interaction. En effet, le rôle d'interaction est une entité active, identifiée par l'attribut *Conv_Id* et son nom.

L'activation du rôle nécessite l'exécution de l'automate correspondant. Ce dernier est régi par un interpréteur de l'automate, qui se base sur l'état de rôle et les événements reçus. Les rôles d'un même agent sont exécutés de façon concurrente.

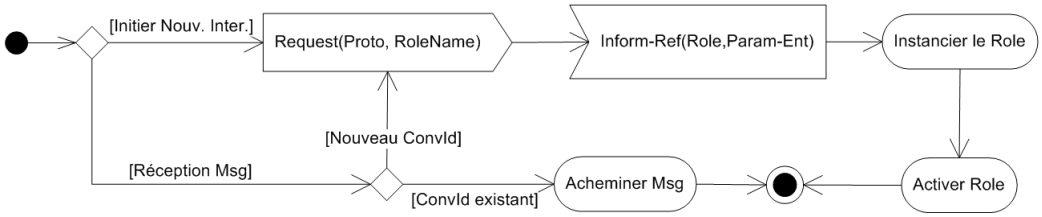


FIG. 5.4 – L'activité du composant de gestion d'interaction

La figure 5.4 décrit l'activité du composant d'interaction. Les deux principales activités sont : (i) l'initialisation d'une nouvelle interaction et (ii) le traitement des messages reçus concernant de nouvelles interactions ou des interactions en cours.

L'initialisation d'une nouvelle interaction est entreprise, suite à une demande du composant de proactivité. Cette demande contient le nom du protocole et le nom du rôle à jouer dans la nouvelle interaction. Pour faciliter l'instanciation des rôles, *INAF* propose des paramètres par

défaut pour le *Time Out* et pour certaines stratégies. Par exemple, si l'agent veut initialiser le rôle *initiateur* de *FIPA-Contract-Net* mais il n'a pas une stratégie d'évaluation de propositions, alors il peut utiliser la stratégie par défaut, qui choisit la première proposition reçue.

Si le composant d'interaction reçoit une demande pour participer à une nouvelle interaction, par exemple suite à la réception d'un message *Cfp* avec un nouveau *Conv-Id*, alors le composant d'interaction instancie le rôle participant du protocole indiqué dans le message *Cfp* reçu, et ajoute une nouvelle ligne (*Conv_Id*, *nom_Role*) dans la liste des interactions. Nous prenons comme hypothèse, que la participation de l'agent à une nouvelle interaction est automatique tant qu'il possède le rôle et les compétences nécessaires pour interagir. Si l'agent échoue dans la récupération ou l'instanciation du rôle, alors il envoie un message *Not-understand* à l'initiateur, pour l'informer qu'il abandonne l'interaction. Le message contiendra le *Conv-Id* que l'agent a reçu dans le premier message *Cfp*.

A la fin de l'exécution du rôle, le composant d'interaction en informe le composant de proactivité. Ce dernier récupère l'état final du rôle d'interaction (i.e. échec ou succès) et les éventuels résultats obtenus par le rôle. Sur la base de cette information, l'agent peut entreprendre certaines actions, tels que l'enregistrement du résultat de ses interactions ou la décision de s'engager dans une nouvelle interaction. Ensuite, le composant d'interaction arrête le processus associé au rôle et retire sa référence (*Conv-Id*, *nom_Role*) de la liste des interactions en cours.

5.3.3 Composant de proactivité

Le noyau de notre agent interactif est le composant de proactivité qui représente une entité autonome. Ce composant est décrit par un framework qui inclut un ensemble minimal de classes et de méthodes qui définissent les principales fonctionnalités du composant de proactivité. Ce framework se compose principalement de la classe *ProactiveComponent*. Une instance de *ProactiveComponent* décrit [Gue03] :

- le but du composant de proactivité, il est implicitement ou explicitement décrit par la méthode *isAlive*,
- les compétences de base du composant de proactivité. Une compétence est une séquence d'actions qui permettent de changer l'état interne de l'agent, de traiter un message ou d'envoyer un message à d'autres composants. Chaque compétence est implémentée par une méthode Java,
- le comportement, qui définit comment les compétences sont choisies, ordonnancées et lancées.

Le Tableau 5.1 décrit les principales méthodes de la classe `ProactiveComponent`.

TAB. 5.1 – Les principales méthodes de la classe `ProactiveComponent`

Méthodes	Description
public abstract boolean isAlive()	teste si le composant de proactivité a atteint son but.
public abstract void step()	représente un cycle d'activité du composant de proactivité.
void proactivityLoop()	décrit le comportement du composant de proactivité. public void proactivityLoop() { while (this.isAlive()) { this.preActivity() ; this.step() ; this.postActivity() ;}}
public void startUp()	Initialise et active le comportement. public void startUp() { this.proactivityInitialize() ; this.proactivityLoop() ; this.proactivityTerminate() ;}

Le comportement de l'agent décrit dans ce composant lui permet de prendre des décisions de contrôle. Il fournit à l'agent un mécanisme d'auto-contrôle dynamique pour adapter ses compétences pour être en accord avec son état interne et sa représentation de l'environnement.

Dans ce composant, le comportement est décrit d'une manière procédurale. Il est implicitement représenté par la méthode abstraite *step*. Différents formalismes déclaratifs (*ATN*, règles de production...) sont alors employés pour représenter le comportement de l'agent. De nouvelles sous-classes de *ProactiveComponent* avec ces formalismes ont été donc présentées. Par exemple, le comportement d'un *ATNBasedProactiveComponent* est modélisé par un *ATN*. Les états correspondent aux différents états du composant de proactivité et les actions correspondent aux comportements de l'agent. La méthode *step* est dans ce cas générique. Elle exécute une étape de l'interpréteur de l'*ATN*.

Le comportement décisionnel nécessaire pour un rôle d'interaction est géré par le composant de proactivité. Pour initier une nouvelle interaction, le composant de proactivité commence par choisir le protocole d'interaction à utiliser. Dans notre travail, nous ne nous intéressons pas au

choix du meilleur protocole qui optimise les résultats de l'agent. Le choix du protocole relève d'un niveau décisionnel propre à l'agent et qui ne fait pas partie du niveau interactif.

Parmi les compétences offertes par ce composant, nous citons celles qui sont relatives à l'interaction :

- demander l'ajout d'un nouveau rôle d'interaction *add_ new_ Role(protocol_ name, role_ name)*,
- activer d'un rôle d'interaction *activate_ Role(role_ name, conv_ id)*,
- demander l'arrêt d'un rôle d'interaction *stop_ Role(protocol_ name, role_ name)*,
- sélectionner les paramètres d'entrée d'un rôle d'interaction *Select_ Input_ Param(role_ name, list_ input_ type_ param)*.

5.3.3.1 Adaptation de l'interaction

Dans le contexte de l'interaction, l'adaptation est la capacité de l'agent à changer de rôles d'interaction. Dans INAF, nous tirons profit de la représentation réutilisable des rôles d'interaction décrite dans le chapitre [**?**], pour faciliter l'adaptation du comportement de l'agent. Nous distinguons deux modes de réutilisation des rôles d'interaction : statique et dynamique.

Grâce au mode de **réutilisation statique**, le développeur peut décrire facilement l'activité d'interaction de son agent. Pour ce faire, le développeur doit spécifier les paramètres d'entrée du rôle pour l'utiliser. En effet, chaque classe de rôle possède un constructeur qui récupère l'ensemble des paramètres d'entrée. Voici la signature du constructeur du rôle *Initiateur* du *FIPA-Contract-Net* :
`InitiatorContractNet(InteractiveAgent agentOwner, String convId, Vector participantAgents, AbstractService serv, Date timeout, EvaluationStrategy strg)`.
Ainsi, le développeur peut décrire d'une manière explicite la dynamique de l'interaction de l'agent, par exemple sous forme de règles. La partie de gauche de la règle contient des conditions sur l'état de l'agent et de ses interactions en cours, et la partie droite contient les actions d'instanciation, d'activation, d'arrêt des rôles d'interaction.

Le deuxième mode est celui de **la réutilisation dynamique** qui permet à l'agent de changer de rôle d'interaction, ou d'acquérir un nouveau rôle, sans interrompre l'exécution de l'agent. Ce mode offre plus de flexibilité au comportement interactif de l'agent. L'agent ne se limitera pas aux protocoles spécifiés par le développeur avant son activation. Il peut, par exemple, ajouter un nouveau rôle d'interaction pour participer à une nouvelle interaction, au lieu de la quitter, car il ne possède pas le rôle d'interaction adéquat.

Pour ce faire, l'agent commence par demander à l'agent *Manager* la description du nouveau rôle d'interaction (cette étape est décrite dans la section 5.4). Ensuite, et après réception du

rôle, l'agent sélectionne les valeurs des paramètres d'entrée pour instancier le rôle par le biais de la méthode *selectionInputParameters*. Cette méthode doit être définie par le développeur pour indiquer ses choix décisionnels. Une simple implémentation de cette méthode serait, par exemple :

```
If(Type_Parameter=="EvaluationStrategy")
ParameterList.add(new MyEvaluationStrategy) ;
```

Pour décrire un comportement plus complexe, cette méthode peut faire appel, par exemple, à une base de règles.

5.4 Bibliothèque des Protocoles d'Interaction

INAF propose une bibliothèque de protocoles d'interaction. L'intérêt principal de la spécification de cette bibliothèque, est de faciliter l'implémentation des agents interactifs. Cette bibliothèque est gérée par un agent *Manager* qui transmet les rôles aux agents. Ces derniers utilisent le protocole *FIPA-Query-Ref* pour communiquer avec le *Manager*.

La figure 5.5 décrit l'interaction entre un agent A et le *Manager* pour la récupération du rôle *participant* du *FIPA-Contract-Net* . Pour ce faire, l'agent A commence par envoyer un message *Query-ref* contenant le nom du protocole et le nom du rôle. Si le rôle existe dans la bibliothèque, alors le *Manager* envoie un message *Inform-ref* contenant une instance de ce rôle (i.e. un objet de la classe du rôle) et une liste des types de paramètres d'entrée de ce rôle. Sinon, l'agent *Manager* envoie un message *Reject* à l'agent A. Le *Manager* donne aussi pour certains types de paramètres d'entrée, tels que les stratégies et l'attribut *TimeOut*, des instances par défaut.

Afin de clarifier la notion de rôle d'interaction nous introduisons quelques éléments de son implémentation. Dans INAF, chaque rôle est implémenté par une classe qui étend la classe *AbstractInteractionRole*. Cette dernière est une spécialisation de la classe *ATN* de la plateforme *DIMA*. Les principaux attributs d'un rôle sont :

- le nom du rôle et le nom du protocole,
- l'identifiant de la conversation, qui est défini par le rôle initiateur au début de l'interaction.
- une boîte à messages propre au rôle,
- l'état final de l'interaction est un attribut qui peut avoir la valeur *"succès"* ou *"échec"*. Cette information est retournée au composant de proactivité après la fin de l'interaction. Sur la base de cette information, l'agent peut entreprendre certaines actions, tels que l'enregistrement du résultat de ses interactions ou la décision de s'engager dans une nouvelle interaction.

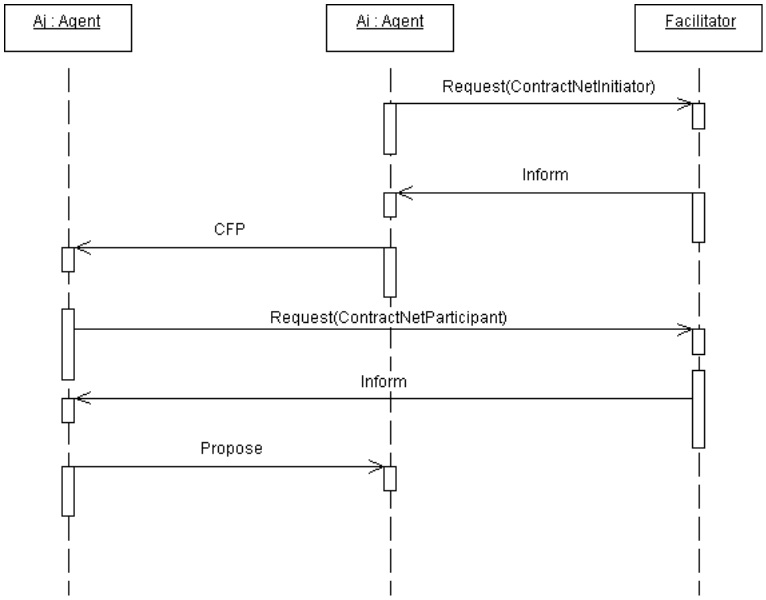


FIG. 5.5 – L'ajout de nouveaux rôles

En ce qui concerne le comportement, tous les rôles possèdent des méthodes pour manipuler les attributs précédemment cités. Parmi lesquelles, nous citons la méthode *getInputParameterTypes* qui retourne le type des paramètres d'entrée du rôle. Par ailleurs, chaque rôle possède un ensemble de méthodes qui correspondent aux différents comportements du rôle. A titre d'exemple, le rôle participant du *FIPA-Contract-Net* possède les méthodes suivantes :

- *decideToPropose* fait appel à la stratégie de proposition définie comme paramètre d'entrée du rôle,
- *performContract* exécute le contrat conclu avec l'initiateur,
- *sendPropose* envoie la proposition à l'initiateur,
- *sendRefuse* envoie un message *Refus* à l'initiateur,
- *sendInformDone* envoie un message *InformeDone* à l'initiateur pour l'informer de l'exécution du contrat.

5.5 Application : Gestion des emplois du temps

Nous proposons de réaliser le benchmark [BGGP03], *"Gestion des emplois du temps"*, du groupe de travail *ASA* du *GdR I3* [1]. Il s'agit d'un système multi-agents qui aide les utilisateurs

[1] http ://www-poleia.lip6.fr/ guessoum/asa.html

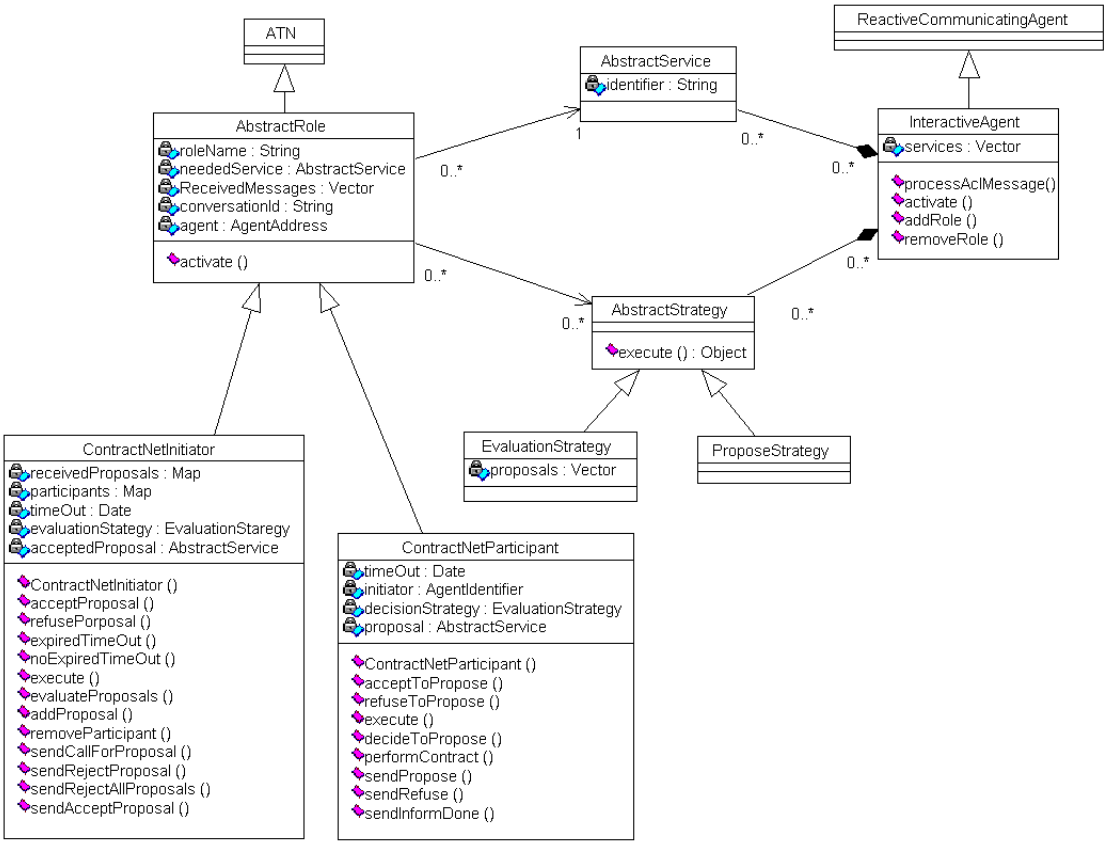


FIG. 5.6 – Les principales classes de INAF

(enseignant et groupe d'étudiants) à fixer leurs séances de cours. Chaque utilisateur possède un agent assistant qui gère son emploi du temps. Cet agent interagit avec :

- l'utilisateur pour obtenir ses demandes de séances décrites par un jour de la semaine et un créneau horaire et un intitulé du cours,
- les autres agents du système pour prévoir des séances.

La conception d'une application avec *INAF*, nécessite la conception du domaine d'application et des agents du système comme une extension de l'ontologie de l'interaction. Cette application contient deux classes d'agents : *Enseignant* et *Groupe Étudiants*.

5.5.1 Le domaine

Le domaine comprend les concepts : emploi du temps, créneau horaire et séance (voir figure 5.7). Chaque agent (enseignant et groupe d'étudiant) possède son propre emploi du temps. Pour simplifier la gestion des emplois du temps, on considère que l'emploi du temps est composé de

créneaux horaires d'une même durée (2 heures). Ainsi la journée contient 4 créneaux horaires : de 8h à 10h, de 10h à 12h, de 14h à 16h, et de 16h à 18h. Au moment de l'initialisation, l'agent peut avoir des contraintes sur certains créneaux.

Le but des agents est d'interagir pour réserver des créneaux libres en tenant compte de leurs disponibilités (i.e. les séances déjà fixées et les contraintes sur les créneaux). Ainsi, l'action de reservation d'un créneau horaire pour une séance donnée, est implémentée comme une spécialisation de la classe *Action* de l'ontologie d'interaction, voir figure 5.7.

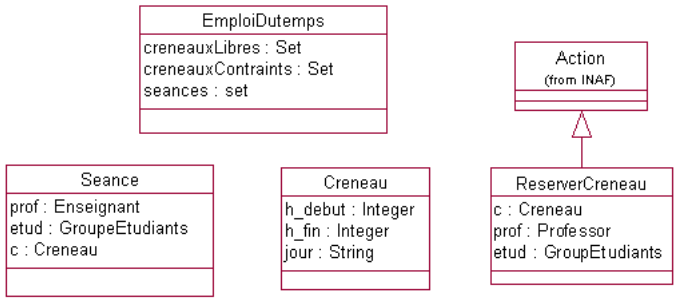


FIG. 5.7 – Les classes du domaine de l'emploi du temps

5.5.2 Les agents

Dans cette application, un agent peut participer simultanément à plusieurs interactions. D'où, le besoin d'avoir des agents capables de gérer plusieurs interactions en concurrence et d'ajouter dynamiquement de nouveaux rôles d'interaction.

Plusieurs scénarios sont envisageables pour développer cette application. Par exemple, nous proposons le scénario suivant :

- les agents *Enseignant* commencent par initier une interaction avec les agents *Groupe* pour définir une séance,
- les enseignants jouent le rôle *initiateur* du protocole *FIPA-Contrat-Net* pour soumettre un *CFP (Call For Proposal)*, demandant une réservation d'un créneau horaire pour une séance donnée, aux différents groupes,
- de leur côté, les groupes d'étudiants répondent aux enseignants, en utilisant le rôle *participant* du *FIPA-Contrat-Net*.

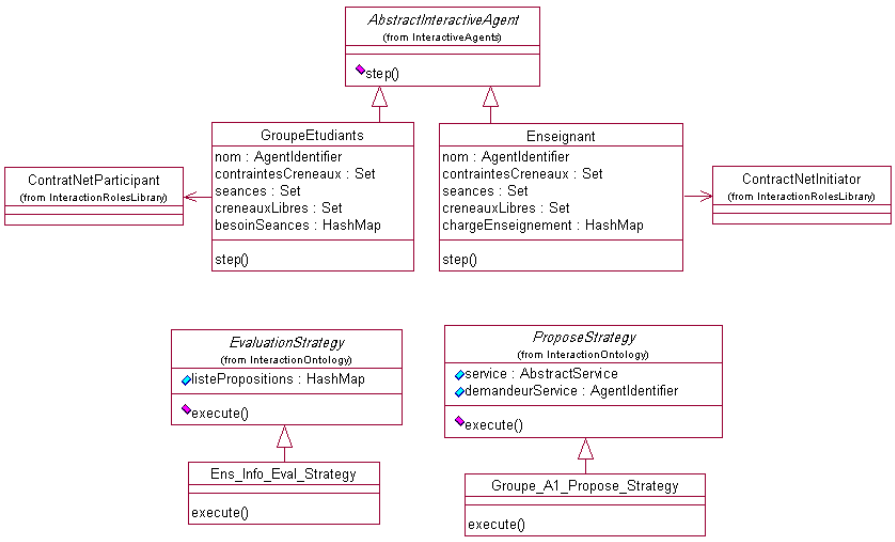


FIG. 5.8 – Les agents de l'emploi du temps

5.5.2.1 L'agent *Enseignant*

Le but de chaque agent enseignant est d'assurer sa charge d'enseignement, en fixant les séances qu'il doit avoir avec l'ensemble des groupes d'étudiants. L'ensemble des séances à assurer, est spécifié dans l'attribut *chargeEnseignement* qui est une liste de couples (identifiant de l'agent groupe, nombre de séances). Par exemple nous pouvons imaginer que l'enseignant *Ens_Info* doit fixer deux séances avec chaque groupe d'étudiants du système. Pour un système contenant quatre groupes, l'attribut *chargeEnseignement* contient quatre entrées. A chaque fois que l'enseignant *Ens_Info* fixe une séance, il retire l'instance correspondante de la liste *chargeEnseignement*. En fait, l'enseignant est toujours actif tant que la liste *chargeEnseignement* contient des éléments.

Selon notre scénario, l'*Enseignant* doit instancier et activer le rôle initiateur de *FIPA-Contract-Net* pour déterminer chaque séance de son emploi du temps. Ainsi, le comportement d'un enseignant peut être décrit par les règles suivantes :

- Si *chargeEnseignement* n'est pas vide, alors initialiser et activer le rôle *Initiator* de *FIPA-Contract-Net*.
- Si *chargeEnseignement* est vide, alors suspendre l'agent *Enseignant*.

