GeNCA : un modèle général de négociation de contrats entre agents

THÈSE

présentée et soutenue publiquement le 2 Novembre 2004

pour l'obtention du

Doctorat de l'Université des Sciences et Technologies de Lille

(spécialité informatique)

par

Marie-Hélène VERRONS

Composition du jury

Président :	Pr. Jean-Paul DELAHAYE	LIFL, Université des Sciences et Technologies de Lille
Rapporteurs :	Pr. Jean-Paul BARTHÈS	Université de Technologie de Compiègne
	Pr. René MANDIAU	LAMIH, Université de Valenciennes et du Hainaut-Cambraisis
Examinateur :	MdC. Samir AKNINE	LIP6, Université de Paris VI
Directeur :	Pr. Philippe MATHIEU	LIFL, Université des Sciences et Technologies de Lille

UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE
Laboratoire d'Informatique Fondamentale de Lille — UMR 8022
Bâtiment M3 – 59655 VILLENEUVE D'ASCQ CEDEX
Tél. : +33 (0)3 28 77 85 41 – Télécopie : +33 (0)3 28 77 85 37 – email : direction@lifl.fr

Mis en page et composé avec LaTeX et la classe thlifl.

À mes parents

Remerciements

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

Je tiens à remercier tout d'abord Jean-Paul Barthès et René Mandiau pour avoir accepté de rapporter mon document de thèse et de m'avoir apporté leur regard sur mon travail. Je remercie également Samir Aknine pour avoir examiné ce document et Jean-Paul Delahaye pour avoir accepté de présider mon jury de thèse et pour l'intérêt qu'ils ont porté à mes travaux.

Je ne serais pas arrivée jusque là si Philippe Mathieu, mon directeur de thèse, ne m'avait pas pris sous son aile bienveillante depuis le DEA. Je le remercie également pour toute l'attention qu'il m'a portée et son soutien inconditionnel au cours de ces trois années. Nos nombreuses discussions et nos échanges de points de vue et d'idées m'ont permis d'avancer dans mes recherches. Pour lui, un chercheur est une *force de proposition*. J'espère ne pas l'avoir trop déçu.

Je remercie également tous les membres de l'équipe SMAC pour leur accueil et leur disponibilité. J'ai trouvé dans cette équipe une chaleur humaine qui a fait d'elle une seconde famille, *ma famille de recherche*. Je remercie plus particulièrement Jean-Christophe qui fut mon tuteur de monitorat et à ce titre m'a beaucoup aidée dans mes débuts en tant qu'enseignante, ainsi que Yann et Bruno pour leur aide précieuse lorsque j'avais des problèmes tant logiciels que matériels.

Je remercie également tous les membres du Laboratoire d'Informatique Fondamentale de Lille qui m'ont accueillie et qui m'ont permis de m'épanouir aussi bien dans mes recherches que dans mes enseignements.

Je remercie enfin ma famille et plus particulièrement mes parents, qui m'ont toujours soutenue dans mes études. Mes pensées vont également vers Frédéric, mon compagnon, pour m'avoir suivie et soutenue tout au long de ma thèse et qui a partagé mes doutes et mes joies durant ces trois années.

Table des matières

Table des figures

Chapitre 1

Introduction

Notre travail se positionne au sein de l'intelligence artificielle et plus particulièrement l'intelligence artificielle distribuée et les systèmes multi-agents. Lorsque plusieurs agents interagissent, des conflits peuvent survenir, ce qui nécessite l'utilisation de mécanismes de résolution de conflits. Parmi ces mécanismes, on trouve notamment la coordination, les systèmes de vote et la négociation. Nous nous proposons d'étudier dans cette thèse la négociation entre agents et de concevoir un modèle général de négociation ainsi qu'une API l'implémentant.

La négociation est un processus grâce auquel plusieurs parties aboutissent à une décision commune. Les parties verbalisent en premier lieu leurs demandes et convergent vers un accord par une succession de concessions ou une recherche de nouvelles alternatives [Pruitt, 1981]. La négociation est utilisée dans le domaine du commerce électronique notamment en utilisant les enchères, dans le domaine des télécommunications par exemple pour partager une bande passante et dans les systèmes multi-agents (SMA) pour l'allocation de tâches et de ressources [Durfee and Lesser, 1989, Sandholm, 1993], l'identification de conflits [Gamble and Sen, 1994], la résolution des disparités entre les buts [Sycara, 1988, Klein, 1991] et la détermination de la structure organisationnelle ; tout cela influence la cohérence de la société d'agents.

La négociation se décompose en deux catégories principales : la négociation *compétitive* et la négociation *coopérative*. La négociation compétitive intervient entre des agents compétitifs, qui essaient de maximiser leur utilité locale. Par contre, lors d'une négociation coopérative, les agents essaient d'atteindre l'utilité globale maximale, qui prend en compte la valeur de toutes leurs activités. Cette forme de négociation qui est très différente de la négociation compétitive, peut être vue comme un processus de recherche distribué.

La négociation dont nous traitons dans cette thèse est la négociation *compétitive*, puisque nous nous intéressons à la négociation entre agents essayant d'obtenir la meilleure solution possible pour eux.

Selon Jennings et al. [Jennings et al., 2000], la recherche sur la négociation automatique peut être décomposée en trois larges thèmes :

- *Les protocoles de négociation* : l'ensemble des règles qui gouvernent l'interaction. Cela couvre les types de participants autorisés, les états de la négociation, les évènements causant le changement d'état de la négociation et les actions valides pour un participant selon l'état courant de la négociation.
- *Les objectifs de la négociation* : l'intervalle de critères pour lesquels une solution doit être atteinte. À un extrême, l'objectif peut contenir un seul critère (comme un prix), mais il peut également contenir des centaines de critères (relatifs au prix, à la qualité, au temps, aux pénalités, aux termes et conditions, etc.). Orthogonalement à la structure de l'accord et déterminé par le protocole de négociation, se trouve le problème des types d'opérations permis sur les accords. Dans le cas le plus simple, la structure et le contenu de l'accord sont fixés et les participants peuvent soit l'accepter, soit le refuser (ie. "à prendre ou à laisser"). À un niveau supérieur, les participants ont la possibilité de changer les valeurs des critères de l'objectif de la négociation (ie. ils peuvent formuler des contre-propositions pour s'assurer que l'accord satisfasse mieux leur objectif de négociation). Finalement, les participants peuvent être autorisés à changer dynamiquement la structure de l'objectif de la négociation, ce qui permet d'ajouter ou de retirer un attribut en cas de blocage (par exemple, une extension de garantie offerte par le vendeur peut conclure une vente).
- *Les modèles décisionnels des agents* : le moyen par lequel un agent atteint ses objectifs de négociation tout en suivant les règles de la négociation. La sophistication du modèle tout comme l'intervalle des décisions qui doivent être prises, sont influencés par le protocole utilisé, par la nature de l'objectif de la négociation et par l'intervalle d'opérations qui peuvent lui être appliquées.