Ce comportement doit être décrit dans sa méthode *Step*. Le code *Java* nécessaire pour l'implémentation de ce comportement est le suivant :

```
IF (!chargeEnseignement.isEmpty())
```

```
//commencer par définir les paramètres d'entrée
//de l'initiateur de "FIPA-Contract-Net"
   { Vector param = new Vector() ;
   param.add(new ReserverSeance(new Seance(8,10,"Lundi"))) ;
   param.add(new Ens_Info_Eval_Strategy()) ;
//ensuite instancier du rôle
   InteractionRole role=new InteractionRole() ;
   this.initialiseRole(role,param) ;
//enfin activer le rôle
   this.activateRole(role) }
ELSE this.setActive("false") ;
```

En résumé, l'implémentation de l'agent *Enseignant* nécessite :

- l'initialisation de son emploi du temps, en spécifiant les éventuelles contraintes sur les créneaux, et sa charge d'enseignement,
- la définition de son comportement, tel que, les deux règles précédemment définies,
- la spécification des paramètres d'entrée des différents rôles d'interaction à jouer, tels que la classe *Ens_Info_Eval_Strategy*, qui correspond à la stratégie d'évaluation pour rôle initiateur de *FIPA-Contract-Net*.

Avec *INAF*, le développeur peut changer facilement les rôles de l'agent, à condition d'avoir les bons paramètres pour l'instanciation du rôle. Donc, nous réduisons considérablement l'effort du développement, car les principales modifications concernent seulement la méthode *step()*, qui décrit le comportement de l'agent. Nous pouvons imaginer le scénario suivant : si l'enseignant ne parvient pas à obtenir une séance en utilisant le protocole *FIPA-Contract-Net*, alors il peut relancer une nouvelle interaction avec le protocole *Request*. Le nouveau comportement de l'agent *Enseignant* devient :

- Si *chargeEnseignement* n'est pas vide, alors initialiser et activer le rôle *Initiator* de *FIPA-Contract-Net*.
- Si état fin du rôle est échec, alors initialiser et activer le rôle *Initiator* de *FIPA-Request*.
- Si *chargeEnseignement* est vide, alors désactiver l'agent *Enseignant*.

5.5.2.2 L'agent *GroupeEtudiant*

L'agent *GroupeEtudiant* a besoin de définir un certain nombre de séances auprès des enseignants du système. Les besoins en séances sont définis dans l'attribut *besoinSeances* qui est une liste de couples (identifiant de l'agent enseignant, nombre de séances).

Selon le scénario défini plus haut, le comportement du *GroupeEtudiants* se réduit à l'instanciation et l'activation du rôle *participant* du *FIPA-Contract-Net*, suite à la réception d'un message *CFP* d'un *Enseignant*. Ce comportement de participation à une nouvelle interaction est déjà défini dans le comportement de base de l'agent interactif de *INAF*. Le développeur doit spécifier seulement la stratégie de proposition qui est le seul paramètre d'entrée de ce rôle.

Ainsi, l'implémentation de l'agent *GroupeEtudiants* nécessite :
- l'initialisation de son emploi du temps, en spécifiant les éventuelles contraintes les créneaux, et ses besoins en séances, et
- la spécification des paramètres d'entrée des différents rôles d'interaction que l'agent peut jouer, tels que la construction de la stratégie de proposition du rôle participant du *FIPA-Contract-Net*.

Le code ainsi implémenté montre bien l'aide qu'apporte le framework *INAF* pour le développement d'agents interactifs.

5.6 Application : Gestion d'une Chaîne logistique

Notre deuxième benchmark concerne la gestion d'une chaîne logistique. Il s'agit d'une simulation de chaînes de montage d'ordinateurs, où les agents assembleurs négocient avec des agents qui stockent des pièces détachées. Il y a six agents *assembleurs* qui fabriquent des ordinateurs en fonction des demandes des clients. Pour réaliser cela, ils doivent négocier avec huit agents *stockeurs* afin d'acheter les pièces détachées qu'offrent ces derniers. A la fin de la négociation, un agent *Banque* se charge d'effectuer le retrait du compte d'un agent assembleur de la valeur des pièces commandées.

Il y a donc deux étapes qui se déroulent l'une après l'autre : la négociation et le paiement. Pour ce faire, chaque agent assembleur interagit avec un ou plusieurs agents stockeurs pour demander les pièces dont il a besoin. Cette demande représente une requête comportant la quantité de pièces à fournir, le type de pièces, ainsi qu'une date de livraison afin de satisfaire les demandes clientes. Les agents stockeurs collectent toutes les demandes envoyées et les traitent en différé, suivant une stratégie devant être définie.

Par la suite, l'agent stockeur qui dispose de ce type de composant retourne à l'agent assembleur une offre. Cette offre contient le prix unitaire de la pièce, la quantité livrable pour cette date. Si son stock est insuffisant, il informe l'assembleur du refus de sa demande.

Si l'agent assembleur accepte cette offre, alors l'offre devient une commande qui sera traitée par l'agent banque (le débit du compte bancaire de l'agent assembleur). Il y a quatre types de composants dont chacun est disponible en deux modèles aux prix différents. Chaque agent stockeur ne peut fournir qu'un type et un modèle de composant.

L'interaction entre l'agent banque et l'agent assembleur représente le scénario de paiement, l'assembleur envoie la somme qu'il doit payer pour sa commande et la banque lui retourne son solde de compte débité de cette somme. Nous allons voir maintenant comment nous avons modélisé ce benchmark.

5.6.1 Le domaine

Nous commençons par décrire l'ontologie du domaine, qui compte trois classes, voir figure 5.9. La classe *PcComponent* décrit les différents pièces qui composent un PC. On trouve l'identifiant du composant et son type.

Nous avons identifié trois actions du domaines : *ProvideComponents*, *TransferAccount* et *SaleComputer*. La première action est utilisée dans l'interaction entre les agents assembleurs et les agents stockeurs. Il est décrit par la classe *ProvideComponents* qui indique la date de livraison, le type de composant et la quantité demandée. La deuxième action est le virement bancaire du compte de l'assembleur au compte du stockeur, après la réception des composants. Il est décrit par la classe *TransferAccount* qui indique les numéros des comptes de l'assembleur et du stockeur, et le montant à verser. La troisième action *SaleComputer* concerne l'opération de vente des PCs assemblés. Ces classes sont une spécialisation de la classe abstraite *Action* de INAF.

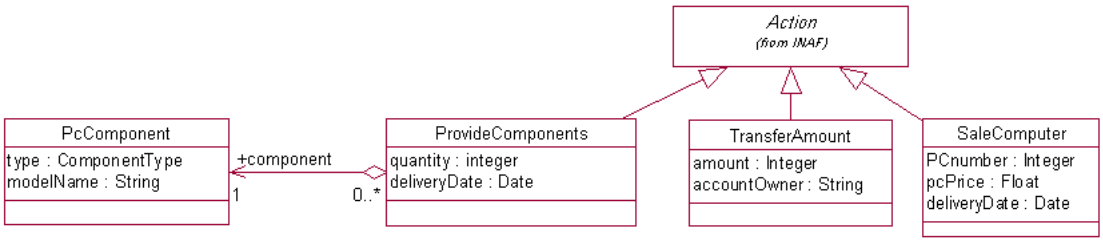


FIG. 5.9 – Les classes du domaine de la chaîne de montage

5.6.2 Les interactions

L'interaction dans *INAF* est basée sur la bibliothèque de protocoles d'interaction de *FIPA*. Dans ce benchmark nous avons mis en oeuvre trois interactions décrites dans le diagramme de séquences de la figure 5.10.

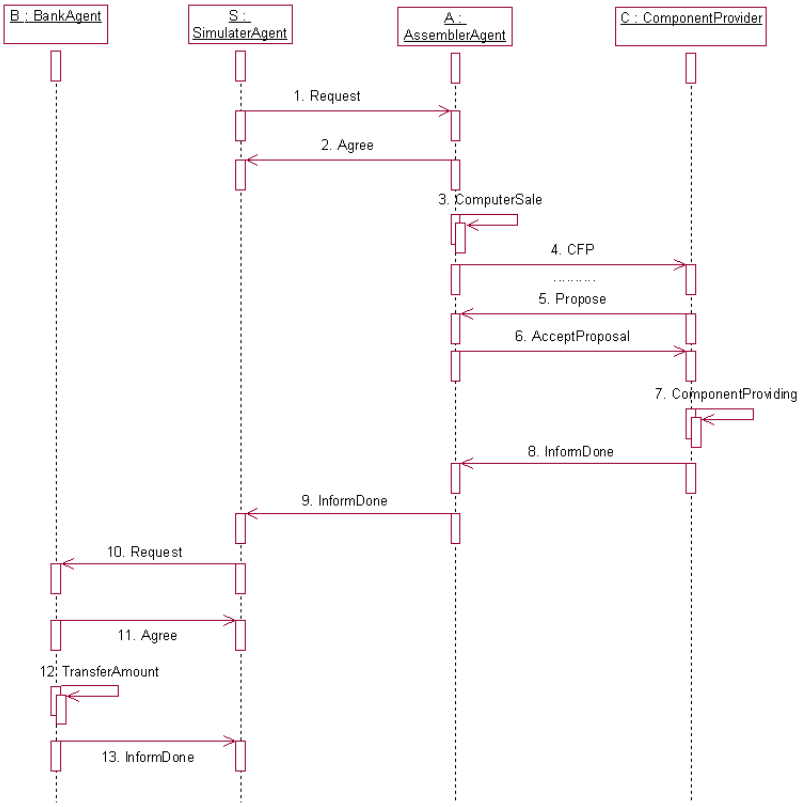


FIG. 5.10 – L'interaction entre les agents

La première interaction entre l'agent client et le(s) agent(s) assembleur(s) permet de démarrer l'activité de la chaîne de montage. L'agent client est l'initiateur de l'interaction, son but est d'acquérir un quantité de PC, en tenant compte du prix et de la date de livraison. Pour ce faire, l'agent client a le choix entre plusieurs protocoles d'interaction : *FIPA-Request*, *FIPA-Contract-Net* ou *FIPA-Iterated-Contract-Net*.

Suite à l'acceptation de la commande de l'agent client, l'agent assembleur initie une interaction avec un ou plusieurs agents assembleurs, pour obtenir les composants nécessaires pour la fabrication du PC. Cette commande est contrainte par le temps de livraison et le prix du composant. L'agent assembleur peut utiliser les protocoles *FIPA-Request*, s'il veut négocier avec un seul agent assembleur à la fois. Comme il peut utiliser le protocole *FIPA-Contract-Net* ou *FIPA-Iterated-Contract-Net*, pour une interaction avec plusieurs agents assembleurs.

Après la réception des composants, l'agent assembleur demande à l'agent banque pour faire un virement sur le compte de l'agent stockeur, le biais du protocole *FIPA-Request*.

5.6.3 Les agents

L'implémentation de l'activité d'interaction des agents demande la spécification des services offerts ou demandés, et des stratégies décisionnelles. Nous présentons dans cette section l'implémentation des agents décrit dans la figure 5.11.

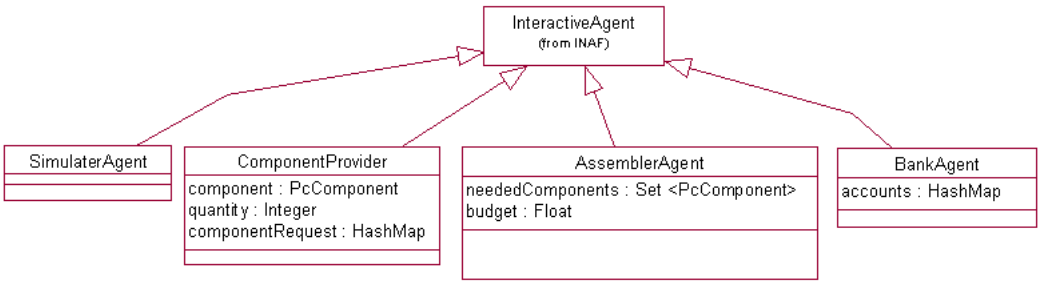


FIG. 5.11 – Les classes des agents de la chaîne logistique

5.6.3.1 L'agent client

L'activité de cet agent est l'initialisation d'une interaction avec chaque agent assembleur avec le protocole *FIPA-Request*. Ainsi, l'agent assembleur instancie le rôle initiateur et l'active. Pour jouer ce rôle, l'agent client doit avoir une description du service *ComputersProviding*, qui contient le nombre de PCs et la date de livraison. Lorsqu'un agent assembleur a honoré sa commande, l'assembleur déclenche la procédure de paiement avec l'agent banque avec le protocole *FIPA-Request*. Le service de cette interaction est *AmountTransfer*.

Ce comportement qui doit être décrit dans sa méthode *Step* est le suivant :

```
//initier l'interaction avec les 6 agents assembleurs
For(i==1 ; i<=6 ; i++)
{
//commencer par définir les paramètres d'entrée de l'initiateur de "FIPA-Request"
 Vector param = new Vector() ;
//le service
 param.add(new ComputerProviding(5,450,"22/10/06")) ;
//l'agent participant
```

```
 param.add(new AgentName(new String("Assembler"+i))) ;
//instancier le rôle
 Initiator-Fipa-Request role = new Initiator-Fipa-Request(param) ;
//activer le rôle
 this.activateRole(role) ;
}
//Si une interaction se termine par un succès alors initier
//l'interaction avec l'agent banque
If(hasEndRoleEvent() and finishedRole.getFinalState() == "success")
{
 integer amount= ((ComputerProviding)finishedRole.getService()).getPrice() ;
 String accountOwner= new String(finishedRole.getParticipant()) ;
 //commencer par définir les paramètres d'entrée de l'initiateur de "FIPA-Request"
 param = new Vector() ;
//le service
 param.add( new AmountTransfer(amount, accountOwner)) ;
//l'agent banque
 param.add(new AgentName(new String("Banque"))) ;
//instancier le rôle
 role = new Initiator-Fipa-Request(param) ;
//activer le rôle
 this.activateRole(role) ;
}
```

En résumé, l'implémentation de l'agent client demande seulement l'initialisation des deux interactions, c'est à dire instanciation des rôles et leur activation.

5.6.3.2 L'agent *Assembleur*

L'activité de l'agent assembleur démarre à la réception du message Request de l'agent client. Alors l'agent assembleur ajoute et instancie le rôle participant du *FIPA-Request*. L'agent assembleur doit avoir une implémentation de l'action *SaleComputers*. Cette action permet d'activer de nouvelles interactions avec les agents stockeurs pour acquérir les composants.

L'agent assembleur joue le rôle initiateur du *FIPA-Contract-Net* pour négocier sur un type donné de composants avec l'ensemble des agents stockeurs. Pour chaque interaction, l'agent définie l'action *ProvideComponents* passée comme paramètre d'entrée au rôle. L'agent assembleur peut initier en simultané plusieurs négociation pour les différents types de composants.

5.6.3.3 L'agent *Stockeur*

Etant donné que l'agent stockeur peut recevoir plusieurs demandes à la fois et qu'il ne peut pas forcément honorer tous les contrats, il doit avoir une vue globale sur la variation de son stock en fonction des commandes passées avec les assembleurs.

Un agent stockeur peut participer simultanément à plusieurs négociations. Il joue le rôle participant du *FIPA-Contract-Net* dans chaque négociation. L'ajout et l'instanciation de ce rôle se fait d'une manière automatique. Le seul paramètre d'entrée de ce rôle est la stratégie de proposition. Dans notre exemple cette stratégie permet de vérifier la quantité en stock pour savoir combien il peut proposer pour la date donnée dans le *CFP*.

5.6.3.4 L'agent *Banque*

L'agent banque est beaucoup plus simple, il se charge juste de la gestion des comptes bancaire des assembleurs. Un compte est un couple (le nom de l'assembleur, le solde de son compte).

Il communique avec l'agent client pour faire des virements sur les comptes des agents assembleurs, par le biais du rôle participant de *FIPA-Request*. Le seul paramètre d'entrée de ce rôle est la stratégie de proposition. Dans cette interaction nous avons ajouté à l'agent banque la stratégie *TransfertDecision* qui permet de vérifier les informations transmises par l'agent client.

L'agent banque joue seulement le rôle de participant, dont l'instanciation et l'activation se font d'une manière dynamique. Donc, il n'existe pas d'activité spécifique à cet agent, qui demande une spécialisation de la méthode *step* de l'agent.

5.7 Évaluation de INAF

L'utilisation de *INAF* facilite le développement des agents interactifs basés les protocoles d'interaction. Pour évaluer cet aspect, nous avons comparé l'effort nécessaire pour l'implémentation d'agents interactifs avec INAF à l'effort de développement avec une autre plate-forme multi-agents. Deux principaux critères d'évaluation sont le nombre de classes et le nombre de lignes de code, nécessaires pour chaque implémentation.

Nous avons choisi pour notre comparaison la plate-forme *DIMA*. Les éléments d'interaction que supporte *DIMA* sont l'acheminement des messages entre les agents via le *AMS* (*Agent Managment Service*) et l'implémentation des actes de communication de *FIPA-ACL*. DIMA

ne propose aucune implémentation des protocoles d'interaction, ni de comportement pour la gestion des interactions de l'agent.

Le tableau 5.2 donne le nombre de classes et de lignes de code dans *INAF* et *DIMA*, pour les deux benchmarks décrits dans les sections précédentes. D'après cette comparaison, nous développons plus de classes avec INAF, mais dont le code est simple, puisqu'il se base sur les classes de base (i.e. agent, protocoles et ontologie) de INAF. Par contre le développement des agents interactif avec DIMA reste *ad-hoc*, moins de classes et plus de lignes de code.

TAB. 5.2 – Comparaison selon les deux benchmarks

	NB de classes	**NB de lignes de code**
La gestion des emplois du temps		
INAF	5 (*Enseignant*, *GroupeEtudiants*, *ReserverCreneau*, *EnsEvalStrg*, *GroupProposStrg*)	215
DIMA	2 (*Enseignant*, *GroupeEtudiants*)	590
La gestion d'une chaîne de montage de PC		
INAF	7 (*Client*, *Banque*, *Stockeur*, *Assembleur*, *ProvideComponents*, *TransferAccount*, *SaleComputer*)	300
DIMA	5 (*Client*, *Banque*, *Stockeur*, *Assembleur*)	1530

Vu que le nombre de messages échangés dans un protocole d'interaction, augmente la complexité de son implémentation, nous pouvons généraliser notre évaluation selon le type des protocoles d'interaction utilisé par application, c'est à dire suivant le nombre de messages échangés dans le protocole. Le tableau 5.3 compare le développement d'un agent utilisant différents protocoles d'interaction avec *INAF* et *DIMA*.

A partir du tableau 5.3, nous constatons que l'effort de développement (le nombre des classes et des lignes de code) augmente d'une façon importante quand on incrémente le nombre de messages du protocole. Un autre avantage d'INAF, qui paraît évident, est celui de la réutilisation du code des protocoles d'interaction. Ainsi, INAF facilite la définition du comportement interactif de l'agent, mais il facilite également la spécification d'autres protocoles d'interaction en se basant sur les protocoles de la bibliothèque. Par exemple, le protocole *FIPA-Iterated-Contract-Net*

D'où, l'intérêt d'utiliser un framework de développement d'agents interactifs, tels que *INAF*, qui offre une bibliothèque componentielle des protocoles d'interaction.

TAB. 5.3 – Comparaison selon le type du protocole d'interaction

	Nombre de classes	**Nombre de lignes de code**
Un agent avec *FIPA-Request* (5 messages)		
INAF	4 (*InteractiveAgent, EvaluationStrategy, ProposeStrategy, Service*)	180
DIMA	1 *ATNBasedCommunicativeAgent, Service*	545
Un agent avec *FIPA-English-Auction* (6 messages)		
INAF	3 (*InteractiveAgent, ProposeStrategy, Service*)	155
DIMA	1 *ATNBasedCommunicativeAgent, Service*	585
Un agent avec *FIPA-Contract-Net* (7 messages)		
INAF	4 (*InteractiveAgent, EvalStrategy, ProposeStrategy, Service*)	180
DIMA	2 *ATNBasedCommunicativeAgent, Service*	615

5.8 Conclusion

Ce chapitre a présenté notre framework INAF qui facilite le développement des agents interactifs utilisant les protocoles d'interaction de FIPA. INAF est une extension de la plate-forme DIMA. La principale caractéristique d'INAF est son architecture modulaire qui facilite l'ajout des rôles d'interaction. INAF offre une bibliothèque de protocoles d'interaction de FIPA, dont les rôles d'interaction sont implémentés sous forme de composants réutilisables. Ces protocoles d'interaction partage une même ontologie qui facilite leur réutilisation dans différentes applications.

Nous avons montré à travers l'implémentation de deux benchmarks, la facilité de développement qu'offre INAF, grâce à la propriété de réutilisation de notre bibliothèque de protocoles d'interaction. Cependant pour développer un système multi-agents avec INAF et DIMA, on doit connaître toutes les classes de la bibliothèque (classes d'agent, classes de comportement, classes composantes de communication, classes de message...). Par ailleurs l'utilisation de ces classes nécessite la maîtrise des concepts d'agent et de multi-agents. Ces difficultés de développement sont dues à la diversité et à la complexité des concepts d'agent et de multi-agents (coordination, interaction, organisation...). Pour réduire cette complexité nous proposons une nouvelle démarche de développement multi-agents à base de modèles, que nous présentons dans le chapitre suivant.

Chapitre 6

Vers une approche d'ingénierie à base de modèles

6.1 Introduction

Dans l'état actuel de la recherche et du développement dans le domaine multi-agents, nous trouvons des contributions sur les architectures d'agents [RSBP95], sur des plates-formes de développement des systèmes multi-agents [DLC87] et sur des méthodologies d'analyse et de conception orientée agent [CP02] [BPG$^+$04] [PG04] [ZJW03] [FG98]. Les architectures d'agent comprennent les composants fonctionnels d'un agent individuel et l'organisation de ces composants afin de faire émerger un comportement cohérent. Des méthodologies ont été introduites pour analyser des problèmes complexes et développer des systèmes multi-agents. Cependant, la plupart des méthodologies ne fournissent pas un processus complet de développement (analyse, conception, implémentation, déploiement...). De plus, elles ne se basent pas sur les architectures d'agent et des outils de développement existants. De récents travaux présentent des méta-modèles de certaines méthodologies représentants les principaux concepts et leurs relations. Bien qu'une unification de ces méta-modèles ait été proposée par Bernon et al. [BCG$^+$04], l'utilisation de ces méta-modèles reste très difficile et leurs domaines d'application sont limités. Par exemple, aucun méta-modèle n'offre les concepts nécessaires à la modélisation d'agents cognitifs.