L'importance relative de ces trois thèmes varie selon le contexte de négociation et le contexte environnemental.

Notre recherche se place dans le domaine des protocoles de négociation. Notre objectif est de fournir un protocole général de négociation permettant de réaliser différentes applications de négociation, comme un système d'enchères ou un système de prise de rendez-vous. Nous souhaitons fournir un modèle général de négociation, et son implémentation, ce qui permet de l'utiliser plus facilement pour créer une application de négociation. Nous prenons en compte les deux autres domaines de recherche, mais sans approfondir le sujet. En effet, nous utilisons une interface avec les stratégies de négociation à utiliser afin de pouvoir séparer le processus décisionnel de l'agent, qui doit

être adapté au problème traité. Les objectifs de la négociation utilisés dans notre modèle sont des contrats portant sur des ressources. Ils représentent l'objet minimal qui sera négocié. Les utilisateurs sont ensuite libres d'y ajouter d'autres critères, comme un prix, une qualité de service, etc. Nous ne figeons donc pas les domaines portant sur les objectifs de la négociation et les modèles décisionnels des agents, nous souhaitons dans un cadre de génie logiciel apporter un système ouvert à ces domaines.

Différents travaux ont été réalisés dans le thème de la négociation automatisée et plusieurs systèmes de négociation ont vu le jour. Cependant, aucun ne permet de traiter l'ensemble des différentes formes de négociation les plus utilisées et la plupart sont dédiés à un type d'application, comme le commerce électronique, par exemple. Nous présentons dans cette thèse les principaux travaux en la matière, en commençant par le plus connu et reconnu : le *Contract Net Protocol* proposé par Smith en 1980 pour l'allocation de tâches dans un réseau. Un manager veut déléguer une tâche et fait un appel d'offres pour connaître les différents nœuds du réseau prêts à l'effectuer pour son compte. Il collecte les offres des différents contractants et choisit celle qui lui satisfait le mieux. La tâche est allouée au contractant associé à la meilleure offre. Le manager attend ensuite les résultats de l'exécution de cette tâche. Ce protocole est à la base de quasiment tous les travaux sur les protocoles de négociation, y compris les nôtres. Nous détaillons également différents travaux réalisés aussi bien dans le milieu universitaire, comme *Kasbah* du MIT, *Magnet* de l'université du Minnesota, *Fishmarket* de l'institut de recherche en intelligence artificielle en Espagne, aussi bien que dans le milieu industriel, comme *Zeus* de British Telecoms ou *SilkRoad* d'IBM. L'étude de ces différents travaux nous a permis de dégager un ensemble de caractéristiques fondamentales et ainsi de formuler un tableau comparatif des différents systèmes de négociation que nous avons étudiés en prenant en compte les critères qui nous semblent les plus importants pour un modèle général de négociation. Nous constatons qu'aucune plateforme présentée ne permet de prendre en compte tous les critères que nous avons définis, ni ne permet l'adaptation à l'ensemble des types de négociations que nous présentons dans cette thèse.

La négociation peut prendre différentes formes, de la négociation la plus basique à la plus complexe. Le niveau le plus faible en terme de négociation est un système dans lequel cette négociation est réduite à sa plus simple expression. C'est notamment le cas dans la négociation dite *à prendre ou à laisser* dans laquelle une offre est formulée qui sera soit acceptée, soit refusée, mais sans autoriser de révision. La négociation s'arrête après cette offre. Le *Contract Net Protocol* est une négociation de niveau plus élevé, puisqu'il consiste en un appel d'offres de la part d'un agent appelé manager. Différentes offres sont alors proposées au manager par des contractants, mais seulement l'une d'elles est choisie par le manager sans autre formulation si aucune offre ne convient. La négociation telle que nous la concevons commence à un niveau supérieur, où plusieurs offres

sont échangées, des contre-propositions sont formulées et des concessions sont faites par les différentes parties impliquées. Les enchères font partie de ces négociations à un niveau certes élémentaire, puisqu'aucune concession n'est faite par l'une des parties et surtout parce que le résultat n'est satisfaisant que pour le vendeur et l'acheteur ayant gagné l'enchère, les autres participants n'ayant rien obtenu. Les systèmes de vote font également partie des négociations dans une forme élémentaire puisqu'une seule intervention des participants est nécessaire pour réaliser la procédure de vote. Les négociations combinées, les négociations multi-niveaux et les négociations par argumentation constituent un niveau supérieur dans les différentes formes de négociation.

De l'étude de ces différentes formes de négociation, nous avons dégagé plusieurs points communs qui existent entre elles, ainsi que différentes propriétés possédant des valeurs variables selon la négociation effectuée. Certains sont évidents comme la présence des participants et de ressources (objets de la négociation), le nombre de propositions qui sont faites et la possibilité de formuler des contre-propositions, d'autres le sont moins comme le nombre minimal d'accords nécessaires pour le succès de la négociation, le délai d'attente des réponses et la possibilité de se rétracter. Ces points communs et ces propriétés nous ont fourni une base pour l'élaboration de notre modèle général. En étudiant le déroulement de ces différentes négociations, nous avons également repéré un ensemble minimal d'actes de langages nécessaires pour réaliser l'ensemble de ces négociations. Cet ensemble minimal, contenant entre autres la proposition d'une offre, son acceptation ou son refus, nous a permis de concevoir notre protocole général de négociation, paramétrable afin d'obtenir un protocole spécifique pour une négociation donnée. Les paramètres correspondent aux différentes propriétés des négociations.

Notre proposition de modèle général de négociation a plusieurs objectifs, dont la généricité, la portabilité, l'uniformisation et l'automatisation des envois de messages. Nous avons en effet l'intention de fournir un modèle de négociation permettant de réaliser plusieurs formes de négociation différentes, sans demander de lourde charge de travail à un utilisateur. Ce modèle prendra toute son ampleur grâce à une implémentation portable de celui-ci, offrant ainsi à l'utilisateur la possibilité de l'utiliser concrètement dans son environnement de travail habituel, sans avoir à installer de nouveaux composants logiciels dépendants de son système. L'uniformisation est une notion très importante dans ce domaine de recherche, car à notre connaissance, il y a très peu de travaux sur ce sujet. Seules une classification, dite de Londres [Benyoucef, 2000] et une taxonomie dite de Montréal [Ströbel and Weinhardt, 2003, Neumann et al., 2003] tentent de répondre au problème de la caractérisation des négociations. Quant à la complexité des négociations en terme d'envois de message, l'étude que nous avons réalisée sur notre protocole montre que dans le pire des cas, lorsqu'une négociation nécessite des renégociations en cascade impliquant plusieurs fois les mêmes personnes, avec un message envoyé par destinataire, la complexité est exponentielle en $O(2^n)$. La complexité d'une négociation qui ne nécessite aucune renégociation est quant à elle

linéaire. Ces résultats sur la complexité montrent le besoin d'automatiser l'envoi des messages lors d'une négociation.

Le modèle général de négociation que nous proposons, appelé *GeNCA* (*Generic Negotiation of Contracts API*), repose sur une architecture à trois niveaux, séparant ainsi les différentes parties d'une application, qui sont la partie communication, la partie négociation et la partie stratégie.

En effet, ces parties sont indépendantes, car la façon dont communiquent les agents n'a pas d'incidence sur le processus de négociation : que les agents communiquent par envoi d'e-mails ou au sein d'un SMA, le résultat de la négociation sera le même. C'est donc une partie totalement dépendante de l'application réalisée, selon qu'elle soit destinée à des agents distribués au sein d'un SMA qui possède ses propres méthodes de communication, ou à des agents disséminés dans le monde entier. C'est pourquoi nous avons choisi de séparer cette partie concernant la communication, en indiquant simplement le type des messages qui doivent être envoyés et reçus. Ceci permet à notre modèle d'être utilisable dans n'importe quelle architecture avec n'importe quel type de communication.

En ce qui concerne la partie de négociation, elle forme le cœur de notre modèle. C'est la partie la plus importante que nous développons car c'est elle qui constitue mon travail de recherche. Elle contient notre protocole général de négociation qui permet d'effectuer n négociations de 1 vers m agents simultanément, de faire des contre-propositions, de se rétracter et de renégocier automatiquement les contrats qui doivent l'être. Tout cela se paramètre dans un fichier de configuration du protocole.

C'est dans cette partie que toutes les données nécessaires à la modélisation des négociations sont définies comme par exemple les ressources, les contrats et les listes de priorité sur les ressources et sur les participants.

Ce niveau de négociation contient également les moyens de gérer les négociations auxquelles un agent participe, ce qui permet de bloquer les négociations de contrats entrant en conflit avec une négociation en cours et d'exécuter simultanément les autres négociations. Des mécanismes sont mis en place pour éviter les attentes infinies de réponse de la part d'agents qui n'existeraient plus et pour éviter les interblocages qui pourraient survenir si les résultats de deux négociations dépendent l'un de l'autre. Les négociations se déroulent donc sans survenue de deadlocks.

La partie stratégie de négociation est également une partie indépendante de notre modèle car non seulement il doit être possible d'utiliser différentes stratégies pour une application donnée, mais surtout parce que selon le type de négociation utilisé, la stratégie est différente. On ne négocie pas de la même manière pendant des enchères anglaises où les prix augmentent, que lors d'enchères hollandaises où les prix descendent. Ceci est encore plus évident lorsqu'on compare des négociations n'impliquant pas les mêmes critères, comme pour le cas d'une négociation commerciale utilisant la notion

de qualité de service, par exemple pour l'utilisation d'une partie de la bande passante d'un réseau ; et le cas d'une négociation non commerciale par exemple pour prendre des rendez-vous. Nous avons donc mis en place une interface entre notre niveau de négociation et notre niveau stratégique, afin que ces niveaux puissent interagir tout en étant interchangeables.

Le modèle général de négociation que nous proposons, appelé lui aussi *GeNCA*, est implémenté sous la forme d'une API Java, ce qui lui confère la portabilité que nous souhaitions. Cette implémentation reprend la partition en trois niveaux de notre modèle. Nous avons donc trois paquetages : communication, négociation et stratégie, auxquels nous avons ajouté un paquetage pour une interface graphique afin de faciliter l'interaction avec un utilisateur humain.

Le paquetage concernant la négociation est prêt à l'emploi et contient toutes les structures de données nécessaires à la négociation des différents contrats proposés par et aux agents. Il utilise des fichiers de configuration au format XML afin de spécialiser le protocole général et de connaître les ressources qui seront négociées ainsi que les stratégies qui seront utilisées par l'agent pour négocier. Les paquetages communication et stratégie sont quant à eux constitués des interfaces que nous avons mentionnées auparavant. Le paquetage stratégie comprend néanmoins une implémentation par défaut pour les stratégies des rôles d'initiateur et de participant. Les implémentations du niveau de communication que nous fournissons afin de pouvoir utiliser directement cette API correspondent chacune à un paquetage distinct. Dans chacun de ces paquetages, on trouve une implémentation de la couche de communication, mais également l'implémentation d'un agent utilisant ce mode de communication et une classe principale permettant de lancer l'application. Parmi ces implémentations, on trouve l'utilisation de deux SMA que sont Magique et Madkit, mais également des agents communicant par envoi d'e-mails ou encore des agents centralisés parlant à tour de rôle.

GeNCA a été utilisée pour concevoir différentes applications de négociation. Parmi ces applications, deux systèmes de vente aux enchères ont été réalisés, l'un concernant des enchères hollandaises (le vendeur propose un article à un prix élevé et diminue ce prix tant que personne n'accepte l'offre ; le premier acheteur à accepter remporte l'enchère au prix courant), l'autre des offres scellées sur plusieurs tours (chaque participant propose une offre de façon privée, elle est donc inconnue des autres participants ; le processus est itéré jusqu'à ce qu'une offre satisfasse le vendeur). Notre protocole permet donc de gérer des enchères, que les prix soient proposés par le vendeur ou par les acheteurs.