Cette partie de la thèse vise à établir le lien entre les architectures d'agent existantes et les outils de développement, et les méthodologies et les méta-modèles multi-agents. Notre idée est de partir des architectures et des outils existants et d'élaborer des méta-modèles afin de formuler les connaissances nécessaires aux processus de développement. Ainsi, à la différence de la plupart des méthodologies existantes qui sont basées sur une approche déscendante, notre conception

est ascendante. Cette nouvelle approche est inspirée de l'approche Model Driven Architecture (MDA), proposée par OMG, qui vise à séparer la logique d'application des technologies sous-jacentes pour améliorer le processus de réutilisabilité et de développement [Cas98]. L'idée est que la connaissance conceptuelle devrait être permanente, tandis que les soucis techniques sont généralement de courte durée et limités à une technologie donnée [orm01].

MDA est fondée sur plusieurs niveaux de modèles, principalement des modèles indépendants des plates-formes (PIM) et des modèles spécifiques aux plates-formes (PSM). Le système est représenté d'une perspective logique par un ensemble de PIMs. Ces derniers sont alors transformés en PSMs qui sont les pierres de base nécessaires pour générer le code. Ainsi, au lieu de programmer, le développeur se concentre sur la logique conceptuelle de l'application.

Ce chapitre illustre l'utilisation de MDA pour définir une nouvelle méthode de développement des systèmes multi-agents, appelée *MDAD* (*Model Driven Agent Development*). *MDAD* est basée sur une bibliothèque de méta-modèles correspondants à différents aspects des systèmes multi-agents (i.e. Agent, Organisation, interaction, etc.). Ces méta-modèles peuvent être employés par le concepteur pour décrire facilement les modèles multi-agents qui sont automatiquement transformés pour produire un système multi-agent. *MDAD* a été expérimentée à l'aide du framework INAF. Cependant, AMMDP pourrait être appliquée à d'autres plates-formes multi-agents, car la méthode présentée est générique.

Pour illustrer notre méthode nous choisissons un ensemble de modèles et de méta-modèles que nous appliquons sur l'exemple de la gestion des emplois du temps. Cet exemple correspond à un système multi-agents réparti qui aide des professeurs et des groupes d'étudiants à déterminer leur emploi du temps (voir section 5.5 du chapitre 4). Nous présentons ainsi les modèles qui ont été présentés et ceux qui ont été produits.

Ce chapitre est organisé comme suit : La section 2 donne un aperçu de l'approche MDA. La section 3 présente notre méthode de développement *MDAD*. L'objet de la section 4 est la définition des méta-modèles du niveau PIM et leur utilisation pour décrire l'application de la gestion des emploi du temps. La section 5 décrit un exemple de méta-modèles spécifique au framework INAF. La section 6 analyse le problème de transformation et donne les règles de transformation.

6.2 L'approche MDA

L'approche MDA propose un cadre méthodologique et architectural pour le développement de systèmes qui assure la pérennisation des architectures métier en les découplant des préoccu-

pations technologiques. Ainsi, l'approche MDA se focalise d'abord sur les fonctionnalités et le comportement d'une application, sans se préoccuper de la technologie avec laquelle l'application sera implémentée. L'implémentation de l'application se fait par le biais des transformations des modèles métiers en modèles spécifiques à une plate-forme cible.

Les avantages recherchés par cette approche sont :
- la pérennité du savoir-faire. Ainsi il n'est pas nécessaire de recommencer la modélisation des fonctionnalités et du comportement du système chaque fois qu'une nouvelle technologie est adoptée,
- les gains de productivité, afin de permettre au développeur de réduire les coûts de mise en oeuvre des applications nécessaires à son métier, et
- la prise en compte des plates-formes d'exécution, permettant ainsi le déploiement d'une même application sur plusieurs plates-formes.

Nous présenterons dans cette section l'approche générale de MDA ainsi que les éléments qui la constituent. Pour plus d'ample d'informations sur l'approche MDA et sur son application, vous pouvez consulter le guide du standard MDA publié par l'OMG [orm01]. Pour la rédaction de cette section nous nous sommes basé sur le livre de Blanc [Bla05] et celui de Kadima [Kad05].

6.2.1 Les différents modèles

Un modèle est une spécification formelle de la fonction, de la structure et/ou du comportement d'une application ou d'un système [orm01].

Un méta-modèle est une description formelle de tous les concepts d'un langage spécifiant la syntaxe et la sémantique des modèles d'un domaine paticulier. Un méta-modèle peut être décomposé en deux parties distinctes : La terminologie et les assertions [Kad05]. La terminologie est l'ensemble des concepts, leurs propriétés et leurs relations. Les assertions sont des règles supplémentaires permettant de contraindre l'utilisation des éléments introduits par la terminologie (concepts, propriétés et relations).

L'approche MDA préconise l'utilisation massive des modèles dans les différentes phases du cycle du développement d'une application. Plus précisément, MDA préconise l'élaboration de modèles d'exigences (CIM), d'analyse et de conception (PIM) et d'implémentation (PSM). Dans la suite de cette section, nous présentons avec plus de détails chacun de ces types de modèles.

6.2.1.1 CIM (*Computation Independent Model*)

Un modèle d'exigences est considéré comme une entité complexe, constituée entre autre d'un glossaire, de définitions de processus métier, des exigences et des cas d'utilisation ainsi que d'une vue systémique de l'application. Il est important de noter qu'un modèle d'exigence ne contient pas d'information sur la réalisation de l'application ni sur les traitements.

Avec UML, un modèle d'exigences peut se limiter à un diagramme de cas d'utilisation. Ces derniers contiennent en effet les fonctionnalités de l'application (cas d'utilisation) ainsi que les différentes entités qui interagissent avec elle (acteurs) sans information sur le fonctionnement de l'application. Une fois les exigences modélisées, elles sont censées fournir une base contractuelle assez stable.

6.2.1.2 PIM (*Platform Independent Model*)

Le niveau PIM couvre les phases d'analyse et de conception du cycle de développement. Nous ne considérons donc ici que la conception abstraite, c'est à dire celle qui est réalisable sans aucune connaissance des techniques d'implémentation. L'approche MDA préconise l'utilisation d'UML comme langage pour réaliser des modèles d'analyse et de conception indépendants des plates-formes d'implémentation. De plus, MDA ne donne aucune indication quant au nombre de modèles à élaborer ni quant à la méthode à utiliser pour élaborer ces PIM. Quelques soient le(s) langage(s) utilisé(s), le rôle des modèles d'analyse et de conception est d'être pérennes et de faire le lien entre le modèle d'exigences et le PSM.

6.2.1.3 PSM (*Platform Specific Model*)

Le modèle PSM est, quant à lui, dépendant de la plate-forme technique. Le PSM est une représentation de l'implémentation d'une technologie particulière. Par exemple, PSM EJB (Entreprise Java Bean) contient des termes propres aux EJB tels que *home interface*, *remote interface*, *entity bean*, *session bean*...[Kad05].

MDA considère que le code d'une application peut être facilement obtenu à partir du PSM. Les modèles de code servent essentiellement à faciliter la génération de code à partir du PIM. Il est parfois difficile de différencier le code des applications des modèles de code. Pour MDA, le code d'une application se résume à une suite de lignes textuelles, comme un fichier Java, alors qu'un modèle de code est plutôt une représentation structurée incluant, par exemple, les concepts de boucle, condition, instruction, composant, événement, etc.

Le PSM est essentiellement productif mais il n'est pas forcément pérenne. Pour élaborer des modèles de code, MDA propose, entre autres, l'utilisation de profils UML. Un profil UML est une adaptation du langage UML à un domaine particulier [OMG03].

6.2.2 Métamodélisation en MDA

6.2.2.1 Rôle d'UML dans MDA

L'OMG propose deux approches possibles pour la définition d'un langage spécifique à un domaine donné. La première approche est basée sur la définition d'un nouveau langage. L'OMG propose le langage MOF qui a servi à la définition de la syntaxe et la sémantique d'UML pour la spécification de nouveaux langages orientés objet. La deuxième approche consiste à étendre la notation UML en ajoutant de nouveaux éléments qui sont une spécialisation des éléments prédéfinies d'UML (Classe, Association, Package, etc.). Les mécanismes d'extension sont les stéréotypes, les valeurs primitives et les contraintes. Les extensions UML pour un domaine donné sont regroupées dans un Profil.

L'avantage de la première approche est d'avoir une notation qui fait une correspondance parfaite entre les concepts du langage et les concepts du domaine d'application. Par contre, elle pose un problème d'incompatibilité avec des langages et des outils de développement dont la majorité se base sur l'approche orientée objet et par conséquent ils utilisent la notation UML.

Avec la deuxième approche nous tirons profit des éléments de modélisation d'UML pour représenter les objets et les concepts élémentaires du système. Par ailleurs, les stéréotypes permettent de décrire des parties du système qui sont spécifiques au domaine. Les contraintes du profil portant sur les éléments de modélisation permettent d'exprimer une certaine sémantique de ces éléments. Elles peuvent être un moyen pour la validation des modèles du domaine d'application. Comparée à la première approche, l'utilisation des profils offre plus de facilités.

6.2.2.2 Rôle des Profils UML dans MDA

Les profils UML ont un rôle crucial dans la définition et la transformation des modèles en MDA. Le profil est un mécanisme qui permet de définir une spécialisation d'un méta-modèle pour un domaine d'application particulier. Concrètement, un profil comprend un ou plusieurs des éléments suivants :

– des éléments standard supplémentaires, c'est-à-dire des instances des méta-éléments *Stereotype*, *TaggedValue* et *Constraint*,

- un sous-ensemble de méta-éléments existants pour délimiter la portée du profil,
- des règles supplémentaires définies au moyen de contraintes exprimées en OCL,
- des descriptions en langage naturel de concepts sémantiques supplémentaires,
- des notations graphiques supplémentaires.

6.2.3 Transformation de modèles

Il est important d'établir les liens de traçabilité entre les différents modèles. En fait, MDA établit ces liens automatiquement grâce à l'exécution de transformation des modèles. Ces transformations portent l'intelligence du processus méthodologique pour la construction de l'application. Elles sont stratégiques et font partie du savoir-faire des personnes qui les définissent, car elles détiennent l'expertise du développement d'applications.

Une transformation en MDA est la création d'un modèle cible à partir d'un modèle source et d'éventuelles informations complémentaires conformément à une définition de transformation. Une définition de transformation est un ensemble de règles de transformation qui décrivent globalement comment un modèle dans un langage source peut être transformé en un modèle avec un langage cible.

Une transformation de modèles MDA est une fonction dont les paramètres d'entrée et de sortie sont des modèles structurés par des métamodèles. En effet, une transformation exprime des correspondances structurelles entre les modèles source et cible. Le rôle des métamodèles dans les transformations de modèles MDA est de définir les structures possibles des modèles source et cible et de servir de base à la définition des règles de transformation.

Dans le contexte des modèles UML, les modèles d'entrée et de sortie sont profilés, c'est à dire qu'ils sont étiquetés par des stéréotypes définis par des profils. Du point de vu transformation de modèles, les profils UML sont considérés comme des méta-modèles dans le sens où ils structurent un ensemble de modèles.

La figure 6.1 décrit le mécanisme général de transformation de modèles basé sur la métamodélisation. Les transformations préconisées par MDA sont essentiellement les transformations CIM vers PIM et PIM vers PSM. La génération de code à partir des PSM n'est quant à elle pas considérée comme une transformation à part entière [Bla05]. Avant de présenter les différentes formes de modèles de transformation, nous introduisons un nouveau type de modèle le PDM. Ce modèle peut servir comme paramètre d'entrée pour les modèles de transformation en plus du PIM.

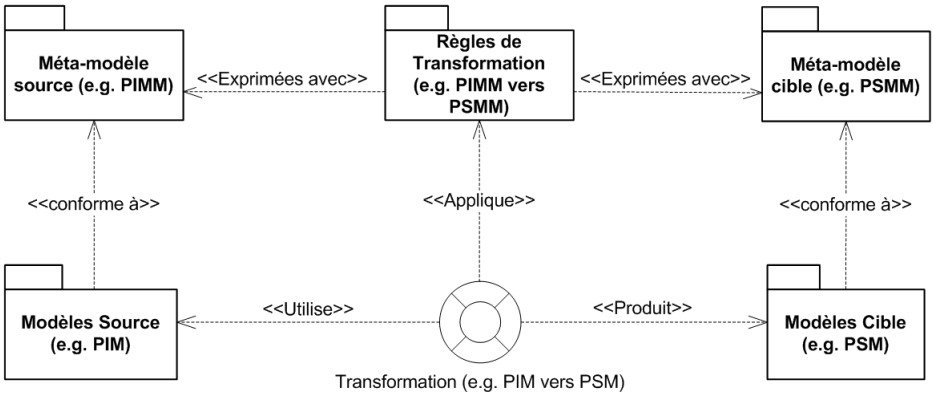


FIG. 6.1 – Processus de transformation piloté par les métamodèles

6.2.3.1 PDM (*Platform Definition Model*)

Dans l'approche MDA, une plate-forme est définie comme une entité technique qui fournit un ensemble cohérent de fonctionnalités grâce à des interfaces. Une application qui s'exécute sur une plate-forme peut bénéficier des fonctionnalités de la plate-forme en utilisant uniquement ses interfaces, sans avoir à connaître les détails d'implémentation de ses fonctionnalités.

Le PDM est un modèle propre à la plate-forme, utile pour la transformation du PIM en PSM. La démarche MDA est ainsi basée sur le détail des modèles dépendant de la plate-forme. Le PDM décrit les différentes fonctionnalités de la plate-forme et précise comment les utiliser. Il est fourni par le créateur de la plate-forme. Généralement ce modèle est sous forme de manuel qui peut être repésenté en UML et/ou OCL.

Le développeur doit choisir une ou plusieurs plates-formes pour l'implémentation du système avec les qualités architecturales désirées.

6.2.3.2 Les techniques de transformation

Pour rendre les modèles productifs il faut procéder à une transformation des modèles abstraits proches des besoins applicatifs en modèles plus concrets proches des plates-formes d'exécution. Quelque soit son type (CIM vers PIM, PIM vers PSM, etc.), une transformation de modèles s'apparente toujours à une fonction qui prend en entrée un ensemble de modèles et qui produit en sortie un autre ensemble de modèles.

Ces modèles sont structurés par leur méta-modèle. C'est d'ailleurs ce qui permet de discerner dans MDA la transformation de modèles de la génération de code. La génération du code permet

de produire des lignes de code sans se baser sur un métamodèle du code. Dans MDA, le passage du PSM vers le code n'est pas considéré comme une transformation de modèle.

Blanc [Bla05] répertorie trois principales approches de transformation de modèles :

1. Approche par programmation : Consiste à utiliser les langages de programmation orientés objet. L'idée est de programmer une transformation de modèles de même manière que l'on programme n'importe qu'elle application informatique. Ces applications ont la particularité de manipuler des modèles. Elles utilisent les interfaces de manipulation de modèles offertes par certains outils et standards de développement tels que JMI de Sun et EMF d'Eclipse. Cette approche est la plus utilisée, car elle est très puissante et fortement outillée.
2. Approche par template : Consiste à définir les canevas des modèles cibles souhaités en y déclarant des paramètres qui seront substitués par les informations contenues dans les modèles sources. Ces canevas sont appelés des "modèles templates".
3. Approche par modélisation : Consiste à appliquer les concepts de l'ingénierie des modèles aux transformations des modèles elles-mêmes. L'objectif est de modéliser les transformations de modèles et de rendre les modèles de transformation pérennes et productifs et d'exprimer leurs indépendances vis-à-vis des plates-formes d'exécution. L'OMG a lancé depuis 2001, un projet pour définir le nouveau langage QVT pour l'interrogation, la visualisation et la transformation de modèles QVT. L'OMG vise à définir le métamodèle MOF 2.0 QVT qui permet de structurer les modèles de transformation de modèles.

MDA est une approche, et non pas une méthode, fondée sur l'utilisation de modèles pour assurer la séparation des préoccupations. Appliquer réellement MDA nécessite inévitablement de définir une méthode. Une telle méthode, propre à un domaine donné, doit identifier les différents modèles ainsi que leurs liens de traçabilité, définir le modèle de transformation à utiliser et son degré d'automatisation. Pour notre part nous avons essayé de tirer profit de l'approche MDA pour proposer une nouvelle méthode pour le développement de systèmes multi-agents, appelée *MDAD*.

6.3 La méthode MDAD

Notre méthode de développement essaie de palier un problème auquel est confrontée la majorité des méthodologies multi-agents, celui de la difficulté du passage du niveau conceptuel au niveau d'implémentation. Elle est fondée sur l'approche MDA, elle vise ainsi à fournir un ensemble de méta-modèles et un modèle de transformation traduisant un savoir-faire dans le domaine du développement multi-agents. Ces méta-modèles sont employés pour décrire des modèles. Ces derniers peuvent être considérés comme les spécifications du système et à ce titre ils constituent les pierres de base utiles pour la construction d'un système multi-agents

opérationnel, qui est obtenu par une succession de transformations réalisées d'une manière automatique ou semi-automatique.

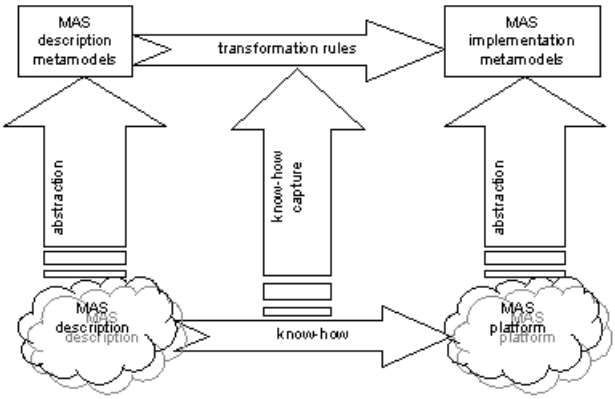


FIG. 6.2 – Elaboration de *MDAD*

Pour l'élaboration de *MDAD* nous avons suivi les étapes suivantes (voir la figure 6.2) :

1. l'identification des différents niveaux d'abstraction : analyse des méthodologies, des méta-modèles et des outils multi-agents pour identifier le PIM et le PSM,
2. la définition d'une bibliothèque de méta-modèles : identification des concepts de chaque niveau d'abstraction et détermination des méta-modèles appropriés,
3. la conception des règles de transformation : Analyse du savoir-faire nécessaire dans le développement du système multi-agents pour définir les règles de transformation.

Quatre niveaux d'abstraction ont été ainsi identifiés, du plus spécifique au plus abstrait :

1. Niveau infrastructure : représente les composants du SMA au niveau de l'implémentation (code) ; il inclut l'implémentation du domaine, l'implémentation générée des agents et le déploiement du système multi-agents,
2. Niveau PSM (platform-specific model) : représente les composants du SMA du point de vue plate-forme de développement. Cette représentation fait abstraction de quelques détails d'implémentation (langage, middlware...). Elle inclut la description des classes d'agent, les constructeurs de comportements, les diagrammes de classes d'initialisation du SMA et des classes de domaine,
3. Niveau PIM (platform-independant model) : représente les besoins du SMA à la phase conceptuelle, indépendamment des plates-formes d'implémentation. L'organisation du SMA et la description des méta-comportements appartiennent à ce niveau,
4. Niveau CIM (computation-independant model) : représente les énoncés d'un problème dans un domaine donné ; de tels énoncés peuvent alors être représentés dans un PIM par une transformation reflétant le savoir-faire des experts du domaine d'application et des concepteurs en multi-agents.

Ce chapitre donne quelques exemples de méta-modèles pour le PIM et le PSM et analyse leur transformation. Le but est de montrer que *MDAD* facilite le développement des systèmes multi-agents.

Pour représenter nos méta-modèles, nous employons les profils UML. Un profil UML étend la notation UML en ajoutant de nouveaux éléments qui sont une spécialisation d'éléments de conception prédéfinis en UML (i.e. Classe, Association, Package, etc.). Un profil UML comprend une ou plusieurs instances des éléments de modélisation suivants : Stereotype, TaggedValue et Constraint.

Nous étendons ainsi la notation d'UML avec deux profils pour représenter le PIM et le PSM. Dans le cadre de l'approche MDA, l'utilisation des profils UML évite l'utilisation de deux notations différentes pour le PIM et le PSM. De plus, elle offre une certaine homogénéité dans la représentation des modèles PIM et PSM. En outre, les stéréotypes du profil faciliteront le repérage des concepts dans les modèles, ainsi que l'application des règles de transformation.

6.3.1 Le processus de développement de *MDAD*

Notre processus de développement des systèmes multi-agents se base sur une approche orientée modèle. Par définition un modèle est une description du système en utilisant un langage qui possède une syntaxe et une sémantique [Kad05].

La sémantique du langage que nous utilisons pour décrire le PIM est donnée par le méta-modèle (i.e. PIMM). Ce dernier donne la sémantique des concepts permettant la définition du SMA à un niveau conceptuel. Le PIM est une représentation assez générique du SMA. L'organisation du SMA et les comportements des agents sont spécifiés à ce niveau.

Pour le niveau PSM nous utiliserons un méta-modèle de la plate-forme (i.e. PSMM) pour décrire l'implémentation du SMA. Le niveau PSM représente les composants du SMA du point de vue plate-forme. Cette représentation fait abstraction de certains détails d'implémentation(langage, middleware...).

Par ailleurs, l'extension de la notation UML basée sur le profil SMA correspond à la syntaxe du langage de spécification du PIM. Cette syntaxe se traduit par l'ensemble des diagrammes d'UML et les règles de notation de chacun d'eux. Nous proposerons aussi un profil pour la plate-forme d'implémentation, dans notre cas le framework INAF.

Nous avons défini un ensemble de règles de transformation assurant le passage du PIM au PSM. Ces règles expriment le savoir faire d'un développeur connaissant la structure du SMA

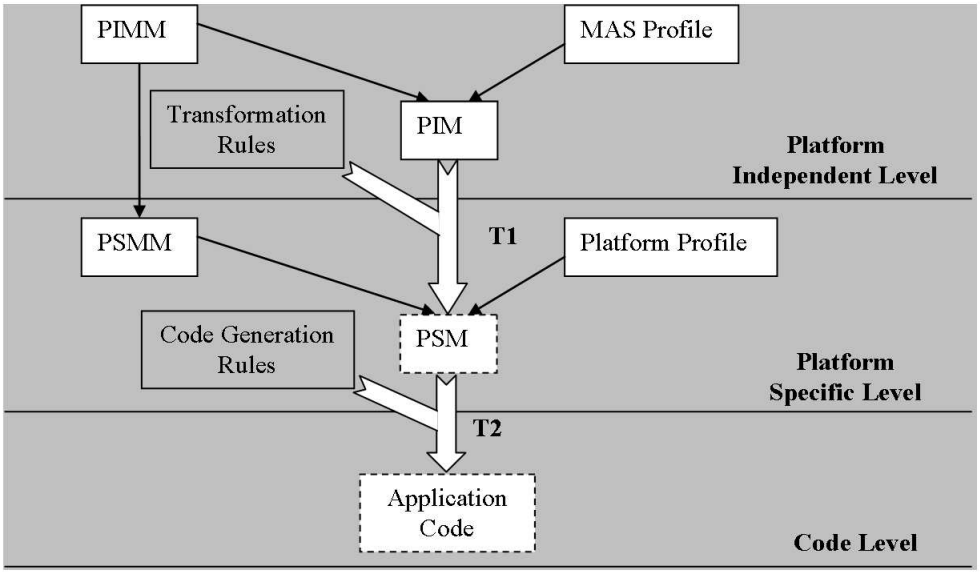


FIG. 6.3 – Le processus de développement de *MDAD*

obtenue à partir de la méthodologie ainsi que la structure et les services offerts par la plate-forme. Les règles de transformation se basent sur les concepts du PIMM et du PSMM.