Nous avons également réalisé des applications ne faisant pas référence au commerce électronique. Ces applications sont un système de création d'emplois du temps, un système de prise de rendez-vous, un jeu de négociation et un système pour négocier

le choix d'un restaurant pour une sortie commune. L'application de création d'emplois du temps implique des enseignants et des groupes d'étudiants. C'est une application que nous avons réalisé dans le cadre du groupe ASA du GDR-I3, afin de comparer les différentes approches de ce problème. Un système de négociation pour résoudre ce problème n'est pas une solution évidente a priori, mais nous montrons que cette possibilité existe et donne de bons résultats [ASA, 2003]. L'application de prise de rendez-vous permet aux utilisateurs de négocier les horaires de leurs réunions et de renégocier automatiquement celles-ci si le rendez-vous trouvé ne convenait plus et qu'il faille en changer. En ce qui concerne l'application de choix d'un restaurant, les utilisateurs utilisent leurs préférences sur les lieux disponibles pour convenir d'un restaurant aimé de tous pour leur sortie commune. Ils peuvent ainsi décider de ne pas aller deux fois de suite au même restaurant, par exemple. Pour réaliser cette application, un système de vote majoritaire peut paraître plus approprié. Dans ce type d'approche, chaque participant donne une voix à un restaurant et le restaurant plus plébiscité est alors choisi. Il y a donc de fortes chances que le même restaurant soit choisi pour chaque sortie et la satisfaction des participants n'est de ce fait pas garantie. En effet, si l'on prend un exemple avec 5 personnes et 4 restaurants et que chaque restaurant obtient une voix sauf l'un d'eux qui en obtient deux, c'est ce restaurant qui sera choisi alors qu'il n'y a que 2 participants sur les 5 qui l'ont choisi. Un système de négociation permet en revanche de tenir compte des avis de chacun et d'aboutir à une solution plus satisfaisante pour les participants. Chaque personne étudie la proposition de restaurant faite par l'un des participants et celui-ci collecte les réponses et essaie de trouver un restaurant convenant au mieux à l'ensemble des personnes. Cela permet d'éviter la situation présentée auparavant, si l'on a fixé qu'au moins la moitié des personnes doivent être satisfaites avec le choix du restaurant effectué.

Notre modèle s'adapte à différentes sortes de négociations, commerciales ou non, ce qui était l'un des objectifs que nous nous étions fixés. L'implémentation de ce modèle est portable grâce à la technologie Java et nous avons réalisé des applications s'exécutant sur différentes plates-formes multi-agents comme Magique et Madkit et différents environnements tels que Windows™ et Linux.

De plus en plus de ventes dans l'immobilier se font aux enchères et de nombreux groupes industriels des domaines de la vente par correspondance et de l'industrie agro-alimentaire utilisent les enchères inversées pour attribuer les marchés[1] à leurs fournisseurs. Un modèle général de négociation ainsi qu'une implémentation de ce modèle facile à utiliser serait donc la bienvenue. Nos travaux de recherche se placent dans ce contexte.

[1]Le nouveau terme e-sourcing est d'ailleurs maintenant utilisé pour décrire ce type d'application

Structuration du document

Nous avons choisi de structurer ce document en 4 parties qui traitent tout d'abord de la définition de la négociation ainsi que d'un état de l'art (chapitre 2), puis de notre proposition de modèle général pour la négociation entre agents (chapitre 3), puis de la réalisation de ce modèle en une API Java (chapitre 4) et enfin de différentes applications de négociation que nous avons conçues afin de valider notre modèle (chapitre 5).

Dans le chapitre 2, nous présentons la définition de la négociation qui est usuellement utilisée par les chercheurs du domaine, puis nous proposons une taxonomie des systèmes de négociation automatique les plus courants et nous présentons différentes plates-formes de négociation qui ont vu le jour. Tous ces travaux s'appuient sur le modèle général du *Contract Net* qui a sans aucun doute été le précurseur des travaux en négociation automatique.

Avec les progrès des technologies de l'information, des systèmes multi-agents (SMA) et des places de marché électroniques, le besoin d'agents logiciels capables de négocier avec les autres à la place de leur utilisateur devient de plus en plus important. C'est pourquoi de nombreux travaux ont été accomplis dans ce domaine. La plupart des formes de négociation décrites dans ce chapitre sont dédiées au commerce électronique, qui connaît un essor important de nos jours.

Dans le chapitre 3, nous présentons notre modèle de négociation général appelé *GeNCA*, basé sur une architecture à trois niveaux : communication, négociation, stratégie et permettant de traiter différentes formes de négociations. Nos objectifs portent principalement sur quatre dimensions : la généricité, la portabilité, l'uniformisation et l'automatisation des envois de messages.

Le besoin d'un modèle général de négociation provient de la richesse de types de négociation différents. Comme nous le montrons dans ce chapitre, certaines notions communes aux différentes négociations peuvent être réifiées au sein d'un modèle général. Cependant, il n'existe pas de framework générique de négociation reconnu par tous les chercheurs du domaine. Notre objectif est donc de fournir ce framework, offrant ainsi à un concepteur d'applications de négociation un outil facilitant la spécification de son application.

Nous présentons dans le chapitre 4 l'implémentation de ce modèle que nous avons réalisé sous la forme d'une API Java. Elle reprend l'architecture en trois niveaux, qui correspondent à trois paquetages différents. Nous détaillons l'implémentation de chacun de ces paquetages et plus particulièrement celui concernant la négociation, qui

forme le cœur de notre modèle. Nous y présentons également le paquetage concernant l'interface utilisateur que nous avons élaborée afin qu'un utilisateur humain puisse interagir avec son agent durant les négociations. Enfin, nous présentons la façon d'utiliser notre paquetage pour la réalisation d'une application.

Le but de cette réalisation est de faciliter le développement d'applications de négociation avec notre modèle général. Comme le veut notre modèle, cette réalisation permet d'utiliser n'importe quel type de communication entre agents (par sockets, par e-mail, etc.) grâce à la définition d'une interface de communication et de choisir la stratégie de négociation à adopter pour l'application là encore à travers l'implémentation de l'interface de la couche stratégique.

Nous présentons dans le chapitre 5 trois applications parmi celles que nous avons réalisées avec *GeNCA* qui utilisent un mode de communication fondamentalement différent. La première utilise la communication par envoi d'e-mails, la deuxième utilise la communication au sein d'un SMA et la troisième utilise une communication synchrone où les agents parlent à tour de rôle. La première application est un système de prise de rendez-vous où les ressources communes à tous les participants. La seconde est une application de ventes aux enchères qui utilise par conséquent des ressources individuelles aux participants. La troisième application est un jeu de négociation de ressources appelé JNego qui implique des ressources communes à tous les participants. Ces applications peuvent être téléchargées à l'adresse suivante :

`http://www.lifl.fr/SMAC/projects/genca`.

La réalisation de ces différentes applications de négociation permet de valider notre modèle de négociation. En effet, notre protocole général est adapté à ces applications qui concernent différents types de négociation : les négociations commerciales comme les enchères et les négociations à but non commercial comme la prise de rendez-vous ou le choix d'un restaurant. Nous avons également réussi à utiliser différents moyens de communication entre les agents soit en les plaçant dans un SMA tels que Magique ou Madkit et en utilisant les moyens de communication qui leur sont propres, soit en utilisant des méthodes de communication telles que l'envoi d'e-mails pour des agents physiquement décentralisés ou encore l'appel de procédure pour des agents centralisés. Différentes stratégies ont également été conçues pour ces applications, puisqu'elles impliquent différents paramètres et qu'il était donc impossible d'utiliser la même stratégie pour chacune d'entre elles.

Nous terminons par la conclusion sur nos travaux de recherche et les perspectives que nous avons pour ce projet. Parmi ces perspectives, la représentation externe du protocole de négociation est celle qui nous paraît la plus importante afin de permettre sa réification et son passage au niveau des paramètres de l'API. Nous souhaitons trouver

un formalisme de description du protocole qui soit assez clair pour être lisible par un humain et facilement intégrable du point de vue génie logiciel. Ceci nous permettrait d'extraire notre protocole de notre modèle afin de pouvoir proposer à l'utilisateur une bibliothèque de protocoles de négociation utilisables avec *GeNCA*.

Chapitre 2

Définition et État de l'art

Avec les progrès des technologies de l'information, des systèmes multi-agents (SMA) et des places de marché électroniques, le besoin d'agents logiciels capables de négocier avec les autres à la place de leur utilisateur devient de plus en plus important. Dans ce chapitre, nous présentons la définition de la négociation usuellement considérée par les chercheurs du domaine, puis nous proposons une taxonomie des systèmes de négociation automatique, afin de montrer qu'il est possible de définir une négociation par un ensemble de paramètres. La plupart des formes de négociation décrites ici sont dédiées au commerce électronique, qui est un domaine qui connaît un essor important de nos jours. Dans la troisième section de ce chapitre, nous décrivons les principales plateformes de négociation qui ont vu le jour. Nous commencerons bien évidemment par présenter le *Contract Net Protocol* qui est sans aucun doute le plus utilisé en matière de négociation par la communauté qui s'intéresse à ce sujet.

2.1 Définition

La définition suivante résume notre point de vue :

Définition 1 *il y a négociation lorsqu'il y a une discussion, des propositions entre les protagonistes et lorsque l'accord final satisfait au mieux tous les participants.*

Il y a probablement autant de définitions similaires de la négociation que de chercheurs dans ce domaine. L'une des plus basiques et succinctes est formulée par Bussman et Muller [Bussmann and Muller, 1992] :

> *"negotiation is the communication process of a group of agents in order to reach a mutually accepted agreement on some matter"*

Tous les chercheurs s'accordent sur la finalité de la négociation, à savoir l'aboutissement à un accord commun satisfaisant. Mais la négociation est elle-même définie comme un *processus*. Toute la diversité des recherches en négociation provient de ce mot : processus. La négociation peut donc être vue comme une boîte noire ayant en entrée un conflit et en sortie un accord, dans le meilleur des cas. La recherche sur la négociation consiste donc à étudier les mécanismes de cette boîte noire, pour la rendre transparente.

Cette définition de la négociation a été reprise par Jennings et al. dans [Jennings et al., 2000] :

> *"Negotiation is the process by which a group of agents come to a mutually acceptable agreement on some matter.(...) to make proposals, trade options, offer concessions and (hopefully) come to a mutually acceptable agreement."*

Ils y indiquent également que le minimum requis pour un agent négociant est la possibilité de faire et de répondre à des propositions et de pouvoir indiquer son insatisfaction avec les propositions qu'ils trouvent inacceptables. Les propositions peuvent être soit faites indépendamment des autres propositions, soit basées sur l'historique de la négociation.

Les auteurs clament également que si les agents peuvent seulement accepter ou refuser les propositions, alors la négociation peut être gourmande en temps et inefficace. Pour améliorer l'efficacité du processus de négociation, le destinataire de l'offre doit être capable de fournir un feedback plus utile sur les propositions qu'il reçoit. Ce feedback peut prendre la forme d'une critique ou d'une contre-proposition. Grâce à de tels feedbacks, l'initiateur devrait être en position de générer une proposition qui est plus à même de conduire à un accord.

Nous sommes du même avis et notre modèle de négociation tient compte de ces remarques.

Le simple achat d'un article dans un magasin où l'acheteur paie le prix indiqué ne constitue pas, pour nous, une négociation, car aucun des protagonistes n'a la possibilité de faire des concessions afin d'aboutir à un accord commun. Cette réflexion nous amène à nous interroger au sujet des enchères automatiques. En effet, lors d'enchères électroniques, il n'y a pas toujours de concessions entre le vendeur et les acheteurs et seuls le vendeur et le gagnant sont satisfaits de la décision finale. Les autres acheteurs sont insatisfaits de cette décision car leur avis n'a pas été pris en compte. Nous détaillons différentes formes d'enchères automatiques dans la prochaine section de ce chapitre. Bien que la négociation par enchères soit la plus connue, d'autres formes de négociation existent, basées sur le protocole d'interaction du *Contract Net*. Nous les décrivons également dans ce chapitre. Bien qu'étant reconnu comme une négociation, le *Contract Net* n'implique pas de discussion ni de contre-proposition, ce qui constitue une négociation pauvre à nos yeux. Les mécanismes de vote sont souvent considérés comme

des négociations. Pour ma part, le vote, qu'il soit majoritaire ou minoritaire, ne constitue pas une négociation. En effet, même si le vote permet de choisir parmi différentes propositions, il ne s'agit là que d'un choix entre plusieurs solutions et non pas de négociation à partir d'une solution pour en trouver une qui satisfasse au mieux l'ensemble des participants. Comme les participants choisissent la solution qu'ils préfèrent, ils ne font pas de concessions pour aboutir à une solution commune. Si l'on prend le cas d'un vote majoritaire (la solution choisie est celle qui remporte le plus de voix), il est possible que la solution trouvée ne satisfasse pas la majorité des participants. C'est par exemple le cas si $(m(m+1)/2)$ participants doivent voter pour choisir entre m solutions (avec $m > 3$). La première solution recueille 1 voix, la seconde 2 voix et ainsi de suite jusqu'à la dernière qui remporte m voix. C'est donc la dernière solution qui l'emporte alors que $(m(m-1)/2)$ participants ne l'ont pas choisie.