Le PSM obtenu par application des règles de transformation est réutilisé pour générer le code du SMA. Ce dernier contient la description des classes d'agents, les classes d'initialisation du SMA and les classes du domaine.

Ainsi, le développeur doit suivre un ensemble d'étapes pour obtenir une implémentation de son SMA (voir figure 6.3) :

1. définir le PIM en utilisant la syntaxe donnée par le PIMM et la notation (UML + Profil SMA),
2. appliquer les règles de transformation sur PIM pour obtenir le PSM, et
3. appliquer les règles de génération de code sur le PSM.

Notre approche est assez générique, elle peut être appliquée à plusieurs méthodologies et plates-formes multi-agents. Nous présenterons dans ce chapitre l'application de notre approche sur un ensemble de méta-modèles décrivant le niveau PIM du SMA et une implémentation avec le framework INAF.

6.3.2 Adpatation de la notation UML

Plusieurs chercheurs considèrent que le standard de modélisation objet UML est l'outil de modélisation le plus adapté pour la modélisation des systèmes multi-agents partant du principe qu'un agent est un objet doté d'une autonomie et d'une intelligence et qu'un objet est la forme la plus basique et la plus simplifiée d'un agent. Plusieurs travaux ont vu le jour, préconisant une extension et une adaptation de la notation UML pour la représentation du monde des agents. Un des premiers travaux est celui de J. Odell qui propose avec ses collèges dans [OPB00] une extension des diagrammes des séquences pour la représentation des protocoles d'interaction. Ce travail a était repris par l'organisation agent FIPA pour la spécification de sa bibliothèque de protocoles [FIP02b]. Plusieurs travaux comme ceux de [BP00] [Hug02] [HOB04] ont continué dans cette même voix. Nous citons aussi les travaux de [BP02] [Wag02] qui définissent des profils UML pour la modélisation des systèmes multi-agents.

Dans *MDAD* nous utilisons les diagrammes UML pour décrire les modèles du PIM et du PSM. Les diagrammes de classes décrivent la structure des agents et des rôles, par contre le comportement de l'agent ou du rôle sont décrits par les diagrammes d'activité.

6.4 Les méta-modèles indépendants de la plate-forme

Plusieurs méthodologies ont été proposées pour développer des systèmes multi-agents. Ces méthodologies proposent différents approches et modèles pour faciliter ce développement. Les modèles de chaque méthodologie se base sur un ou plusieurs méta-modèles qui représentent ses principaux concepts et leurs relations.

La méthodologie VOYELLE [Dem95] proposée Y. Demazeau, décrit un système multi-agents par un ensemble de modèles : modèles d'agent, modèles d'interaction, modèles d'organisation et modèles d'environnement. Plusieurs méta-modèles sont en effet définis pour représenter chaque aspect, ainsi le système multi-agents devient une combinaison de ces modèles. Notre approche est fondée sur cette idée. Nous proposons une bibliothèque de méta-modèles : un méta-modèle de domaine, des méta-modèles d'organisation, et des méta-modèles d'agents, des méta-modèles de rôles, etc.

Dans *MDAD*, la conception d'un système multi-agent exige ainsi un modèle de domaine, un modèle d'organisation, et plusieurs modèles d'agent. Nous notons qu'un modèle d'agent peut se baser sur un modèle d'organisation. Les sections suivantes décrivent des exemples de modèles et de méta-modèles.

6.4.1 L'approche VOYELLES

L'approche voyelles est fondée sur la décomposition d'un système multi-agents en quatre éléments : l'Agent, l'Environnement, l'Interaction et l'Organisation. Cette décomposition permet de modulariser le système multi-agents, ce qui permet d'obtenir une simplification de la construction du système et une meilleure réutilisation du code.

Demazeau [Dem95] propose une approche intégrée des systèmes multi-agents basée sur la décomposition de ceux-ci en quatre parties (ou briques) :

1. Agents, qui concernent les modèles (ou les architectures) utilisés pour la partie active de l'agent, depuis un simple automate à un complexe système à base de connaissances.
2. Environnements, qui sont les milieux dans lesquels sont plongés les agents. Ils sont généralement spatiaux dans la plupart des applications multi-agents.
3. Interactions, qui concernent les infrastructures, les langages et les protocoles d'interactions entre agents, depuis de simples interactions physiques à des interactions langagières par actes de langage.
4. Organisations qui structurent les agents en groupes, hiérarchies, relations, etc.

Outre cette décomposition en quatre briques, l'approche Voyelles est guidée par trois principes. Le premier est le principe déclaratif. Comme nous venons de le voir, et d'un point de vue déclaratif, un système multi-agents est composé d'agents, d'environnements, d'interactions, et d'organisations.

SMA = Agents + Environnement + Interactions + Organisations

D'un point de vue computationnel, les fonctionnalités d'un système multi-agents incluent les fonctionnalités individuelles des agents enrichies des fonctionnalités qui résultent de la valeur ajoutée par le système multi-agents lui-même, parfois appelée intelligence collective. Ceci constitue le principe fonctionnel.

Fonction(SMA) = S Fonction(Agents) + Fonction collective

Enfin, pour capturer l'esprit de ce que doit être la programmation orientée multi-agents, les systèmes multi-agents peuvent être considérés à un niveau d'abstraction supérieur comme des entités multi-agents. Ce dernier principe est appelé le principe de récursivité.

SMA* = SMA

D'après le paradigme Voyelles, pour un problème à résoudre (ou un système à simuler) dans un domaine donné, l'utilisateur choisit le modèle d'agent, le modèle d'environnement, le modèle d'interaction et le modèle d'organisation qu'il instancie ensuite, ainsi que leurs dynamiques selon

les trois principes Voyelles, de manière à engendrer le système multi-agents computationnel et déployable qui résoudra le problème dans le domaine considéré.

Nous utilisons la décomposition AEIO comme décomposition de base des systèmes multiagents. Selon ce paradigme, initialement proposé par Yves Demazeau dans [Dem95], le système multi-agents est décomposé en quatre composantes ou briques qui sont Agents, Environnements, Interactions et Organisations. Les modalités de la décomposition AEIO sont décrites dans la partie 1.1.1. La décomposition AEIO permet d'obtenir une modularité au niveau des modèles multi-agents, plutôt qu'au niveau des agents et des compétences d'agent. Cette possibilité d'interchanger et de réutiliser les modèles de chaque brique offre un fort potentiel de réutilisation, et surtout une polyvalence inégalée, car le paradigme AEIO ne présuppose pas l'usage d'un modèle particulier à priori. Il n'impose que le découpage entre ces modèles. Une plate-forme utilisant la décomposition AEIO n'est qu'un cadre dans laquelle viennent d'insérer des modèles, orthogonalement aux entités distribuées du système multi-agents.

Chaque méthodologie propose un ensemble de modèles pour représenter les différentes facettes du système. Pour décrire le niveau PIM nous avons choisi quatre modèles qui sont : le modèle du domaine, le modèle d'organisation, le modèle du rôle et le modèle d'agent. Cette décomposition une simple proposition, que nous jugeons bien adaptée pour la modélisation des agents interactifs.

Dans la suite de cette section nous présentons le métamodèle (i.e. formalisme) permettant de décrire chacun des modèles précédemment cités.

6.4.2 Le méta-modèle du Domaine

Le modèle du domaine fournit l'ontologie du domaine et la bibliothèque des comportements prédéfinis qui peuvent être réutilisés pour décrire le comportement des agents. Le méta-modèle du domaine s'inspire de la représentation de l'ontologie dans la méthodologie PASSI [CP02]. Cette ontologie fournit les trois éléments suivants : l'action, le concept et le prédicat. Ces trois éléments composent notre méta-modèle du domaine (voir figure 6.4).

La figure 6.5 décrit le modèle du domaine correspondant à l'application de l'emploi du temps. Ce domaine contient les concepts de l'application emploi du temps : Séance, Créneau et Emploi du temps. L'action Réserver Créneau, prend comme paramètre le créneau et produit une instance de la métaclasse Séance.

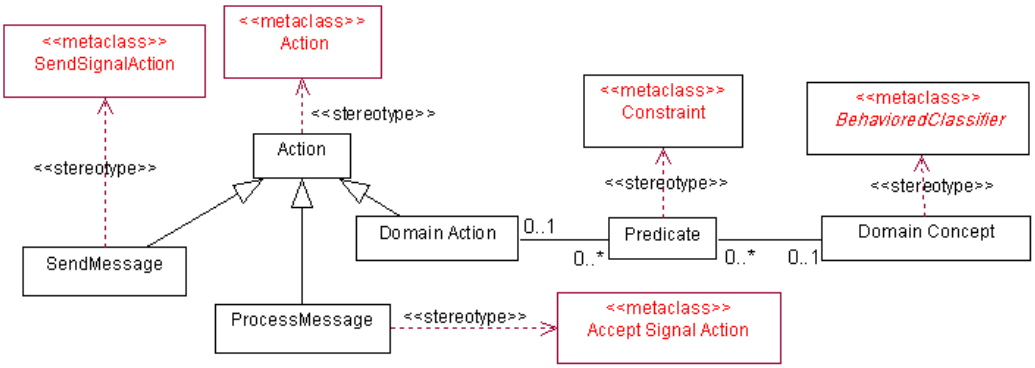


FIG. 6.4 – Le méta-modèle du domaine du PIM

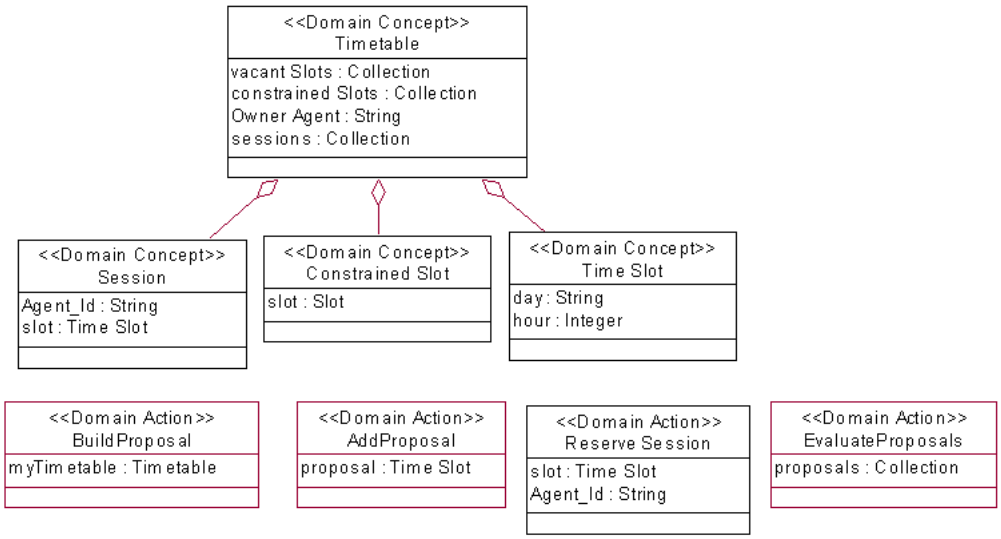


FIG. 6.5 – Un exemple de modèle du domaine

6.4.3 Le méta-modèle de l'organisation

Un système multi-agents est un ensemble d'agents organisés qui interagissent dans un environnement commun. Il y a deux principaux aspects dans la conception des systèmes multi-agents :

- L'aspect micro (orienté agent) s'intéresse à la manière dont nous concevons et construisons un agent autonome dans ses actions,
- L'aspect macro (orienté organisation) s'occupe de la façon dont nous obtenons une société des agents coopérant efficacement.

Pour concevoir un système multi-agents, on doit décrire l'organisation des agents. Sans l'aspect organisationnel, le concepteur du système multi-agents ne peut tirer profit des structures sociales telles que l'émergence et le scalabilité [OLHN04]. Toute métamodélisation multi-agents doit adresser les aspects organisationnels, au moins les concepts : agent, groupe et role.

L'organisation que nous proposons est décrite par un ensemble de rôles avec leurs interactions. Elle peut donc être décrite par un graphe où chaque noeud correspond à un rôle et un trait décrit les interactions entre les rôles associés (voir la figure 6.6). Ce modèle d'organisation est inspiré de la méthodologie Aalaadin [FG98].

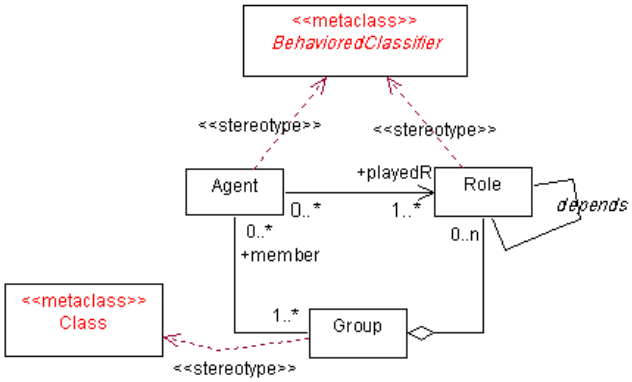


FIG. 6.6 – Le méta-modèle organisationnel

Notre modèle d'organisation qui est représenté par un diagramme de classe. La figure 6.7 décrit l'organisation de l'application emploi du temps, conforme au méta-modèle organisationnel. Elle est composée d'un seul groupe *TimetableManaging* contenant les deux rôles *SessionSeeker* qui essaye définir une séance de cours avec le rôle *SessionProvider*. Chaque rôle est joué par un agent représenté par une association. Dans notre exemple l'agent *Professor* joue le rôle *SessionSeeker* et l'agent *StudentGroup* joue le rôle de *SessionProvider*.

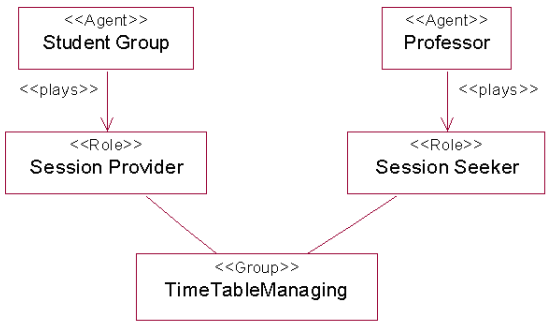


FIG. 6.7 – Un exemple de modèle organisationnel

Le concept *Group* est défini comme un stéréotype de la métaclasse *Class*. De même les concepts *Agent* et *Role* sont des stéréotype de la même métaclasse UML, dont le choix est expliqué dans les sections qui suivent.

6.4.4 Le méta-modèle du rôle

Le méta-modèle du rôle décrit la structure et le comportement de chaque rôle du système. Nous retrouvons le concept de rôle dans la majorité des méthodologies multi-agents Aalaadin, PASSI, GAIA, ROADMAP, etc. Certains travaux, tels que [PV02], ont évoqué l'importance de ce concept pour une meilleure conception, et pour une gestion dynamique du système multi-agents.

Dans le méta-modèle [CBP05] proposé par le groupe de travail AOSE de AgentLink, qui est une synthèse de plusieurs méta-modèles et de méthodologies, le rôle est définit comme une abstraction d'une partie du comportement social de l'agent. Dans ce méta-modèle, le role assure un ensemble de tâches et il est joué par un ou plusieurs agents. Notre métamodèle 6.8 ajoute à cette représentation le concept *Goal*, ainsi le rôle peut avoir un but qu'il essaie de réaliser. L'adoption d'un rôle par un agent donné, implique son engagement à satisfaire ce but. Dans ce méta-modèle le rôle possède un seul *Goal*. Un rôle réalise un ensemble d'actions qui définissent son comportement.

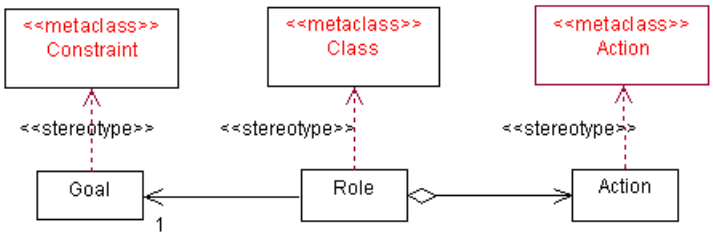


FIG. 6.8 – Le méta-modèle du rôle

Pour représenter le concept de *Role* avec UML 2, nous avons proposé le stéréotype *«Role»* de la métaclasse *Class*. UML 2 a ajouté la métaclasse *BehavioredClassifier* comme une spécialisation de la métaclasse *Classifier*. La spécificité de *BehavioredClassifier* est qu'il possède son propre *Behavior* invoqué au moment de son instanciation. La métaclasse *Class* hérite de *BehavioredClassifier* (voir figure 6.9). Elle a l'attribut booléen *isactive* qui indique si la classe active ou non. Chaque objet d'une classe active possédera son propre son thread de contrôle, et vice versa. La navigation entre la classe et son comportement peut se faire dans les deux sens, c'est à dire le *BehavioredClassifier* peut accéder à son Behavior par l'attribut *itsBehavior*, et le *Behavior* possède l'attribut *context* qui référence son *BehavioredClassifier* et lui permet d'accéder à ses attributs et ses opérations (voir figure 6.9).

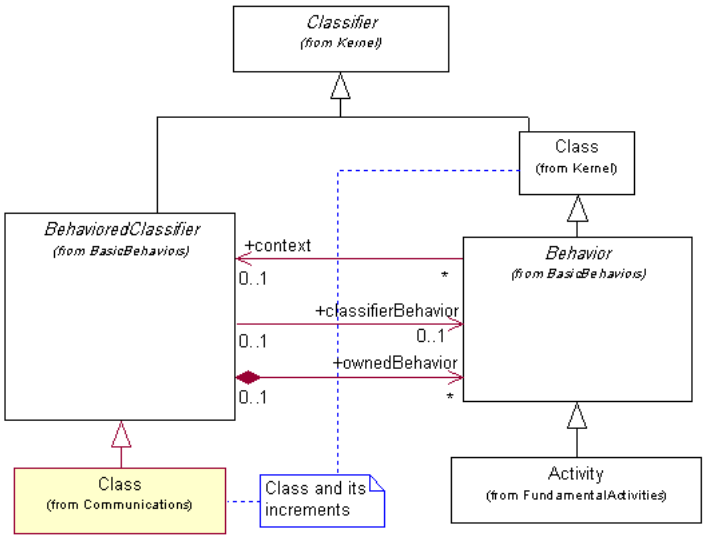


FIG. 6.9 – La métaclasse *Class* de UML 2

Cela justifie le choix de la métaclasse *Class* pour la représentation du rôle. Ainsi la métaclasse *Class* permettra de décrire les connaissances propres au rôle et la métaclasse *Activity*, qui est une spécialisation de la métaclasse *Behavior* permettra de représenter le comportement du rôle, qui est un enchaînement d'actions.

Concernant le concept *Goal*, nous avons défini un stéréotype de la métaclasse *Constraint*. Une contrainte concerne une *NamedElement*, y compris la métaclasse *Class*. Un contrainte en UML 2 est décrite dans un note liée à un *NamedElement*. Le *Goal* est défini comme une expression booléenne que le rôle peut évaluer au cours de son activité et qui est écrite avec le langage OCL.

Au niveau modèle, le développeur utilisera les diagrammes de classes pour décrire la structure du rôle et les diagrammes d'activité pour décrire son comportement. Les informations que possède le rôle *SessionSeeker* sont représentées sous forme d'attributs, voir figure 6.10. Le *Goal* du *SessionSeeker* est de définir une séance en basant sur la description de l'action *ReserverSeance* de l'attribut du rôle *askingForSession*. Par ailleurs, l'attribut *acceptedProposal* contiendra la description de l'action *ReserverSeance* acceptée par le *SessionSeeker*. Alors, nous pouvons dire que si l'attribut *acceptedProposal* est non null alors le *Goal* du rôle est atteint, et vice versa. L'expression du *Goal* a la syntaxe des expressions OCL dont l'attribut *context* indique la classe à la quelle le *Goal* est associé.

Le rôle possède un attribut *classifierBehavior*, hérité de la métaclasse *BehvioredClassifier*, qui référence l'élément de modèle *Activity* qui décrit son comportement. Le comportement du rôle sont décrit par le diagramme d'activité de la figure 6.11. Le rôle *SessionSeeker* commence par envoyer un message de type *CFP* à l'ensemble des *SessionProvider*. Un message *CFP* contient la

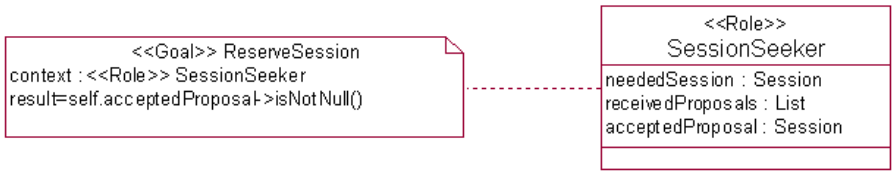


FIG. 6.10 – La classe qui décrit le rôle *SessionSeeker*

description de l'action à exécuter par les receveurs du message. Dans notre cas, l'action est *ReserverSeance* qui permet de reserver un *TimeSlot* du *TimeTable*. Le *SessionSeeker* peut joindre à cette action un ensemble de conditions, par exemple des contraintes sur certains *TimeSlot*. A la réception d'un message *Propose*, le rôle enregistre la proposition, par contre s'il reçoit un *Refuse* il retire le *SessionProvider* de la liste des participants. Le message *Propose* contient une description de l'action *ReserverSeance* qui contient le *timeslot* que le *SessionProvider* propose au *SessionSeeker*. A l'expiration du *timeout* le *SessionSeeker* évalue les propositions reçues pour en choisir une proposition. En cas d'échec, aucune proposition n'a était acceptée, le rôle termine son activité par l'envoi du message *RejectProposal* à l'ensemble des participants. En cas de succès, le participant retenu est informé par un message *AcceptProposal* et les autres participants reçoivent un message *RejectProposal*. Suite à la réception du message *InformDone* confirmant la réservation de la séance, le rôle *SessionSeeker* exécute l'action *ReserverSeance* pour fixer la séance avec le rôle *SessionProvider*, terminant ainsi son activité.