Notion de protocole Chaque forme de négociation possède son propre protocole, qui définit le déroulement du processus de négociation, c'est-à-dire les actes de langage utilisés et leur séquencement. Dans ce qui suit, nous présentons le problème de la négociation défini par Kraus dans [Kraus, 2001]. Un ensemble d'agents $\{A_1, .., A_N\}$ a besoin d'aboutir à un accord sur un problème donné. Les agents peuvent intervenir dans la négociation seulement à certains moments dans l'ensemble $\mathcal{T} = \{0, 1, ...\}$ qui sont déterminés par avance et connus des agents. A chaque instant $t \in \mathcal{T}$ de la négociation, si la négociation ne s'est pas terminée auparavant, l'agent dont c'est le tour de proposer une offre à l'instant t va suggérer un accord possible (en respectant le problème de négociation spécifique) et chaque autre agent peut soit accepter l'offre, la rejeter ou quitter la négociation. Si une offre est acceptée par tous les agents, alors la négociation se termine et cette offre est concrétisée. Si au moins un agent quitte la négociation, alors la négociation se termine et le conflit n'est pas résolu. Si personne n'a quitté la négociation mais qu'au moins un agent a refusé l'offre, la négociation continue à la période $t+1$ et l'agent suivant formule une contre-proposition et les autres agents répondent à nouveau.

Les agents ne sont pas informés des réponses des autres agents à une proposition d'offre au cours d'une période de négociation. Ce protocole est ainsi appelé *protocole à réponse simultanée*.

Dans ce protocole, il n'y a pas de limite sur le nombre de périodes (card $\mathcal{T} = \infty$) et un agent peut faire la même proposition que celui qui le précédait. Ce protocole ne nous satisfait pas car tous les agents doivent être d'accord avec l'offre faite pour que la négociation soit un succès, d'une part, ce qui n'est pas forcément toujours le cas et d'autre part, les offres sont proposées successivement sans savoir si elles seront plus proches des offres acceptables pour les autres agents. Il nous semble donc nécessaire qu'un agent tienne un rôle de centralisateur qui collecte les propositions de modification de l'offre et en fasse une synthèse pour proposer une nouvelle offre ayant de bonnes chances d'être acceptée.

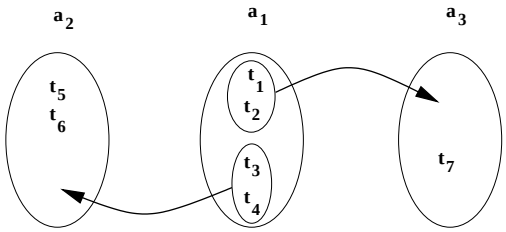


FIG. 2.1 – *Allocation groupée de tâches. Cette figure décrit une situation dans laquelle l'agent a1 propose à l'agent a2 l'ensemble de tâches formé de t3 et t4. Ce dernier accepte de les exécuter. a3 répond positivement à l'agent a1 sur l'ensemble des tâches t1 et t2.*

Nous décrivons maintenant cinq approches proposées par Sandholm [2]. Dans la première approche de Sandholm [Sandholm, 1993], les décisions des agents pour l'allocation des tâches sont fondées sur le calcul des coûts marginaux. Dans la deuxième approche d'allocation de tâches [Sandholm and Lesser, 1995], Sandholm et Lesser se sont intéressés au problème de l'allocation de plusieurs tâches en même temps pour un seul agent (cf. Figure 2.1). D'après Sandholm et Lesser, leur approche réduit le temps d'exécution des tâches lorsque celles-ci sont dépendantes car les moyens nécessaires à l'exécution d'une tâche peuvent servir à exécuter les tâches qui en dépendent.

Néanmoins, cette extension présente certaines limites car elle ne répond pas à un des problèmes entraînés par cette recomposition des tâches. En effet, un manager doit-il favoriser les agents qui postulent pour tout un ensemble de tâches ou ceux qui postulent seulement pour une partie de cet ensemble ? Et pour quels types d'applications faut-il que les uns soient favorisés par rapport à d'autres ? Dans tous les cas, si le système multi-agent est compétitif, les propositions émises par les contractants risquent d'être moins importantes que celles émises dans le cas d'utilisation du *Contract Net Protocol* car les agents contractants auront tendance à réduire leurs propositions puisqu'ils se proposent pour accomplir plusieurs tâches à la fois. À ce propos, dans [Aknine et al., 2001], les auteurs pensent que cette extension reste fortement dépendante du domaine d'application.

Afin d'améliorer le temps de négociation, Sandholm et Lesser considèrent que les agents qui exécutent séparément leurs tâches peuvent parfois échanger certains contrats (Swap-contracts). La Figure 2.2 décrit cette situation. Les protocoles définis dans cette approche pour les agents individualistes et les agents coopératifs sont différents. Un agent coopératif peut s'engager à exécuter la tâche échangée sans compensation de la part du gestionnaire du contrat. Par contre, un agent individualiste impose une compensation pour chaque traitement supplémentaire effectué par rapport à la tâche échangée. Parfois les agents peuvent avoir recours à une compensation monétaire. Dans [Aknine et al., 2001], les auteurs pensent que ce genre de protocole n'est pas suffisant pour assurer la coordination des agents. En effet, dans certaines applications,

[2] Cette description est issue de [Aknine et al., 2001], avec l'accord de l'auteur

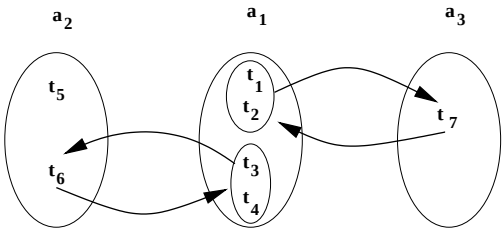


FIG. 2.2 – *Allocation de tâches par échanges. Cette figure décrit une situation dans laquelle l'agent a1 propose à l'agent a2 l'ensemble de tâches formé de t3 et t4. a2 accepte d'exécuter ces deux tâches à condition que a1 prenne en charge la tâche t6.*

pour que ce protocole soit utilisable, il doit être combiné avec d'autres protocoles car les agents peuvent ne pas vouloir échanger leurs tâches, par exemple, pour des raisons de confidentialité.