Le diagramme d'activité de la figure 6.11, présente les différentes actions du rôle *SessionSeeker*. Les actions d'envoi de message sont des instances de la métaclasse *SendMessage* qui est un stéréotype de la métaclasse *SendSignalAction*. Le concept *SendMessage* est une spécialisation du concept *Action*,voir le métamodèle du domaine (figure 6.4). Une action *SendMessage* possède l'attribut *message* qui indique le type du message à envoyer, une instance du concept *Message*.

Les actions de réception de message sont des instances de la métaclasse *ReceiveMessage* qui est un stéréotype de la métaclasse *AcceptEventAction*. Le concept *ReceiveMessage* est une spécialisation du concept *Action*, voir figure 6.4. Une action *ReceiveMessage* possède l'attribut *message* qui indique le type du message reçu, une instance du concept *Message*.

Les actions *SendSignalAction* et *AcceptEventAction* ont été utilisées dans les travaux [Wag02] [KWT04], pour modéliser les actions d'envoi et de réception de messages. Le profil UML de la méthodologie *AOR* [Wag02] stéréotype ces métaclasses.

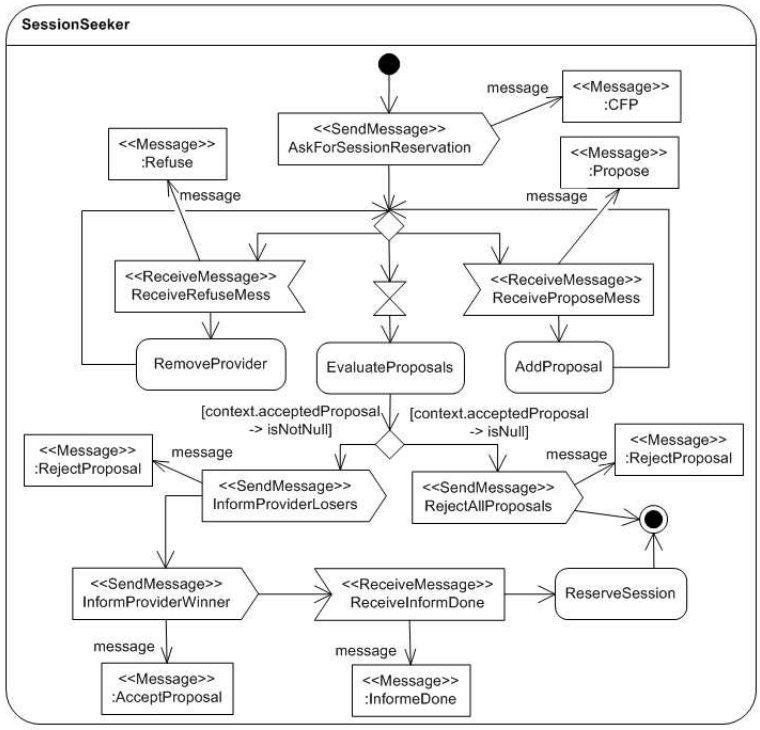


FIG. 6.11 – Le diagramme d'activité du rôle *SessionSeeker*

6.4.5 Le méta-modèle de l'Agent

Un agent est une entité active et autonome, caractérisé principalement par ses rôles et son comportement. Le comportement définit l'action/reaction de l'agent à l'évolution à son environnement selon son but. Ce comportement se fond sur les rôles d'agent, son but et son environnement. Plusieurs méta-modèles peuvent être donc définis pour décrire des agents et leurs rôles. Ces modèles se basent sur le type de connaissance des agents. La connaissance peut être superficielle (pour un agent réactif) ou profonde (pour un agent cognitif/deliberatif).

Ce méta-modèle présente une représentation de base de l'agent qui ressemble à la proposition du métamodèle [CBP05]. Le concept *Agent* possède un *Goal* et exécute un ensemble d'actions. Les concepts *Action* et *Goal* ont été décrits dans la section précédente. Nous rappelons que *Agent* peut jouer un ensemble de rôles, voir le métamodèle de l'organisation.

Le concept d'agent est généralement représenté par une classe stéréotypée [KWT04] [OLHN04]. Les connaissances de l'agent sont décrites par les attributs de la classe. La classe de la figure 6.13 représente les connaissances de l'agent *Professor*. Le but du professeur est de définir les séances de son emploi du temps. Puisque l'attribut *neededSession* contient l'ensemble des séances à

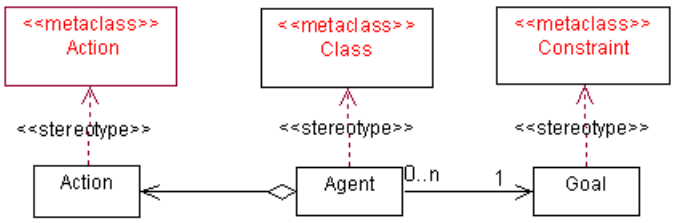


FIG. 6.12 – Le méta-modèle de l'agent

fixer, alors nous pouvons exprimer le *Goal* par un expression booléenne : la liste *neededSessions* est vide.

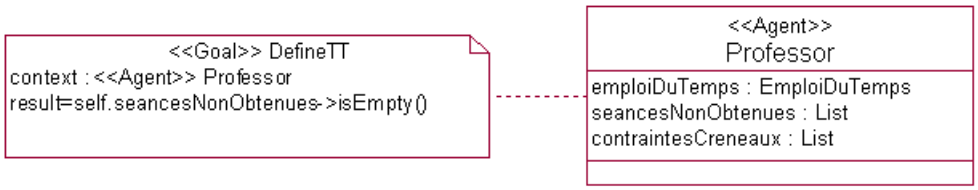


FIG. 6.13 – La classe qui représente l'agent *Professor*

Les diagrammes d'activité d'UML ont été utilisés pour décrire le comportement des agents [KWT04]. Le comportement de l'agent *Professor* est décrit par le diagramme d'activité de la figure 6.14. L'agent commence par évaluer son *Goal*. Si son goal n'est pas encore atteint alors l'agent demande au gestionnaire du groupe *TimeTableManaging* de jouer le rôle *SessionSeeker* par le biais de l'action *RequestRole*. En cas d'acceptation de la demande du rôle, l'agent sélectionne la séance à demander. Ensuite il active le rôle *SessionSeeker*. A la fin du rôle, l'agent évalue de nouveau son *Goal*.

6.5 Les méta-modèles spécifiques à la plate-forme : le cas du framework INAF

INAF possède deux principaux concepts : l'agent et le rôle d'interaction, voir figure 6.15. L'agent peut offrir des services et possède un ensemble de stratégies qu'il utilise au cours de ses interactions. Dans INAF, le comportment de l'agent peut est implémenté par un ATN ou une base de règles. De même pour le rôle d'interaction qui est décrit par un ATN. Tous les concepts de ce méta-modèle, sont représentés comme des stéréotypes de la métaclasse *Class*. Cela nous paraît tout à logique pour un framework utilisant un langage orienté objet.

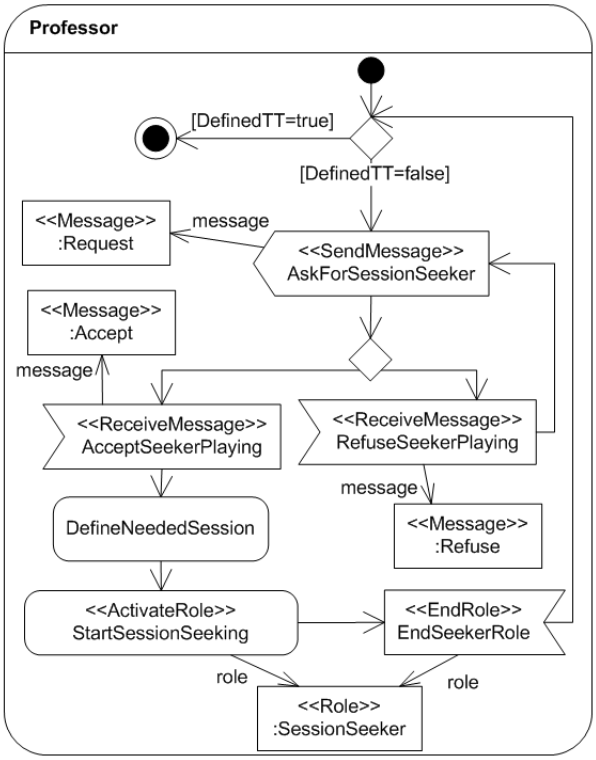


FIG. 6.14 – Le diagramme d'activité de l'agent Professor

6.6 Transformations

Nous nous sommes intéressés à la représentation des diverses connaissances impliquées dans le développement multi-agents. Une partie de ces connaissances apparaît sous forme de modèles et de méta-modèles, d'autres parties sont sous la forme de règles de transformation (par exemple celles entre le PIM et le PSM). La transformation des modèles constitue le coeur de l'approche MDA [ACR+03].

Les règles de transformation du PIM vers un PSM représentent le savoir-faire du développeur. De telles règles de transformation sont considérées comme implicites et employées manuellement par le programmeur. Pour définir les règles de transformation, nous avons d'abord identifié les diverses relations entre les deux niveaux : PIM et PSM. Nous avons alors défini les règles de transformation pour les méta-modèles proposés.

Certains travaux ont proposé des transformations de modèles, par exemple [AFV04]. Cependant, les règles de transformation sont souvent réduites à des problèmes de transformation de diagrammes UML. Nous préconisons que les transformations dans le cadre des SMA, représentent tous le savoir-faire des chercheurs et des développeurs de ce domaine.

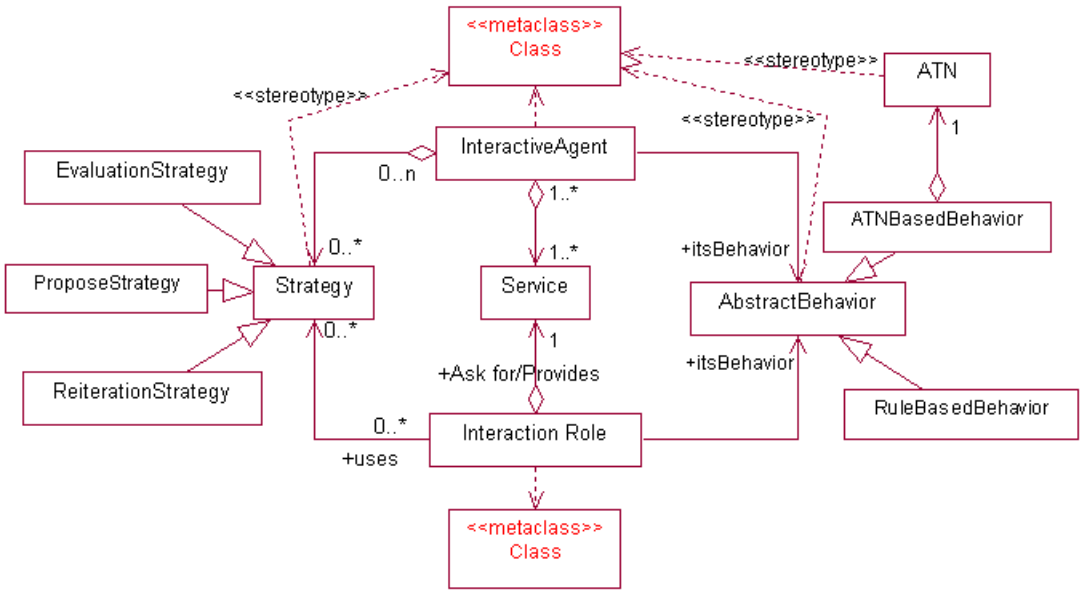


FIG. 6.15 – Le méta-modèle d'INAF

Dans cette section nous allons analyser les différents modèles et identifier les règles de transformation, ainsi que les difficultés du processus de transformation du PIM en PSM. Pour cela, nous avons identifié deux relations entre les concepts de ces deux types de modèles : *mappings* et *use*.

6.6.1 Relation entre les méta-modèles

MDAD est basée sur deux niveaux d'abstraction : la description du SMA et de son implémentation. Chaque niveau d'abstraction peut avoir plusieurs facettes, chacune d'elles correspond à un méta-modèle. Les relations entre les méta-modèles sont les suivantes :

- «Mapping» : un modèle décrit avec le méta-modèle cible peut être obtenu par transformation d'un modèle décrit dans les méta-modèles de source. Par exemple, un modèle de rôle peut être transformé en modèle de comportement d'agent,
- «Use» : le modèle source a besoin du modèle cible. Par exemple, le code généré sous DIMA ne compilera pas si l'implémentation du domaine est absente.

En se basant sur les méta-modèles spécifier plus haut, nous allons définir dans ce qui suit les relations *Mapping* et *Use* qui existent entre les concepts PIM et PSM.

6.6.2 Les relations de type Mapping

Un modèle décrit avec le méta-modèle cible (par exemple le PSM) peut être obtenu par transformation d'un modèle décrit avec un ou plusieurs méta-modèles source (par exemple le PIM). Par exemple, un modèle de rôle peut être transformé en modèle de comportement d'agent, Nous présentons dans le tableau 6.1 les concepts du PIM qui ont une correspondance directe avec les concepts du PSM.

TAB. 6.1 – Les règles de mapping entre les concepts du PIM et du PSM

Concept PIM	Concept PSM
Agent	Interactive Agent
Action	Action
Goal	Operation
Activity	ATN
Role	rien
Group	rien
rien	Service
rien	Strategy
rien	Interaction Role
rien	ATNBasedBehavior

Ce mapping est basé sur les différents concepts PIM et PSM. Par exemple pour réaliser le mapping entre ***Agent*** et *InteractiveAgent*, nous utilisons : (i) la description de la classe stéréotypée «Agent» associée à l'Agent dans modèle d'agent, et plus précisément dans le diagramme de classe de l'agent, et (ii) le diagramme d'activité associé à la classe de l'agent qui décrit son comportement. Le diagramme de classe permet de construire une nouvelle instance du concept InteractiveAgent dans le PSM avec le même nom et les mêmes propriétés. Le diagramme d'activité de l'agent se transforme en une instance d'ATN. Ce mapping entre le diagramme d'activité et l'ATN sera détaillé dans la section 6.6.3.1.

De même le concept ***Action*** possède une correspondance directe dans INAF celle de la métaclasse *Action*. La figure 6.16 décrit le métamodèle des actions du niveau PIM. Les sous-types du concept Action permettent de manipuler les concepts *Role* et *Message*. La figure 6.17 décrit le méta-modèle des actions et des conditions dans INAF. Les actions du PIM sont transformées en un ensemble d'actions dans INAF (voir tableau 6.2).

Par contre, la métaclasse *RequestRole* ne possède pas de correspondance dans INAF, car elle fait partie des actions qui gèrent l'activité organisationnelle, un aspect non représenté dans

INAF. Ainsi, nous considérons que l'agent peut jouer n'importe rôle. L'utilisation du *RequestRole* est toujours suivi par la réception d'un *Accept* ou d'un *Refuse*. Au moment de la transformation tous les éléments (ActivityEdge, AcceptEventAction, Action) qui sont liés à l'action *RequestRole* ne sont pas transformés.

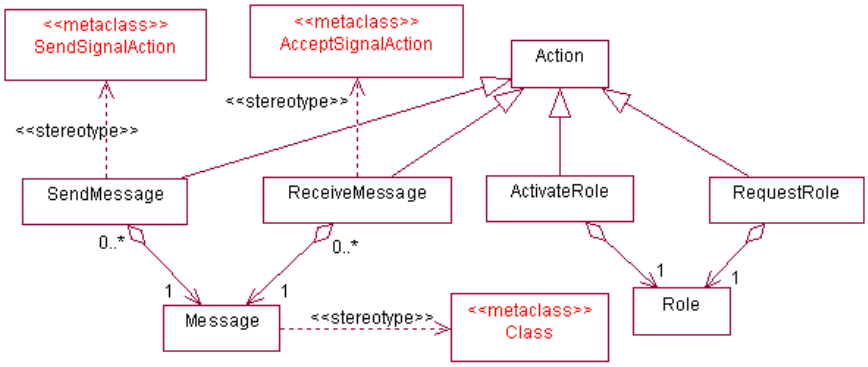


FIG. 6.16 – Le méta-modèle des actions du PIM

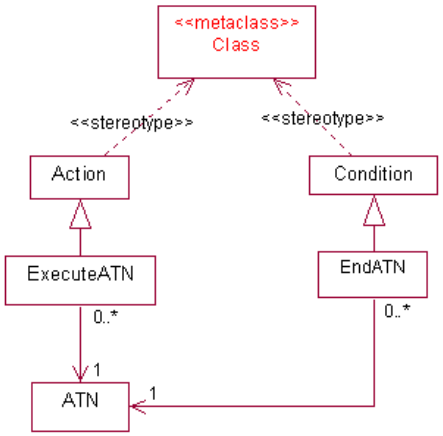


FIG. 6.17 – Le méta-modèle des actions et des conditions dans INAF

TAB. 6.2 – Les règles de mapping entre les actions du PIM et ceux du PSM

Concept PIM	Concept PSM	Commentaires
SendMessage(M)	Action	M est le message à envoyer
ReceiveMessage(M)	Condition	M est le message reçu
TimeEventAction(T)	Condition	T est le temps d'attente
ActivateRole(R)	ExecuteATN(A)	A est l'ATN obtenu par transformation du comportement du rôle R
EndRole(R)	EndATN(A)	A est l'ATN obtenu par transformation du comportement du rôle R
RequestRole(R)	rien	R est le rôle demandé par l'agent

La deuxième étape consiste à analyser les autres concepts qui ne peuvent pas être liés avec ceux du PSM.

6.6.3 Relations de type USE

La relation mapping correspond à la transformation directe d'un concept PIM en un concept PSM. Cependant, d'autres concepts PIM ne sont pas directement transformable dans le PSM, par exemple le concept de Role. Pour ces concepts nous avons défini le type de relation USE. Une relation de type USE établit un lien de transformation entre un concept PIM et un ou plusieurs concepts PSM. Nous présentons dans cette section quelques concepts PIM et les relations USE qui leurs sont associées.

Dans le framework INAF, nous n'avons pas le concept de ***Role***. La modélisation du rôle dans le PSM dépend de sa représentation dans le PIM. Il y a différentes possibilités :

- Si au niveau du PIM, le rôle ne possède pas un comportement prédéfinie => alors le Role sera décrit dans le PSM par une instance de la métaclasse AbstractBehavior au nom du rôle, avec les mêmes propriétés.
- Si le comportement du Role est prédéfini et décrit par un diagramme d'activité associé à la classe stéréotypée «Role», nous le représenterons dans INAF par un ATNBasedBehavior contenant un ATN qui implémente le comportement du rôle.

Le concept ***Group*** n'a pas de correspondance directe dans INAF. L'implémentation de ce concept et de sa gestion sous INAF s'avère compliquée. Alors, nous avons choisie de ne pas reprendre ce concept dans INAF.

Dans le PIM, le ***Goal*** est un stéréotype de la métaclasse Constraint qui peut être décrit par une expression OCL. Par contre dans INAF le but des agents et des rôles n'est pas réifié. Dans INAF, le but de l'agent interactif est décrit dans la méthode *isActive*. Nous pouvons l'associer à la métaclasse *Condition*, une classe stéréotypée, qui implémente l'expression OCL du *Goal*.

6.6.3.1 Transformer un *Activity* en un ATN

Le mapping entre un *Activity* et un ATN est composé d'un ensemble de règles de mapping entre ces deux concepts,voir le tableau 6.3. Pour une bonne compréhension de ce mapping nous avons repris dans la figure 6.18 les principaux concepts UML utilisés pour décrire un diagramme d'activité. Ce dernier correspond à l'élément de modélisation *Activity* du méta-modèle d'UML.

Le métamodèle de l'ATN est décrit dans la figure 6.19. La structure d'un ATN ressemble à celle d'un AG, sauf que dans l'ATN nous n'avons pas la notion d'événement. De plus dans

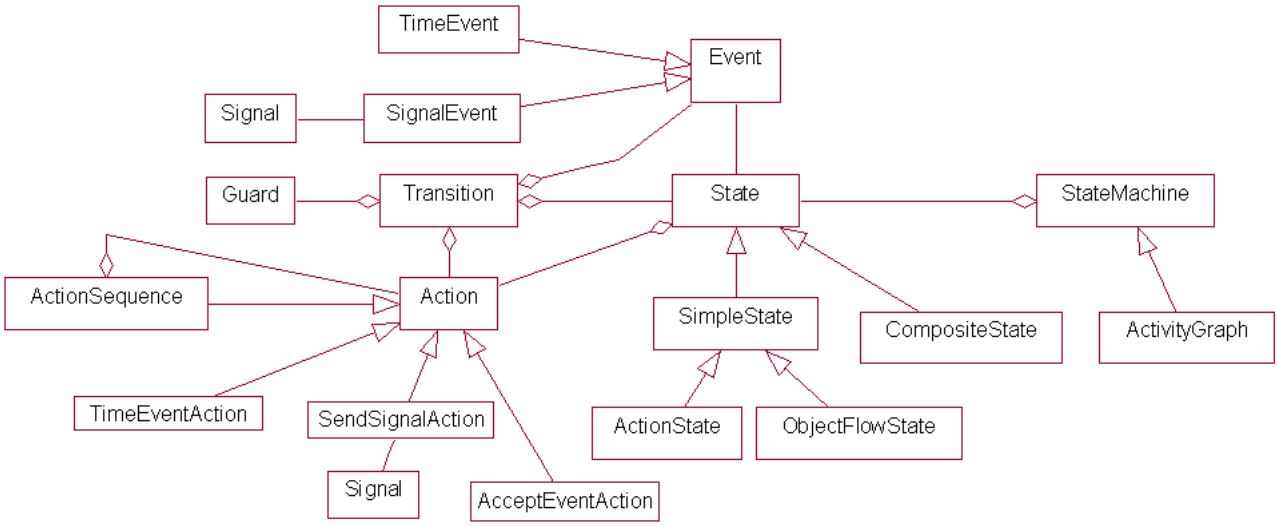


FIG. 6.18 – Le méta-modèle de Activity

l'ATN, les actions sont rattachées aux transitions, par contre dans AG les actions sont spécifié dans les états.

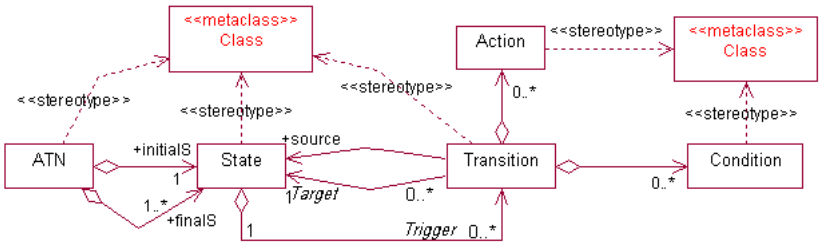


FIG. 6.19 – Le méta-modèle de l'ATN

Le tableau 6.3 montre les relations de correspondance entre les concepts de *Activity* et ceux de l'*ATN*. Dans la suite de cette section nous présenterons les différentes règles de transformation permettant d'obtenir un ATN à partir d'un *Activity*.