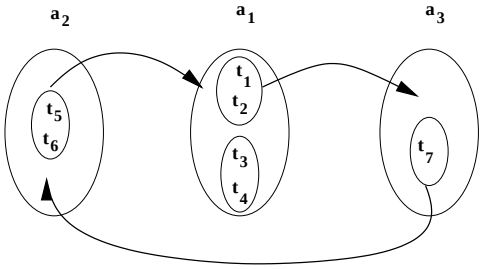


FIG. 2.3 – *Allocation de tâches par circulation des annonces. Cette figure indique que l'agent a1 a reçu une proposition (t5 et t6) de l'agent a2. a1 n'est pas intéressé par ces tâches, il diffuse l'information à l'agent a3. a3 peut également proposer une de ses tâches à l'agent a2.*

La quatrième approche présentée par Andersson et Sandholm [Andersson and Sandholm, 1998] montre qu'il est possible de faire circuler des contrats entre agents par des échanges de messages. Un contrat peut être considéré comme valide si l'un des agents intéressés signe le contrat. La Figure 2.3 décrit cette situation. Cependant, ce protocole reste applicable uniquement dans un système multi-agent coopératif dans lequel les agents peuvent contribuer à améliorer le fonctionnement du manager d'un contrat. Par contre, les agents d'un système multi-agent compétitifs se garderont de diffuser l'information annoncée afin d'augmenter leurs chances d'obtention des contrats. Pour un système multi-agent coopératif, l'avantage de cette approche réside dans le fait que les annonces sont largement diffusées entre les agents, i.e., chaque agent les diffuse auprès de ses propres connaissances. Ce qui peut éventuellement faciliter la candidature de plusieurs postulants à une annonce. Néanmoins, ses limites restent tout aussi importantes : (1) le système multi-agent risque d'être facilement submergé par des messages dupliqués par l'envoi simultané de propositions par différents agents. En effet, avec n agents dans le système et k

tâches à réaliser, $(kn)^2$ propositions peuvent circuler à la fois entre les agents ; (2) par conséquent, la charge calculatoire de chacun des agents se trouve ainsi inutilement augmentée par des traitements de diffusion et de réception de messages ; (3) la durée de négociation devient ainsi plus élevée.

La cinquième approche présentée par Sandholm et Lesser [Sandholm and Lesser, 1996] considère qu'un agent peut se désengager d'un contrat établi s'il paye, en contre partie, une certaine pénalité exigée par le manager du contrat. Néanmoins, le recours aux pénalités n'est pas admissible pour des systèmes multi-agents coopératifs, i.e., des systèmes dans lesquels les agents participent volontairement à la réalisation des tâches des autres agents sans demander des compensations. L'application du principe de pénalisation dans ces systèmes, empêcherait les agents de se proposer volontairement à d'autres tâches, ce qui changerait la nature même coopérative du système multi-agent. D'autre part, les pénalités ne sont pas toujours faciles à calculer, un consensus préalable sur la valeur de cette pénalité est de plus indispensable pour les agents. Ces problèmes n'ont pas été étudiés dans la définition de ce protocole.

2.2 Les différents types de négociation automatique

La négociation fait partie de notre quotidien : sans le savoir nous négocions lors d'achats dans les brocantes, ou lorsque nous prenons des rendez-vous (réunions, garagiste, médecin, etc.). Dès que deux ou plusieurs personnes se mettent à discuter pour prendre une décision nous entrons dans une phase de négociation. Nous ne décrivons ici que les négociations qui peuvent être informatisées. Nous ne traitons pas dans cette thèse des aspects sociologiques, culturels et psychologiques qui entrent en compte dans la négociation entre agents humains.

Définition 2 *Une négociation met en jeu des ressources, qui seront rassemblées afin d'être négociées dans un contrat et un ensemble de personnes qui participent à cette négociation. Il y a toujours un ou plusieurs manageurs (vendeur ou autre) et un ou plusieurs contractants (acheteurs ou autre).*

La négociation peut prendre diverses formes qui peuvent être regroupées en plusieurs familles. La première famille est celle des systèmes de vote qui sont utilisés pour choisir une solution parmi plusieurs. On y trouve notamment le vote à la pluralité et les méthodes de Borda, de Hare et de Condorcet. La seconde famille que nous présentons dans cette thèse est celle des enchères. Les enchères permettent aux personnes de vendre leur bien au meilleur prix possible mais également aux distributeurs d'attribuer des marchés aux fournisseurs proposant les prix les plus bas (enchères inversées). Les quatre grands types d'enchères sont les enchères anglaises, les enchères hollandaises,

les offres scellées au meilleur prix et les offres scellées au second meilleur prix. Nous présentons ensuite la famille des négociations basées sur le *Contract Net*, parmi lesquelles les négociations multi-étapes et les négociations combinées puis pour finir la famille des négociations à base d'argumentation.

2.2.1 Les systèmes de vote

Les systèmes de vote [3] sont utilisés pour élire une alternative parmi les différentes alternatives possibles. Les plus simples concernent un choix entre une alternative et le statu quo. Cela revient à proposer l'alternative et à recueillir les votes *pour* et les votes *contre* cette alternative. Les systèmes plus complexes impliquent un nombre d'alternatives supérieur à deux, les votants devant alors choisir l'alternative qu'ils préfèrent. Le terme de *choix social* est utilisé afin de représenter l'alternative qui satisfait au mieux la population. Dans ce qui suit, nous considérons le problème suivant : soit un ensemble A de n alternatives dénotées a, b, c, etc. et un ensemble P de personnes (de votants) dénotées p_1, p_2, p_3, etc. Chaque personne p de l'ensemble P classe obligatoirement toutes les alternatives sous la forme d'une liste selon ses préférences. Nous supposons qu'il n'y a pas d'égalité au sein de ces listes. Nous représentons ces listes de manière verticale depuis les alternatives les plus préférées en haut de la liste jusqu'aux moins préférées en bas de la liste. Nous présentons dans cette sous-section différentes procédures de choix social qui fournissent la ou les alternatives les plus préférées des personnes ainsi que des propriétés souhaitables pour de telles procédures.

Différentes procédures de choix social

Définition 3 *Une* procédure de choix social *est une fonction qui prend en paramètre une séquence de listes (sans égalités) d'un ensemble A (l'ensemble des alternatives) et qui produit soit un élément de A, soit un sous-ensemble de A (en cas d'égalité).*

Cette définition montre qu'une procédure de choix social ne fournit pas nécessairement un seul choix mais peut en fournir plusieurs en cas d'égalité. S'il est nécessaire de n'avoir qu'un seul choix social, on peut par exemple utiliser une autre procédure de choix social pour départager les alternatives qui sont à égalité pour une procédure donnée ou même un simple tirage aléatoire.

Nous décrivons ici différentes procédures de choix social, de la pluralité à la dictature en passant par différentes méthodes de notation des alternatives. Ces procédures ont été conçues pour des élections politiques.