6.6.4 Spécification des règles de transformation

Pour exprimer les règles de transformation, l'OMG a soumis en 2002 un appel à proposition [Gro02]. Le but est de définir un langage pour spécifier les règles de transformation. QVT est l'abréviation de (*Query View Transformation*). *Query* est une requête qui prend en entrée un modèle et sélectionne des éléments spécifiques de ce modèle. *View* est un modèle qui dérive d'autres modèles. *Transformation* prend un modèle en entrée pour le modifier ou en créer un

TAB. 6.3 – Les règles de mapping entre les concepts de *Activity* et de l'ATN

Concept Activity	Concept ATN
SendSignalAction	Action
ReceiveMessage	Action
TimeEventAction	Condition
StateNode	State
ActivityEdge	Transition
Guard	Condition
DecisionNode	State
ForkNode	rien
MergeNode	rien

autre. *QVT* [Gro05] propose un langage pour l'expression des relations entre métamodèles. Nous utiliserons ce langage pour l'écriture de quelques règles de transformation. Une relation a la syntaxe suivante :

```
relation M1_To_M2 { checkonly domain d1 el1 :M1{pattern1} ;
                    enforce domain d2 el2 :M2{pattern2} ;
                    When{expression1}
                    Where{expression2}
                  }
```

La variable *d1* référence le modèle source qui contient l'élément de modèle *el1*, instance de la métaclasse *M1*. Le modèle source est précédé par le terme *enforce domain*. La variable *d2* référence le modèle cible qui contient l'élément de modèle *el2*, instance de la métaclasse *M2*. Le modèle cible est précédé par le terme *enforce domain*. Les patterns sont appliqués sur l'élément du modèle, pour définir des contraintes sur les éléments concernés par la règle. Le champ *When* définit les préconditions de la relation. Le champ *Where* décrit les postconditions de la relation.

Pour faciliter les transformations de nos modèles, nous prenons comme hypothèse que le modèle du domaine est le même à tout les niveaux de modélisation. Donc, nous commençons par écrire les règles de transformation du Role.

6.6.4.1 les règles de transformation du rôle

Dans cette section nous donnons l'ensemble des règles de transformation nécessaires pour obtenir un PSM compatible au framework INAF.

Règle 1 : Pour un *Role R* dont l'activité est décrite par un *Activity G* alors :

1. On invoque la règle 6 pour transformer *G* en un *ATN T*.
2. Ensuite, on crée un *ATNBasedBehavior TB* à partir du nom et des attributs de *R*.
3. Enfin nous associons *T* à l'attribut *atn* de *TB*.

```
relation RoleToATNBasedBehavior
{ nm : String ;
  checkonly domain sma r :Role {itsBehavior = a :Activity, name=nm} ;
  enforce domain inaf b :ATNBasedBehavior {name=nm, atn=t :ATN{}} ;
  when{}
  where{ActivityToATN(a,t) ;}
}
```

Règle 2 : Pour un *Role R* dont l'activité n'est pas définie alors on crée un *AbstractBehavior AB* à partir du nom et des attributs de *R*.

```
relation RoleToAbstractBehavior
{ nm : String ;
  checkonly domain sma r :Role {itsBehavior=null, name=nm} ;
  enforce domain inaf b :AbstractBehavior {name=nm} ;
  when{}
  where{}
}
```

6.6.4.2 La règle de transformation de l'Agent

Règle 3 : Pour un *Agent A*, si l'ensemble de ses rôles ont été transformés (i.e. on retrouve les rôles de l'agent à partir du diagramme d'organisation), et si l'agent possède un comportement décrit par un *ActivityGraph G*, alors :

1. On crée un *InteractiveAgent IA* avec le même nom et les attributs de *A*.
2. Ensuite, on invoque la règle 6 pour transformer *G* en un *ATN T*.

3. Enfin, on crée un objet t de T qu'on associe à l'attribut *behavior* de *IA*.

```
relation AgentToInteractiveAgent
{ nm : String ;
  checkonly domain sma a :Agent {itsBehavior=g :Activity, name=nm, playedRole=r :Role} ;
  enforce domain inaf ia :InteractiveAgent { name=nm, atn=t :ATN,
                                             hasBehavior=b :ATNBasedBehavior} ;
  when{RoleToATNBasedBehavior(r,b) ;}
  where{ActivityToATN(g,t) ;}
}
```

6.6.4.3 Les règles de transformation du Goal

Règle 4 : Pour un *Goal* G, avec G est associé à un *Role* R et que R est déjà transformé en un élément C (C peut être une *ATNBasedBehavior* ou un *AbstractBehavior*)alors :

1. On ajoute à C une *Operation* booléenne O qui le même que G. Le corps de cette opération est l'expression OCL de G.

Règle 5 : Pour un *Goal* G, avec G est associé à un *Agent* A et que A est déjà transformé en un *InteractiveAgent* *IA* alors :

1. On invoque la règle 5 pour G,
2. On ajoute à *IA* une *Operation* booléenne nommée *"isActive"* qui invoque l'opération G.

6.6.4.4 Les règles de transformation de l'activité

Règle 6 : Pour un *Activity* G alors :

1. On crée un *ATN* T avec le même nom que G.
2. Ensuite un interpréteur est utilisé pour parcourir G. A chaque rencontre d'un élément du diagramme alors on active la règle correspondante. L'ensemble des règles utilisées pour la transformation de G sont 7, 8, 9, 10, 11 et 12.

Règle 7 : Pour chaque instance de la métaclasse *Action* alors :

1. Créer une instance de la métaclasse *Transition* ActionTrans.
2. Créer une instance de la métaclasse *Action*.
3. Créer deux instances de la métaclasse *State* preAction et postAction.
4. Associer au state preAction la transition ActionTrans en tant que transition de sortie.
5. Associer au state preAction la transition d'entrée du *Action* en tant que transition d'entrée.

6. Associer au state postAction la transition ActionTrans en tant que transition d'entrée.
7. Associer au state postAction la transition de sortie du *Action* en tant que transition de sortie.

Règle 8 : Pour chaque instance de *SendMessage S*, liée a une instance *M* de *Message* alors :

1. Créer une instance *A* de *Action* qui reprend le nom de *S*.
2. Créer une instance *M1* de *FIPAMessage* qui reprend le nom de *M*.
3. Ajouter une association entre *M1* et *S*.

Règle 9 : Pour chaque instance de *ReceiveMessage R*, liée a une instance *M* de *Message* alors :

1. Créer une instance *C* de *Condition*.
2. Créer une instance *M1* de *FIPAMessage* qui reprend le nom de *M*.
3. Ajouter une association entre *M1* et *C*.

Règle 10 : Pour chaque instance de la métaclasse *TimeEventAction* alors :

1. Créer une instance de la métaclasse *Condition*.
2. Créer une instance de la métaclasse *Timeout* dont la valeur est celle de l'attribut time du *TimeEventAction*.
3. Associer ce *Timeout* à la *Condition* créé.

Règle 11 : Pour chaque *Decision Node* alors :

1. Créer une instance de la métaclasse *State*.
2. Créer des transitions au nombre des transitions sortantes du *Decision Node*.
3. Associer ces transitions à l'attribut *transitions* du *State* créé.
4. Chaque *Guard* de chaque *Transition* se transforme en *Condition*.
5. La *Transition* entrante du *Decision Node* se transforme en une *Transition* dont l'attribut *state* est associé au nouveau *State* créé.

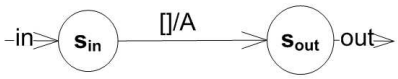
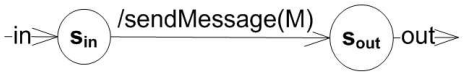
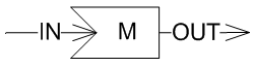
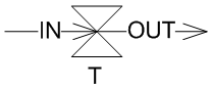
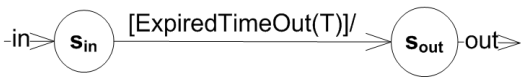
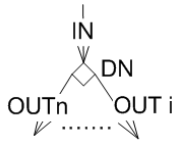
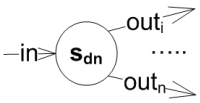
Règle 12 : Pour chaque couple (*Fork Node* et *Join Node*) :

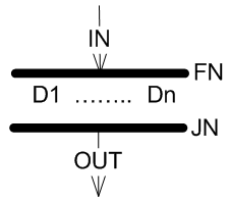
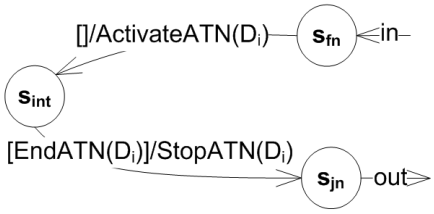
1. Créer des instances de la métaclasse *ATN* au nombre des transitions sortantes du Fork Node.
2. Ajouter à chaque *ATN* une instance de State défini comme état initial et une deuxième instance définie comme un état final.
3. Parcourir chaque branche et la transformer en un ATN : la première transition sera liée à l'état initial et la dernière transition sera associé à état final, précédemment créé.
4. Remplacer le Fork Node par un State FN qui prend comme transition entrante celle du Fork Node.
5. Remplacer le Join Node par un State qui prend comme transition sortante celle du Join Node.

6. Créer un instance du State nommé Inter.
7. Créer un instance de Transition nommée FNtoInter qui a pour état source FN et pour état cible Inter.
8. Créer une instance de la métaclasse ExecuteATN pour chaque ATN créé et les associer à la transition FNtoInter.
9. Créer une instance de Transition nommée IntertoJN qui a pour état source Inter et pour état cible JN.
10. Créer une instance de la métaclasse EndATN pour chaque ATN créé et les associer à la transition IntertoJN.
11. Créer une instance de la métaclasse StopATN pour chaque ATN créé et les associer à la transition IntertoJN.

Règle 13 : Pour un *RequestRole RA* pour un *Role R*, si *R* est transformé en ATNBasedBehavior *AB*, et si l'activité de R est transformée en un *ATN AT* alors :

1. Créer une action *ExecuteATN A*.
2. Associer *AT* à l'attribut *atn* de *AB*.

Concepts de Activity	Concepts de ATN	Règles de transf.
 Action		Règle 7
 SendMessage		Règle 8
 ReceiveMessage		Règle 9
 Time Event Action		Règle 10
 Decision Node		Règle 11

Concepts de Activity	Concepts de ATN	Règles de transf.
 Fork et Join Nodes		Règle 12

6.6.5 Application : Emploi du temps

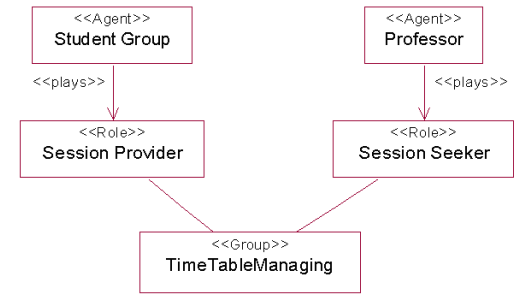
Dans cette section, nous considérons les PIM de l'emploi du temps et nous montrons le déroulement des transformations pour générer le PSM correspondant au framework INAF. Le tableau 6.5 regroupe les différents éléments des modèles de l'organisation, de rôles et d'agents. Nous ne traitons pas dans ce travail la transformation du domaine. Nous supposons que le modèle du domaine existe au niveau de INAF.

Etape 1 :
Au début de la transformation, la seule règle déclenchable est la règle 1 : Pour un *Role R* dont l'activité est décrite par *Activity G* alors :

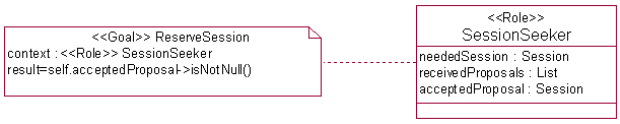
1. On invoque la règle 6 pour transformer *G* en un *ATN T* : La règle 6 parcourt le diagramme d'activité et transforme chaque élément du diagramme, en invoquant la règle correspondante.
2. Ensuite, on crée un *ATNBasedBehavior TB* à partir des attributs et des opérations de *R*.
3. Après, on ajoute à *TB* une méthode *buildATN*.
4. Ensuite, pour chaque action/condition de *T*, on ajoute une opération dans *TB* qui reprend le nom de l'action/condition.
5. Enfin, si le rôle possède un *Goal* alors on invoque la règle 4 : chaque Goal se transforme en une opération booléenne et dont le corps est celui du *Goal*.

La règle 1 est appliquée deux fois pour générer les deux rôles *SessionSeeker* et *SessionProvider*. L'invocation de la règle 6 à partir de cette règle permet de construire deux *ATN T1* et *T2* décrits par des diagrammes de classe correspondants aux *Activity* des deux rôles *SessionSeeker* et *SessionProvider*. La figure 6.20 décrit l'ATN *SessionSeekerActivity* obtenu par transformation du diagramme d'activité du rôle *SessionSeeker*, en utilisant les règles 7, 8, 9, 10 et 11. La figure 6.21 décrit les différents éléments *State*, *Condition* et *Transition*, qui composent l'*ATN* du *ATNBasedBehavior SessionSeeker*.

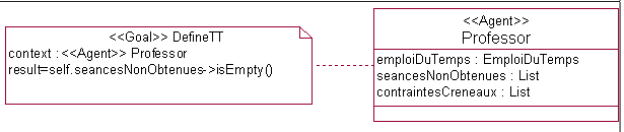
TAB. 6.5 – Les diagrammes PIM de l'emploi du temps



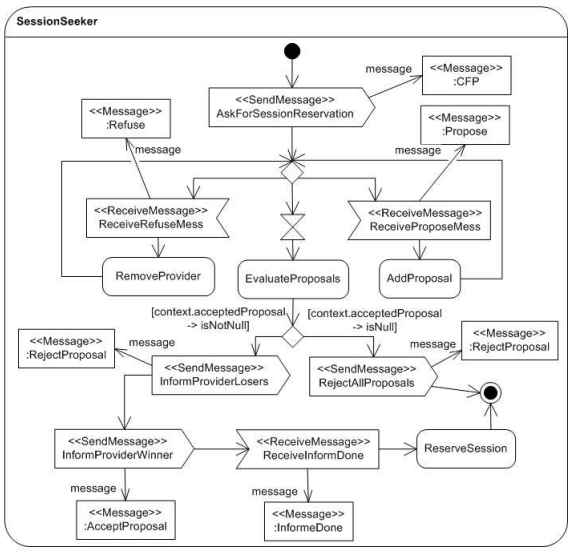
(a) Modèle Organisationnel



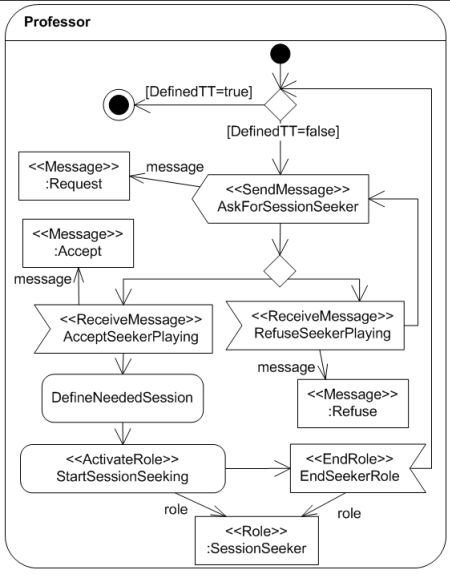
(b) Structure du role SessionSeeker



(c) Structure de l'Agent Professeur



(d) L'activité du role SessionSeeker



(e) Le comportement de l'Agent Professeur

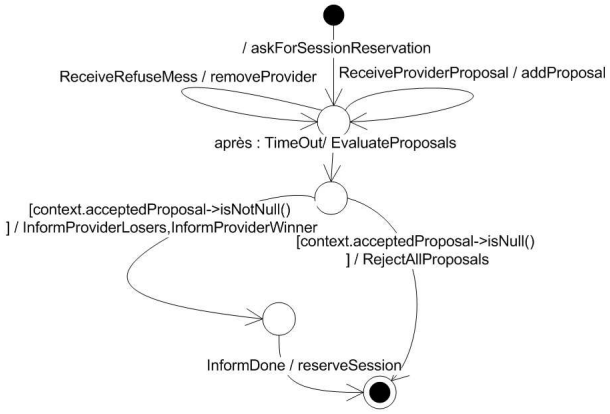


FIG. 6.20 – L'ATN *SeessionSeekerActivity*

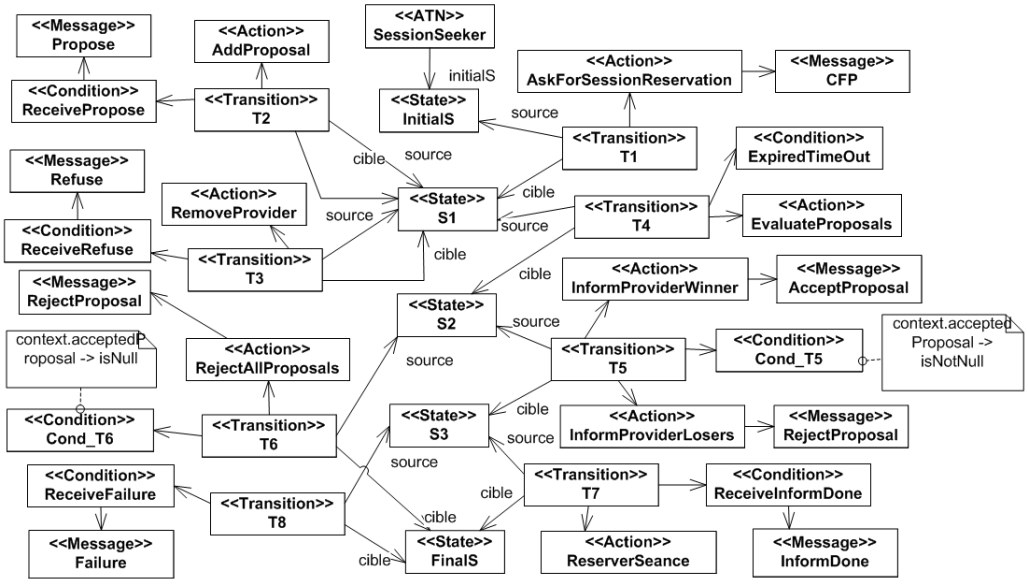


FIG. 6.21 – Le diagramme d'objets de l'ATN *SeessionSeekerActivity*

L'exécution de la règle 1 permet ainsi de construire deux *ATNBasedBehavior SessionSeeker* et *SessionProvider* correspondants aux deux rôles du PIM. Les deux ATN obtenus par application de la règle 6 seront utilisés dans la phase de génération de code (voir section 6.7), pour définir les opérations des classes *SessionSeeker* et *SessionProvider*. La figure 6.22 décrit l'*ATNBasedBehavior* résultat de la transformation du rôle *SessionSeeker*.

Etape 2 :

Au deuxième passage, la règle déclenchable est la règle 3 : Pour un *Agent A*, si l'ensemble de ses rôles ont été transformés (i.e. les liens entre l'agent et ses rôles sont décrits dans le modèle

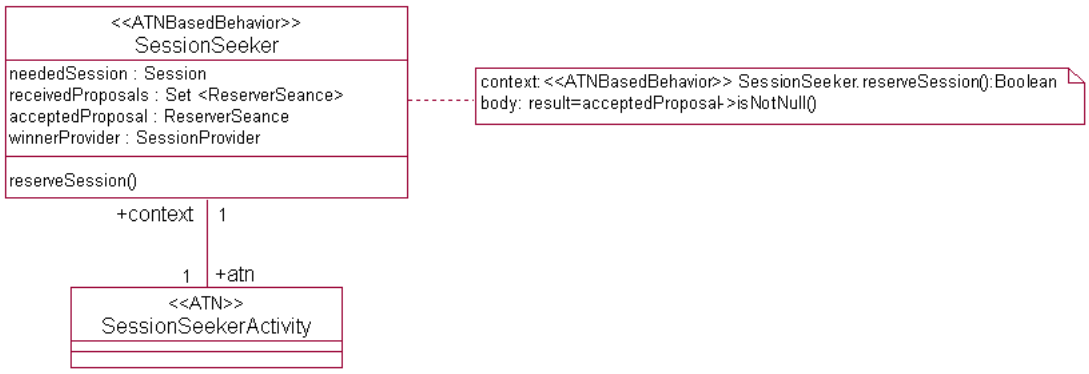


FIG. 6.22 – La structure de l'*ATNBasedBehavior SessionSeeker*

organisationnel), et si A possède un comportement décrit par un diagramme d'activité A_ACT, alors :

1. On commence par invoque la règle 6 pour transformer A_ACT en un ATN T.
2. Ensuite, on crée un *InteractiveAgent* IA avec les attributs et les opérations de A.
3. Après, pour chaque action AC de T, on ajoute une opération à IA qui reprend le nom de AC.
4. Pour chaque condition C de T, on ajoute une opération à IA qui reprend le nom de C.
5. Enfin, si A possède un *Goal* G alors on invoque la règle 5 pour le transformer en une opération booléenne à IA, qui reprend le nom G et dont le corps est celui du G.

Dans cette application, la règle 3 est appliquée deux fois pour générer les deux agents interactifs *Professor* et *StudentGroup*. La classe de la figure 6.23 décrit la structure de l'agent interactif *Professor*. La transformation de son *Goal DefineTT* a générée l'opération booléenne *defineTT()* et dont le corps reprend l'expression OCL de *DefineTT*. Nous utilisons à ce stade de la transformation la notation OCL, car les concepts PSM auxquels sont associées ces expressions, ne sont pas encore transformés en éléments de code Java.

L'invocation de la règle 6 permet d'obtenir les ATN des deux agents. La règle 6 parcourt le diagramme d'activité et invoque les règles 7, 8, 9, 10 et 11, qui transforment les éléments du diagrammes d'activité en éléments de l'ATN (état, action, condition, transition). La figure 6.24 décrit l'ATN représentant le comportement de l'agent *Professor*.

Les actions *SendMessage* du niveau PIM sont transformées en actions au niveau PSM. Les actions *ReceiveMessage* sont transformées en conditions. La condition reprend le nom de l'action *ReceiveMessage*. Le message lié à une action *SendMessage* ou *ReceiveMessage* est repris au niveau PSM.

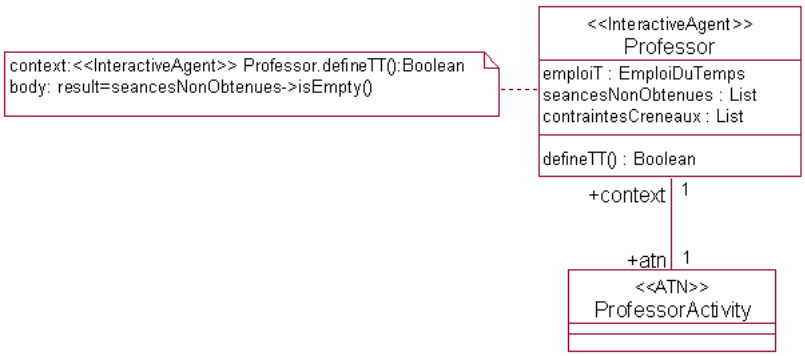


FIG. 6.23 – L'agent interactif *Professor*

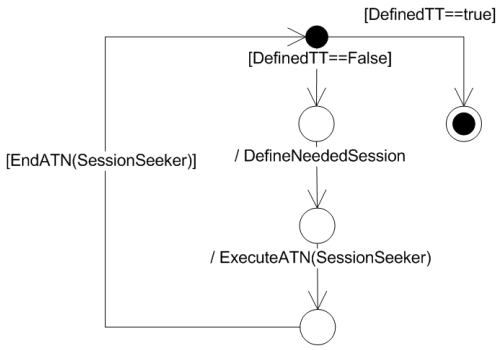


FIG. 6.24 – L'ATN *Professor*

6.6.6 Synthèse

Le savoir-faire nécessaire pour passer d'un modèle PIM basé sur l'architecture organisationnelle AGR à un modèle PSM se basant sur INAF peut se résumer aux points suivants :

1. Puisque INAF n'implémente pas le concept *Role*, alors nous l'implémentons par un *ATNBasedBehavior*, si rôle possède un comportement prédéfini. sinon nous le transformons en un *AbstractBehavior*.
2. L'activité du rôle est implémentée par un *ATN*.
3. Il faut savoir que la structure de groupe et sa gestion, ne sont pas supportées par INAF. Ainsi, toute l'activité liée à la gestion du groupe, y compris l'acquisition des rôles, par l'action *RequestRole*, ne seront pas implémentées dans INAF. Alors nous proposons de ne pas reprendre concept *Group* dans le passage à l'implémentation, et le mécanisme d'acquisition du rôle est remplacé par un ajout et une exécution automatique de l'*ATNBasedBehavior* qui implémente du role.
4. Le concept *Goal* n'a pas de correspondance directe dans INAF. Nous proposons de l'implémenter par une opération booléenne de la classe à la quelle il est associé.

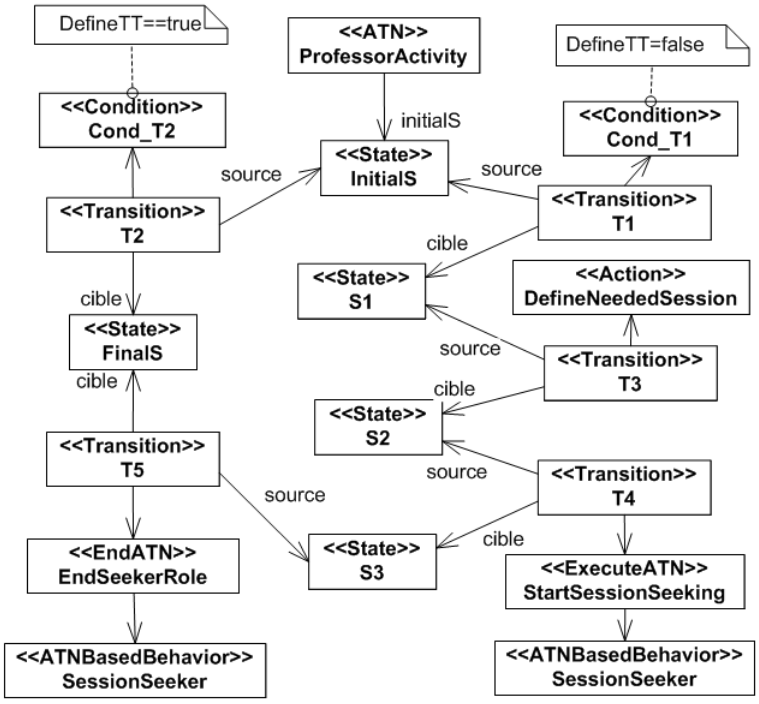


FIG. 6.25 – Le diagramme d'objets de l'ATN *Professor*

5. L'activité de l'agent est implémentée par un ATN qui est le résultat de la transformation du comportement de l'agent, décrit par un diagramme d'activité.
6. La gestion des messages peut être différente d'une plate-forme à une autre. Dans INAF, les messages sont stockés dans une boîte à messages que l'agent consulte de façon régulière.
7. l'interaction dans le niveau PIM se déroule entre les rôles, par contre, dans INAF l'interaction se déroule entre les agents. Ainsi les actions d'envoie et de réception de messages définies dans un *ATNBasedBehavior*, sont assurées par l'agent qui l'a adopté.

Le PSM obtenu par application des règles de transformation peut être raffiné dans le but de l'améliorer et de le rapprocher un plus de l'implémentation opérationnelle du système.

6.7 La génération du code

La troisième étape de notre méthode *MDAD* est la génération du code à partir du modèle PSM. Comme on l'a expliqué précédemment (cf. 6.2.3), la génération du code ne fait pas partie de la phase de transformation de modèles.

Avant d'arriver à l'implémentation finale du système nous devons générer à partir du modèle PSM qui est conforme au méta-modèle de la plate-forme, un modèle de classes qui est une

extension des classes de base de la plate-forme d'implémentation. Nous donnons dans ce qui suit l'ensemble des règles qui permettrons la génération de ce modèle de classes. Ce modèle intermédiaire est important pour une représentation plus claire de la structure du système, et plus facile à comprendre et à modifier, que le code.

6.7.1 Règles de génération de classes

Règle 1 : Pour chaque *InteractiveAgent IA*, on crée une classe qui hérite de la classe *InteractiveAgent* et qui reprend les attributs et les opérations de *IA*.

Règle 2 : Pour chaque *ATNBAsedBehavior AB*, on crée une classe qui hérite de la classe *ATNBAsedBehavior* et qui reprend les attributs et les opérations de *AB*.

Règle 3 : Pour chaque *ATN N* et si l'agent *A* auquel il est associé a été transformé en une classe *C*, alors :

1. on ajoute une opération *buildATN()* à la classe *C*.
2. on ajoute la ligne de code : "ATN atn = new ATN() ;"
3. on parcourt *N* à partir de son état initial *IS*, modélisé par un *State* lié à *N* par une association intitulée *initialS* : on ajoute à la liste *StatesList* le State *IS*
4. Pour chaque *State A* de *StatesList* alors :

   on invoque la règle 4 pour *A*.

   Pour chaque *Transition T* dont *A* est la source alors :

   - Si le *State B*, cible de *T* n'est pas encore transformé alors on invoque la règle 4 pour *B*.
   - on invoque le règle 5 pour la transition *T*.
   - on ajoute à *O* la ligne de code : *A.name*+".addTransition("+*T.name*+") ;" - Si B n'existe pas dans *StatesList* alors on ajoute *B* à *StatesList* et on revient à l'étape 4.
5. on ajoute à *O* la ligne de code : "return atn ;"

Règle 4 : Pour un *State S* d'un *ATN T* alors on génère la ligne de code "State "+S.name+" = new State() ;".

Règle 5 : Pour un *Transition T* dont la source est un *State A* et la cible est un *State B* alors :

1. on génère la ligne de code : "Transition "+*T.name*+" = new Transition("+*A.name*+", "+*B.name*+") ;"
2. pour chaque *Condition C* associé à *T* on invoque la règle 6.
3. pour chaque *Action N* associé à *T* on invoque la règle 7.

Règle 6 : Pour un *Condition C* de l'*ATN T* alors :

1. on génère la ligne de code "NamedCondition "+*C.name*+" = new NamedCondition("+C.name+") ;"
2. on ajoute une opération *O* booléenne au nom *C.name* à la classe *A* qui correspond à l'attribut *T.context*
3. s'il existe un *Message M* associé à la *Condition C* alors on ajoute à *O* la ligne de code : "return currentMessage.getPerforamtive()="+*M.name*+" ;"

Règle 7 : Pour un *Action A* de l'*ATN T* alors :

1. on génère la ligne de code "NamedAction "+*A.name*+" = new NamedAction("+*A.name*+") ;"
2. on ajoute une opération *O* au nom *A.name* à la classe *C* qui correspond à l'attribut *T.context*

6.7.2 Application : Emploi du temps

L'application des règles décrites précédemment sur le modèle PSM nous permet de générer les classes du système qui spécialise les classes de base de INAF.

La première étape consiste à construire les différentes classes *ATNBasedBehavior* par application de la règle 2. Dans notre application il existe deux *ATNBasedBehavior* : *SessionSeeker* et *SessionProvider*. La classe *SessionSeeker*, de la figure 6.26, étend la classe *ATNBAsedBehavior* dont elle implémente la classe statique *buildATN()*. Le code java de cette méthode est décrit dans une note associée à la classe *SessionSeeker*. Ce code est obtenu par application de la règle 3. Les méthodes *askForSessionReservation*, *informProviderLosers*, *informProviderWinner* et *RejectAllProposals* sont obtenues par application de la règle 7 sur l'ensemble des actions de l'ATN *SessionSeekerActivity*. Les méthodes booléennes *receivePropose*, *receiveRefuse*, *receiveInformDone* sont obtenues par application de la règle 6 sur l'ensemble des conditions de l'ATN *SessionSeeker*.

La deuxième étape consiste à construire les différentes classes *InteractiveAgent* par application de la règle 1. La figure 6.27 décrit la classe *Professor* qui héritent de la classe *InteractiveAgent* dont elle implémente la classe statique *buildATN()*. Le code java de cette méthode est décrit dans une note associée à la classe *Professor*. Ce code est obtenu par application de la règle 3. Les méthodes *defineNeededSession* et *startSessionSeeking* sont obtenues par application de la règle 7 sur l'ensemble des actions de l'ATN *ProfessorActivity*. Les méthodes booléennes *cond_ T1*, *cond_ T2* et *endSeekerRole* sont obtenues par application de la règle 6 sur l'ensemble des conditions de l'ATN *Professor*.

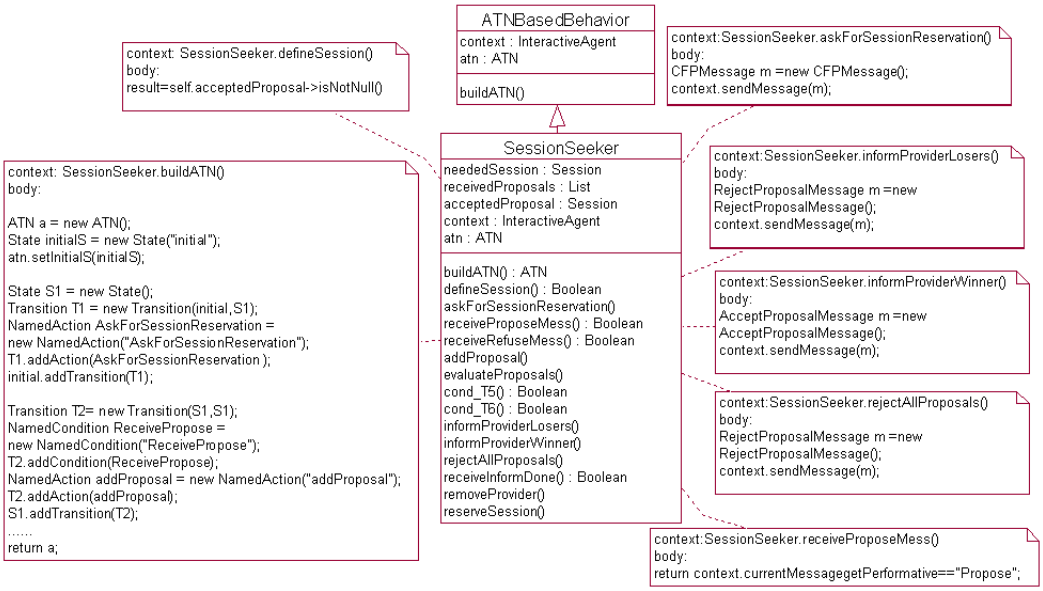


FIG. 6.26 – La classe de l'*ATNBasedBehavior SessionSeeker*

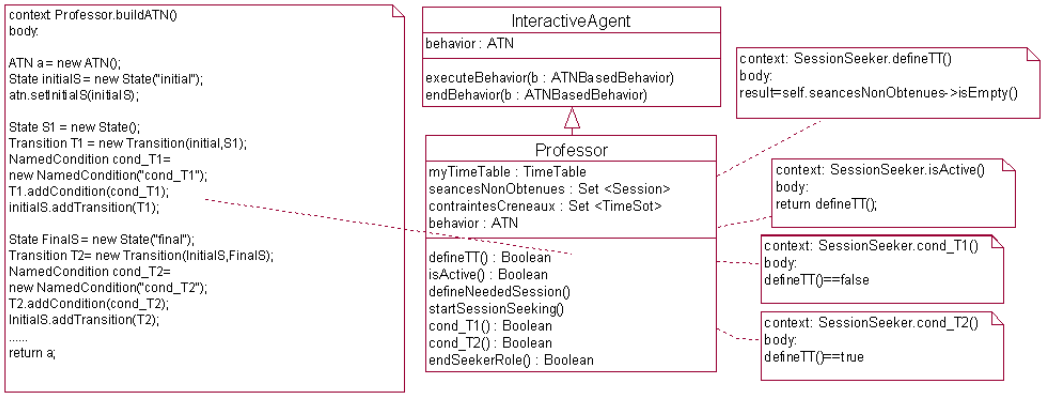


FIG. 6.27 – La classe *Professor*

6.8 Conclusion

Dans ce chapitre nous avons présenté une nouvelle méthode, appelée *MDAD* (*MOdel Driven Agent Development*), pour l'ingénierie des systèmes multi-agents. *MDAD* est basée sur l'approche de *MDA*, ainsi elle utilise plusieurs méta-modèles et connaissances pour transformer des représentations abstraites d'un système multi-agents en un système opérationnel. Ce chapitre a donné des exemples des méta-modèles pour les différents aspects multi-agents : domaine, organisation, agent et role.

Une première implémentation de notre méthode a été réalisée pour un exemple [TGPB03], avec l'outil de développement Metagen [RSBP95]. Le développement d'application avec MetaGen utilise deux types de transformation de modèles, l'un analogue à celui du PIM-PSM, et l'autre pour la génération de code. Dans notre travail, ces transformations sont réalisées par une base de règles de premier ordre, et le moteur d'inférence orienté objet Neopus. Dans cette implémentation des éditeurs de modèles sont automatiquement générés à partir de la description des méta-modèles. Ces éditeurs sont employés par le concepteur pour décrire des modèles. D'ailleurs, le code produit est conforme à la plate-forme cible qui doit l'exécuter et le déployer.

Ce premier développement avec *MDAD* a prouvé que cette nouvelle approche est intéressante et prometteuse qui améliore le processus de développement en multi-agents. Cependant, plus d'expériences avec des applications réelles sont nécessaires pour valider l'approche proposée. Nous travaillons actuellement pour le développement d'applications plus complexe afin de valider et enrichir les transformations par méta-modèles.

Chapitre 7

Conclusion générale

7.1 Introduction

Nous nous sommes intéressés dans ce travail à l'ingénierie des systèmes multi-agents, et plus particulièrement à la conception et l'implémentation des agents interactifs. Pour définir les interactions entre agents, différents protocoles d'interaction ont été proposés. Un protocole d'interaction est défini comme un pattern standard d'interaction entre deux ou plusieurs agents pour réaliser une tâche.

La majorité des travaux existants, notamment les travaux de la FIPA, décrivent les séquences de messages échangés et leurs contenus : performative, émetteur, receveur, etc. Les représentations proposées des protocoles d'interaction (voir les travaux de El Fallah [FSHM99], d'Odell [OPB00] et de Yoo [Yoo99]) offrent plusieurs facilités pour la conception formelle ou semi-formelle, et la validation des protocoles d'interaction. Cependant, l'étape de l'intégration des protocoles d'interaction dans la structure et l'activité de l'agent n'a pas été traitée. En effet, les implémentations existantes des agents qui utilisent les protocoles d'interaction nécessitent plus d'efforts et de savoir-faire du développeur. Cela explique l'implémentation de façon *ad-hoc* des protocoles d'interaction, qui ne favorise pas leur réutilisation.

De plus, la majorité des implémentations d'agents interactifs prévoit l'utilisation d'un seul protocole d'interaction, voir les architectures des agents proposées dans [SFJ00], [KM01] et [Rah02]. Par conséquent, le comportement interactif de l'agent est statique et peut devenir inadapté aux éventuelles évolutions de son environnement. En d'autres termes, l'agent ne peut pas changer dynamiquement de protocoles. Tous ces éléments font que le développement des agents interactifs est difficile et demande plus d'effort de la part du développeur.

Cette difficulté de développement, se pose aussi dans le niveau le plus général, celui du système multi-agents, malgré l'existence de plusieurs méthodologies de conception et de plates-formes multi-agents. Cela est du principalement à une incompatibilité entre les methodologies et les plates-formes multi-agents. La majorité des méthodologies existantes se limite à un seul modèle d'agent ou à un seul mode d'interaction. Malheureusement, la méthodologie ne permet pas de modéliser un système contenant, par exemple, différents modèles d'agents (cognitif, réactif, BDI, situé, etc). Une autre limite au niveau méthodologique est l'absence quasi-générale d'une démarche automatisée ou semi-automatisée qui aide le développeur dans l'implémentation de son modèle conceptuel. D'ailleurs, ce développement exige une maîtrise des aspects multi-agents, qui sont assez divers et complexes (coordination, interaction, organisation...). Tous ces éléments ont engendré une difficulté de passage d'une représentation conceptuelle du système multi-agents à son implémentation opérationnelle.

7.2 Contribution

Nous nous sommes appliqués dans ce travail à améliorer la développement des agents interactifs et à aller vers une adaptation du comportement interactif de l'agent. Pour ce faire, nous avons été amenés à :

- définir une représentation réutilisable des protocoles d'interaction,
- proposer une ontologie des protocoles d'interaction,
- développer un framework pour le développement des agents interactifs.

Nous nous sommes également intéressé à la conception des systèmes multi-agents et nous avons proposé une démarche orientée modèle pour le développement des systèmes multi-agents. Ces différents points sont résumés dans ce qui suit.

7.2.1 Une représentation componentielle des protocoles d'interaction

Convaincus que l'approche par composant est la solution la plus réaliste pour modifier, maintenir ou faire évoluer dynamiquement un système, nous avons proposé une représentation componentielle des protocoles d'interaction. Ainsi l'agent peut substituer, supprimer et ajouter de nouveaux protocoles d'interaction. Cette propriété est importante dans un environnement ouvert où les agents sont confrontés à des situations d'interaction non prévues au moment de leur conception. Cela rejoint les propos de Briot [Bri06] qui préconise une approche d'ingénierie *"proactive/optimiste"* qui offre aux agents la capacité de s'adapter aux fonctionnements imprévus, tenant compte ainsi de la mobilité et la dynamique de l'environnement.

L'analyse des protocoles d'interaction de FIPA nous a permis de réifier les principaux concepts de l'interaction, qui nous ont ainsi permis de définir une ontologie des protocoles d'interaction. Notre ontologie, baptisée ONTOPRO, regroupe les concepts pouvant être utilisés comme paramètres d'entrée pour l'instanciation des rôles d'interaction qui sont définis comme des composants réutilisables.

7.2.2 INAF : un framework pour les agents interactifs

Pour faciliter le développement des agents interactifs, nous avons développé le framework *INAF*, qui est une extension de la plate-forme *DIMA*. *INAF* propose une architecture modulaire de l'agent interactif qui facilite l'ajout des rôles d'interaction. Il offre aussi une bibliothèque de protocoles d'interaction de *FIPA*, dont les rôles d'interaction sont implémentés sous forme de composants. La spécification des rôles d'interaction se base sur l'ontologie d'interaction.

Nous avons montré à travers l'implémentation des deux benchmarks "gestion des emplois du temps" et "gestion d'une chaîne logistique", la facilité de développement qu'offre *INAF*.

7.2.3 MDAD : une démarche de développement à base de modèles

Notre avons proposé une nouvelle démarche pour l'ingénierie des systèmes multi-agents. *MDAD* se base sur l'approche *MDA*, offrant ainsi un ensemble de méta-modèles pour décrire le niveau conceptuel et le niveau d'implémentation. Pour permettre une conception plus diversifiée des aspects agents et multi-agents, MDAD permet d'intégrer plusieurs méta-modèles pour chaque aspect multi-agents selon la décomposition Voyelles. Dans ce rapport nous nous sommes limité a des exemples de méta-modèles d'agent et d'organisation. Cependant, plusieurs méta-modèles d'agent et d'organisation peuvent être intégrés. Au lieu d'être contraint par un seul méta-modèle, c'est le cas du travail de Bernon et al. [BCG$^+$04] qui proposent une unification de quelques méta-modèles, notre démarche permet au développeur de réutiliser les méta-modèles agents et multi-agents les mieux adaptés à son application.

De plus, plusieurs connaissances nécessaires pour transformer ces méta-modèles en un système multi-agents opérationnel ont été présentées sous forme de règles de transformation utilisant le langage *QVT*. Un premier exemple, celui du benchmark *"Gestion des emplois du temps"*, a été utilisé pour illustrer notre démarche.

7.3 Perspectives

Il est intéressant de noter que ce travail peut prévoir de nombreuses extensions, dont les principales sont :

7.3.1 Vers une méthodologie orientée modèle

Nous nous sommes limités dans notre méthode à l'étude des transformations de quelques méta-modèles. Les perspectives de l'étude des transformations sont nombreuses. Un premier travail serait de généraliser les transformations en considérant d'autres méta-modèles du PIM (méta-modèle d'agent, méta-modèle d'organisation, méta-modèle d'interaction...). Cela permettra d'avoir une bibliothèque de méta-modèles qui couvrent les différents aspects multi-agents. De plus, il serait nécessaire de proposer une validation formelle de ces transformations.

D'un point de vue processus de développement, nous avons étudié dans notre méthode le passage du PIM au PSM, qui constitue une partie du processus de développement de l'approche MDA. Pour aller vers une méthodologie complète qui couvre les différentes phases du cycle de développement, nous envisageons d'ajouter le modèle CIM (*Computation Independent Model*), ainsi que le passage du CIM au PIM. En outre, pour une meilleure representation des plates-formes, il serait intéressons d'intégrer au processus de développement le modèle PDM (*Platform Description Model*), qui sera utilisé comme un modèle source dans la transformation entre le PIM et le PSM.

7.3.2 Outiller la méthode MDAD

L'implémentation d'un premier exemple utilisant MDAD, a été réalisée avec l'outil de développement MetaGen, qui propose une démarche de transformation proche de l'approche MDA. Cependant, pour tirer profit de notre méthode *MDAD* nous devons développer un ensemble d'outils pour :

1. aider le développeur dans la spécification et la visualisation de ses modèles PIM et PSM,
2. permettre l'écriture des règles de transformation utilisant la syntaxe du langage QVT,
3. construire des modèles PSM par application des règles de transformation. Pour ce faire, il faut définir une stratégie d'application des règles, une taxonomie de ces stratégies est donnée dans [CH03].
4. générer une partie du code du système pour une plate-forme cible.

Bibliographie

[ACR+03] B. Appukuttan, T. Clark, S. Reddy, L. Tratt, and R. Venkatesh. A model driven approach to building implementable model transformations. October 2003. Workshop in Software Model Engineering (WiSME) 2003.

[AFT03] M. Amor, L. Fuentes, and J. M. Troya. A component-based approach for interoperability across FIPA-compliant platforms. In Matthias Klusch, Sascha Ossowski, Andrea Omicini, and Heimo Laamanen, editors, *Cooperative Information Agents VII, 7th International Workshop, CIA 2003, Helsinki, Finland, August 27-29, 2003, Proceedings*, volume 2782 of *Lecture Notes in Computer Science*, pages 266–280. Springer, 2003.

[AFV04] M. Amor, L. Fuentes, and A. Vallecillo. Bridging the gap between agent-oriented design and implementation using mda. In *Agent Oriented Software Engeneering Workshop, helt at AAMAS'04*, pages 93–108, 2004.

[Aus62] J. L. Austin. *How to Do Things With Words.* Oxford University Press, Oxford, 1962.

[BCG+04] C. Bernon, M. Cossentino, M. P. Gleizes, P. Turci, and F. Zambonelli. A study of some multi-agent meta-models. In *Agent Oriented Software Engineering Workshop helt at AAMAS'04*, pages 62–77, 2004.

[BCTR04] F. Bellifemine, G. Caire, T. Trucco, and G. Rimassa. *Jade Programmer's guide JADE 3.2*, July 2004.

[BD01] J.-P. Briot and Y. Demazeau, editors. *Principes et architecture des systèmes multi-agents.* Collection IC2. Hermes-lavoisier, 2001. in-french.

[BGGP03] C. Bernon, M.-P. Gleizes, P. Glize, and G. Picard. Problème de l'emploi du temps. Technical report, Groupe de travail ASA, 2003.

[Bla05] X. Blanc. *MDA en Action.* Eyrolles, 2005.

[BP00] F. Bergenti and A. Poggi. Exploiting UML in the design of multi-agent systems. In Andrea Omicini, Robert Tolksdorf, and Franco Zambonelli, editors, *Engineering Societies in the Agents World*, pages 106–113, December 2000.

[BP02] F. Bergenti and A. Poggi. *Agent-Oriented Software Construction with UML*, volume 2, pages 757–769. World Scientific, 2002.

[BPG+04] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos : An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3) :203–236, 2004.

[BPJ02] C. Bartolini, C. Preist, and N. R. Jennings. A generic software framework for automated negotiation. In *AAMAS'02*, Bologna, Italy, 2002. ACM.

[Bri06] J.-P. Briôt. *Software Engineering for Multi-agent Systems III*, foreword Foreword, pages i–ii. Springer, 2006.

[Cas98] C. Castelfranchi. Modelling social action for AI agents. *Artificial Intelligence*, 103 :157–182, 1998.

[CBP05] M. Cossentino, C. Bernon, and J. Pavon. Modelling and meta-modelling issues in agent oriented software engineering : The agentlink aose tfg approach. Technical Report Agent Link III, The AgentLink AOSE Technical Forum Group, Ljubljana, Slovenia, 28 February 2005.

[CCS04] A. Chella, M. Cossentino, and L. Sabatucci. Tools and patterns in designing multi-agent systems with PASSI. In *In 6th WSEAS International Conference on Telecommunications and Informatics*, Cancun, Mexico, May 2004. WSEAS.

[CCZ05] L. Cernuzzi, M. Cossentino, and F. Zambonelli. Process models for agent-based development. *Journal of Engineering Applications of Artificial Intelligence*, Elsevier. Vol.18(2) :205–222, March 2005.

[CH03] K. Czarnecki and S. Helsen. Classification of model transformation approaches. In *Workshop of generative techniques in the context of model-driven architecture, held at OOPSLA'03 Conference*, 2003.

[CKM01] J. Castro, M. Kolp, and J. Mylopoulos. A requirements-driven development methodology. *Lecture Notes in Computer Science*, 2068, 2001.

[Cos03a] M. Cossentino. Mas meta-model in passi. Technical report, Istituto di Calcolo e Reti ad Alte Prestazioni, Août 2003.

[Cos03b] M. Cossentino. Mas meta-model in passi. Technical report, Istituto di Calcolo e Reti ad Alte Prestazioni, 8 2003.

[CP02] M. Cossentino and C. Potts. A case tool supported methodology for the design of multi-agent systems. In *The 2002 International Conference on Software Engineering Research and Practice*, Las Vegas (NV), USA, June 24-27 2002. SERP'02.

[CP03] S. Cranefield and M. Purvis. Ontologies for interaction protocols. In *Workshop on Ontologies in Agent Systems, at AAMAS'03*, Tilburg, The Netherlands, 2003. CEUR Publications, KUB Tilburg University.

[Dem95] Y. Demazeau. From interactions to collective behaviour in agent-based systems. 1995.

[Dex00] *11th International Workshop on Database and Expert Systems Applications (DEXA'00), 6-8 September 2000, Greenwich, London, UK*. IEEE Computer Society, 2000.

[DLC87] E. H. Durfee, V. Lesser, and D. D. Corkill. Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers*, 36(11) :1275–1291, 1987.

[dSWL] L. P. de Silva, M. Winikoff, and W. Liu. Extending agents by transmitting protocols in open systems.

[dSWL03] L. Priyalal de Silva, M. Winikoff, and W. Liu. Extending agents by transmitting protocols in open systems. In *Challenges in Open Agent Systems'03 workshop, held at AAMAS'03*, Melbourne, Australia, 2003.

[EC04] L. Ehrler and S. Cranefield. Executing agent uml diagrams. pages 906–913, Washington, DC, USA, July 2004. Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04), IEEE Computer Society.

[FB03] J. Freire and L. Botelho. Executing explicitly represented protocols. Bologna, Italy, July 2003. In Workshop on Challenges in Open Systems, held at AAMAS'02.

[Fer95] J. Ferber. *Les systèmes multi-agents, vers une intelligence collective*. InterEditions, Paris, 1995.

[FFMM94] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an agent communication language. In *Third international conference on information and knowledge management*. ACM Press, November 1994.

[FG98] J. Ferber and O. Gutknecht. Alaadin : a meta-model for the analysis and design of organizations in mutli-agent systems. In Y. Demazeau, editor, *ICMAS'98*, pages 128–135, Paris, 1998.

[FGM03] J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations : An organizational view of multi-agent systems. In Paolo Giorgini, Jörg P. Müller, and James Odell, editors, *AOSE*, volume 2935 of *Lecture Notes in Computer Science*, pages 214–230. Springer, 2003.

[FIP02a] FIPA. Fipa acl message structure specification. Technical report, 2002.

[FIP02b] FIPA. Fipa interaction protocol specifications. Technical report, Foundation for Intelligent Physical Agents, 2002.

[fIPA04] Foundation for Intelligent Physical Agents. Fipa agent management specification. Technical Report 00023, 18/03/2004.

[FSHM99] A. El Fallah-Seghrouchni, S. Haddad, and H. Mazouzi. Protocol engineering for multiagent interactions. In *MAAMAW'99*, number 1647 in LNAI, pages 128–135. Springer Verlag, 1999.

[GB99a] M. Greaves and J. Bradshaw. Specifying and implementing conversation policies. In *Autonomous Agents Workshop*, Seattle, USA, May 1999.

[GB99b] Z. Guessoum and J.-P. Briot. From active objects to autonomous agents. *IEEE Concurrency*, 7(3) :68–76, 1999.

[GF00] O. Gutknecht and J. Ferber. The MADKIT agent platform architecture. In Thomas Wagner and Omer F. Rana, editors, *Agents Workshop on Infrastructure for Multi-Agent Systems*, volume 1887 of *Lecture Notes in Computer Science*, pages 48–55. Springer, 2000.

[GHB00] M. Greaves, H. Holmback, and J. Bradshaw. What is a conversation policy ? In *Issues in Agent Communication*, pages 118–131, London, UK, 2000. Springer-Verlag.

[GJM00] O. Gutknecht, J.Ferber, and F. Michel. Madkit : Une expérience d'architecture de plate-forme multi-agent générique. In *JFIADSMA'2000*. Hermès, 2000.

[GM98] R. H. Guttman and P. Maes. Cooperative vs. competitive multi-agent negotiations in retail electronic commerce. In *Proceedings of the Second International Workshop on Cooperative Information Agents*, Paris, July 1998.

[GMP02] F. Giunchiglia, J. Mylopoulos, and A. Perini. The tropos software development methodology : Processes, models and diagrams. In *AOSE'02*, pages 162–173, 2002.

[Gro02] Object Managment Group. Request for proposal : Mof 2.0 query / views / transformations rfp. Request for proposal ad/2002-04-10, OMG, 28 October 2002.

[Gro04] Agent Oriented Software Group. *JACK Intelligent Agents JACK Manual Release 4.1*, Mai 2004.

[Gro05] Object Managment Group. Mof qvt final adopted specification. Technical Report ptc/05-11-01, OMG, Novembre 2005.

[Gua98] N. Guarino. Formal ontology and information systems. In N. Guarino, editor, *Formal Ontology in Information Systems*, pages 3–18. IOS Press, Amsterdam, 1998.

[Gue03] Z. Guessoum. *Modèles et architectures d'agents et de systèmes multi-agents adaptatifs*. Habilitation à diriger des recherches, Université de Pierre et Marie Curie, 2003.

[Gut01] O. Gutknecht. *Proposition d'un modèle générique de systèmes multi-agents - Examen de ses consèquences formelles, implèmentatoires et méthodologiques*. PhD thesis, Université Montpellier 2, September 2001.

[Hew77] C. Hewitt. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8 :323–364, 1977.

[HOB04] M.-P. Huget, J. Odell, and B. Bauer. *The AUML Approach*, volume 11, chapter 1. Kluwer, 2004.

[Hug01] M.-P. Huget. *Une ingénierie des protocoles d'interaction pour les systèmes multi-agents*. Thèse de doctorat en informatique, Université de Paris IX - Dauphine, 15 juin 2001.

[Hug02] M.-P. Huget. Agent uml class diagrams revisited. In *In Proceedings of Agent Technology and Software Engineering*, Erfurt, Germany, October 2002.

[IGG99] C. Iglesias, M. Garrijo, and J. Gonzalez. A survey of agent-oriented methodologies. In Jörg Müller, Munindar P. Singh, and Anand S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555, pages 317–330. Springer-Verlag : Heidelberg, Germany, 1999.

[JFL+00] N.R. Jennings, P. Faratin, A.R. Lomuscio, C. Sierra, and M. Wooldridge. Automated negotiation : Prospects, methods and challenges. In *Pacific Rim International Conference on Artificial Intelligence*, 2000.

[JFN+00] N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, and B. Odgers. Autonomous agents for business process management. *Int. Journal of Applied Artificial Intelligence*, 14(2), 2000.

[JG06] T. Jarraya and Z. Guessoum. Reuse interaction protocols to develop interactive agents. In Jiming Liu and Benjamin W. Wah, editors, *Proceedings of IEEE Conference on Intelligent Agent technology : IAT'06*, Hong Kong, China, 18-22 Décembre 2006. IEEE Press.

[JGD04] T. Jarraya, Z. Guessoum, and E. Desjardin. Using generic interaction protocols to develop multi-agent systems. In *Conference on Advances in Intelligent Systems - Theory and Applications In cooperation with the IEEE Computer Society (AISTA'04)*, Luxembourg, Novembre 2004.

[JGD05] T. Jarraya, Z. Guessoum, and E. Desjardin. Towards adaptive interactive agents. In *Fifth European Workshop on Adaptive Agents and Multi-Agent Systems (AAMAS'05)*, Paris, France, Mars 2005.

[JGF04] T. Jarraya, Z. Guessoum, and N. Faci. Inaf : a new interactive agents framework. In Abdelmajib Ben Hamadou, Faïez Gargouri, and Mohamed Jmail, editors, *Proceedings of eighth International Maghrebian Conference on Software Engineering and Atificial Intelligence : MCSEAI'04*, pages 99–112, Sousse, Tunisie, 9-12 mai 2004. Centre de Publication Universitaire.

[Jou00] D. Jouvin. Utilisation de la délégation pour l'adaptation de protocole et le suivi de conversations entre agents. In *Journées Francophones sur l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents (JFIAD-SMA 2000)*, Saint-Jean-La-Vêtre, France, Octobre 2000. Hermes.

[Jou03] D. Jouvin. *Délégation de Rôle et architectures dynamiques de systèmes multi-agents conversationnels.* PhD thesis, Université de Lyon I - Claude Bernard, Décembre 2003.

[JPS02] T. Juan, A. Pearce, and L. Sterling. ROADMAP : extending the gaia methodology for complex open systems. In M. Gini, T. Ishida, C. Castelfranchi, and W. Lewis

Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, pages 3–10. ACM Press, 2002.

[Kad05] H. Kadima. *Conception orientée objet guidée par des modèles*. DUNOD, 2005.

[KM01] N. Karacapilidis and P. Moraitis. Building an agent-mediated electronic commerce system with decision analysis features. In *Decision Support Systems*, 2001.

[KP01] J.-L. Koning and S. Pesty. Modèles de conversation. In J.-P. Briot and Y. Demazeau, editors, *Principes et architecture des systèmes multiagents*, Collection IC2 (Information - Commande - Communication), Informatique et Systèmes d'Information. Hermès Sciences Publications, Paris, France, 2001.

[Kra01] S. Kraus. Strategic negotiation in multiagent environments. The MIT Press, Cambridge, MA, 2001.

[KWT04] M. Kang, L. Wang, and K. Taguchi. Modeling mobile agent applications in uml 2.0 activity diagrams. In *AOSE'04*, 2004.

[Maz01] H. Mazouzi. *Ingénierie des protocoles d'interaction : des Systèmes Distribués aux Systèmes Multi-Agents*. PhD thesis, Université de Paris IX - Dauphine, octobre 2001.

[MFSH02] H. Mazouzi, A. El Fallah-Seghrouchni, and S. Haddad. Open protocol design for complex interactions in multi-agent systems. In *AAMAS*, pages 517–526, Bologna, Italy, July 2002. ACM.

[MPS02] P. Moraitis, E. Petraki, and N. I. Spanoudakis. Engineering jade agents with the gaia methodology. In *Agent Technologies, Infrastructures, Tools, and Applications for E-Services*, pages 77–92, 2002.

[MV04] P. Mathieu and M.-H. Verrons. Three different kinds of negotiation applications achieved with GeNCA. In *Proceedings of the International Conference on Advances in Intelligent Systems - Theory and Applications (AISTA) In cooperation with the IEEE Computer Society*, Centre de Recherche Public Henri Tudor, Luxembourg-Kirchberg, Luxembourg, 15-18 novembre 2004.

[NNLC99] H. S. Nwana, D. T. Ndumu, L. C. Lee, and J. C. Collis. ZEUS : a toolkit and approach for building distributed multi-agent systems. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 360–361, Seattle, WA, USA, 1999. ACM Press.

[Occ03] M. Occello. *Methodologie et architectures pour la conception de systèmes multi-agents*. Habilitation à diriger les recherches, Université de Joseoh Fourier de Grenoble, Juillet 2003.

[OLHN04] J. Odell, R. Levy, M.-P. Huget, and M. Nodine. Fipa modeling tc : Agent class superstructure metamodel. Technical Report TBA, FIPA, 1 2004.

[OMG03] OMG. *Unified Modeling Language Specification version 1.5*, chapter 3, pages 135–155. March 2003. http ://www.omg.org/docs/formal/03-03-01.pdf.

[OPB00] J. J. Odell, H. V. Parunak, and B. Bauer. Representing agent interaction protocols in UML. In *AOSE Workshop held in the Fourth International Conference on Autonomous Agents*, 2000.

[orm01] OMG TC Document ormsc. Model driven architecture (mda). Technical report, OMG, 2001.

[Pas01] P. Pasquier. Communication entre agents jack 3.0 avec des messages fipa-acl. Technical report, DAMAS, Février 2001.

[PBH00] S. Poslad, P. Buckle, and R. Hadingham. FIPA-OS : the FIPA agent Platform available as Open Source. In Jeffrey Bradshaw and Geoff Arnold, editors, *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000)*, pages 355–368, Manchester, UK, April 2000. The Practical Application Company Ltd.

[PG04] G. Picard and M.-P. Gleizes. *The ADELFE Methodology - Designing Adaptive Cooperative Multi-Agent Systems*, volume Methodologies and Software Engineering for Agent Systems, chapter Chapter 8, pages 157–176. Kluwer, 2004.

[PGS03] Juan Pavón and Jorge J. Gómez-Sanz. Agent oriented software engineering with INGENIAS. In *CEEMAS*, pages 394–403, 2003.

[PV02] I. Partsakoulakis and G. Vouros. Importance and properties of roles in mas organization : A review of methodologies and systems, 2002.

[QSA03] J. G. Quenum, A. Slodzian, and S. Aknine. Automatic derivation of agent interaction model from generic interaction protocols. In *The Fourth International Workshop on Agent-Oriented Software Engineering AOSE-2003, held at AAMAS'03*, Melbourne, Australia, 2003.

[Que05] J. G. Quenum. *Conception et exécution d'interactions dans les SMA : Configuration et Sélection de protocoles génériques.* PhD thesis, Université de Paris VI Pierre et Marie Curie, Laboratoire d'Informatique de Paris 6, 12 2005.

[Rah02] I. Rahwan. Intelligent agents for automated one-to-many e-commerce negotiation. In *Computer Science 2002 of the ACS Series Conferences in Research and Practice in Information Technology*, volume 4, 2002.

[Rec94] M. Reck. Types of electronic auctions. In *the international conference on Information and communications technologies in tourism*, 1994.

[RSBP95] N. Revault, H. Sahraoui, G. Blain, and J-F Perrot. A metamodeling technique : The métagen system. In *TOOLS Europe'95*, pages 127–139, 1995.

[San99] T. W. Sandholm. Distributed rational decision making. In Gerhard Weiss, editor, *Multiagent Systems : A Modern Approach to Distributed Artificial Intelligence*, pages 201–258. The MIT Press, Cambridge, MA, USA, 1999.

[SBPL04] J. Sudeikat, L. Braubach, A. Pokahr, and W. Lamersdorf. Evaluation of agent - oriented software methodologies - examination of the gap between modeling and platform. In P. Giorgini, J. P. Müller, and J. Odell, editors, *Agent-Oriented Software Engineering V, Fifth International Workshop AOSE 2004*, pages 126–141. Springer Verlag, 7 2004.

[Sea69] J. R. Searle. *Speech Acts : An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, London, 1969.

[SF02] J. Gomez Sanz and R. Fuentes. The ingenias methodology. Seville, Spain, November 2002. Fourth Iberoamerican Workshop on Multi-Agent Systems, held at the VIII Iberoamerican Conference on Artificial Intelligence (IBERAMIA'02).

[SFJ00] C. Sierra, P. Faratin, and N. R. Jennings. A service oriented negotiation model between autonomous agents. In *Collaboration between Human and Artificial Societies - Coordination and Agent-Based Distributed Computing*, number 1624 in LNAI, pages 201–220. Springer Verlag, 2000.

[SGBP03] K. Somefun, E. Gerding, S. Bohte, and H. La Poutr. Automated negotiation and bundling of information goods. In *in Proceedings of the 5th Workshop on AgentMediated Electronic Commerce (AMEC V)*, Melbourne, Australia, July 2003.

[Smi81] R. G. Smith. The contract net protocol : High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12) :1104–1113, 1981.

[Tay95] A. D. Taylor. *Mathematics and Politics : Strategy, Voting, Power, and Proof*. Springer-Verlag, 1995.

[TGPB03] A. Thiefaine, Z. Guessoum, J.-F. Perrot, and G. Blain. Génération de systèmes multi-agents à partir de modèles. In *JFSMA'03*, 2003.

[TH03] S. Toivonen and H. Helin. Representing interaction protocols in daml. In Ludger van Elst, Virginia Dignum, and Andreas Abecker, editors, *Agent Mediated Knowledge Management, International Symposium*, volume 2926 of *Lecture Notes in Computer Science*, pages 310–321, Stanford, USA, 2003. Springer.

[TR03] J.-P. Tolvanen and M. Rossi. Metaedit+ : defining and using domain-specific modeling languages and code generators. In *OOPSLA Companion*, pages 92–93, 2003.

[Tve01] A. Tveit. A survey of agent-oriented software engineering. May 2001.

[TWD02] V. Tamma, M. Wooldridge, and I. Dickinson. An ontology based approach to automated negotiation. In *Fourth International Workshop on Agent-Mediated Electronic Commerce, AMEC-2002*, Bologna, Italy, 2002.

[Ver04a] M.-H. Verrons. *GeNCA : Un modèle général de négociation de contrats entre agents*. PhD thesis, Université des Sciences et Technologies de Lille, Novembre 2004.

[Ver04b] M.-H. Verrons. Une forme de négociation entre entités virtuelles. In *Conférence MajecSTIC'04*, 13-15 octobre 2004.

[Wag02] G. Wagner. A uml profile for external aor models. In *In Third International Workshop on Agent-Oriented Software Engineering*, Bologna, Italy, July 2002.

[WC01] M. Wooldridge and P. Ciancarini. Agent-Oriented Software Engineering : The State of the Art. In P. Ciancarini and M. Wooldridge, editors, *First Int. Workshop on Agent-Oriented Software Engineering*, volume Lecture Notes in AI Volume 1957, pages 1–28. Springer-Verlag, Berlin, 2001.

[WJK00] M. Wooldridge, N. R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3) :285–312, 2000.

[WWE98] P. R. Wurman, Michael P. Wellman, , and W. E.Walsh. The michigan internet auctionbot : A configurable auction server for human and software agents. In *international conference on autonomous agents (AGENTS'98)*, pages 301–308, Minneapolis, MN, May 1998.

[Yoo99] M.-J. Yoo. *Une approche componentielle pour la modélisation d'agents coopératifs et leur validation.* PhD thesis, Université de Paris 6, 1999.

[Yu95] E. Yu. *Relationships for Process Reengineering.* PhD thesis, University of Toronto, Departement of Computer Science, Toronto, USA, 1995.

[ZJW03] F. Zambonelli, N.R. Jennings, and M. Wooldridge. Developing multiagent systems : The gaia methodology. *Software Engineering and Methodology*, 12(3) :317–370, Juillet 2003.