[3] Cette sous-section est fortement inspirée du livre [Taylor, 1995]

Nous illustrons chacune de ces procédures avec l'exemple suivant. On considère un problème avec $n = 5$ alternatives a, b, c, d et e, et 7 personnes dont les listes de préférences sont les suivantes :

p_1	p_2	p_3	p_4	p_5	p_6	p_7
a	a	a	c	c	b	e
b	d	d	b	d	c	c
c	b	b	d	b	d	d
d	e	e	e	a	a	b
e	c	c	a	e	e	a

Le scrutin uninominal majoritaire à un tour (pluralité) Cette méthode simple consiste à nommer *choix social* l'alternative étant classée première le plus grand nombre de fois parmi les listes de préférences des votants (majorité relative). C'est le système de vote classique par excellence qui est utilisé dans les élections politiques pour élire le président du Mexique, du Kenya ou encore de l'Islande (en France, on utilise deux tours). En cas d'égalité, le résultat de cette méthode est un sous-ensemble d'alternatives. Un inconvénient de cette méthode est qu'une alternative peut être choisie alors qu'elle n'est pas désirée par la majorité absolue des votants.

Avec notre exemple, a remporte 3 voix, b en remporte 1, c en remporte 2, d en remporte 0 et e remporte 1 voix. L'alternative a avec 3 voix est donc le choix social selon cette procédure, alors que 4 personnes sur 7 n'ont pas voté pour elle.

La méthode de Borda Cette procédure a été proposée pour la première fois par Jean-Charles Borda en 1781. L'idée générale est d'attribuer des points à chaque alternative. Le principe est le suivant : pour chaque liste de préférences, l'alternative classée première remporte $n-1$ points, la seconde $n-2$ points et ainsi de suite jusqu'à la dernière qui obtient 0 point. Pour chaque alternative, on calcule le nombre total de points qu'elle a recueilli. L'alternative ayant le meilleur score est déclarée choix social. Comme toutes les alternatives sont classées par les votants, elles ont toutes une note, on est donc sûr de trouver le choix social. En cas d'égalité, cette procédure renvoie un sous-ensemble d'alternatives. Cette méthode a l'inconvénient d'encourager les votes tactiques ou raisonnés. En effet, les votants peuvent être incités à classer les alternatives non pas selon leurs réelles préférences mais de façon à favoriser l'alternative qu'ils préfèrent. Ainsi, classer une alternative (autre que sa préférée) ayant des chances de remporter l'élection à la fin de sa liste, après des alternatives ayant peu de chances d'être élues, permet de favoriser son alternative préférée et de léser l'autre alternative. Il semble que Borda lui-même en était conscient, prétextant que sa méthode était conçue pour des hommes honnêtes.

Avec notre exemple, a remporte $4+4+4+0+1+1+0 = 14$ voix, b remporte $3+2+2+3+2+4+1 = 17$ voix, c remporte $2+0+0+4+4+3+3 = 16$ voix, d

remporte $1+3+3+2+3+2+2=16$ voix et e remporte $0+1+1+1+0+0+4=7$ voix. L'alternative b est donc le choix social selon la méthode de Borda.

La méthode de Hare Cette procédure a été proposée par Thomas Hare en 1861. Le principe général est de déterminer le choix social en éliminant successivement les alternatives les moins désirées. Le principe est le suivant : si une alternative est classée première sur au moins la moitié des listes de préférences, alors c'est le choix social et la procédure est terminée. Si aucune alternative n'est classée première sur au moins la moitié des listes, alors on sélectionne l'alternative classée première sur le moins de listes et on l'enlève de toutes les listes. Si plusieurs alternatives sont à égalité, on les enlève toutes des listes. Les alternatives qui suivaient celle qui a été enlevée sur les listes gagnent alors une place. A ce stade, les listes de préférences sont plus courtes d'au moins une alternative. On recommence alors la procédure qui est de chercher si une alternative est classée première sur au moins la moitié des listes et d'effacer l'alternative la moins désirée sinon. La procédure s'arrête dès qu'une alternative apparaît en premier sur au moins la moitié des listes ou si toutes les alternatives restantes apparaissent en premier sur exactement le même nombre de listes. On est sûr de trouver un choix social (ou un sous-ensemble d'alternatives) car toutes les alternatives sont classées par les votants et on ne les enlève pas toutes puisqu'on s'arrête dès qu'une alternative est classée première sur au moins la moitié des listes ou que toutes les alternatives restantes sont classées premières sur exactement le même nombre de listes.

Avec notre exemple, il faut qu'une alternative soit classée première sur au moins 4 listes pour être le choix social. Aucune alternative ne répond à ce critère (l'alternative a choisie par le plus de personne ne remporte que 3 voix), il faut donc retirer l'alternative la moins désirée. L'alternative la moins classée première dans les listes de préférences est l'alternative d car elle est classée première sur aucune des listes. On enlève donc l'alternative d de toutes les listes. On obtient donc les listes suivantes :

p_1	p_2	p_3	p_4	p_5	p_6	p_7
a	a	a	c	c	b	e
b	b	b	b	b	c	c
c	e	e	e	a	a	b
e	c	c	a	e	e	a

Nous n'avons toujours pas d'alternative classée première sur au moins quatre listes, il nous faut donc à nouveau retirer l'alternative la moins désirée. Les alternatives b et e sont à égalité avec une seule première place. On enlève donc ces deux alternatives des listes de préférences. On obtient alors :

p_1	p_2	p_3	p_4	p_5	p_6	p_7
a	a	a	c	c	c	c
c	c	c	a	a	a	a

L'alternative c est classée première sur quatre listes, elle est donc le choix social par la méthode de Hare.

La méthode de Condorcet Cette procédure a été proposée par le marquis de Condorcet au XVIIIe siècle. Le principe est le suivant : pour chaque paire d'alternatives, on détermine le nombre d'électeurs ayant voté pour l'une ou l'autre en vérifiant, sur chaque liste de préférences, comment l'une était classée par rapport à l'autre. Si une alternative gagne toutes les comparaisons avec les autres (ie. pour chaque autre alternative prise séparément, elle est mieux classée sur au moins la moitié des listes), alors elle remporte le vote. Il arrive qu'aucune alternative ne soit élue suite au décompte des votes. Condorcet avait remarqué une contradiction interne dans cette méthode appelée le *paradoxe de Condorcet* : dans une élection a peut être préférée à b, elle-même préférée à c, elle-même préférée à a. Il faut alors prévoir une méthode pour résoudre le conflit, par exemple en utilisant une autre méthode de vote.

Avec notre exemple, il faut comparer deux à deux les 5 alternatives, celle qui gagne est celle qui est mieux classée sur au moins 4 listes par rapport aux autres alternatives.

	a	b	c	d	e
a	-	3	3	3	5
b	4	-	4	3	6
c	4	3	-	5	4
d	4	4	2	-	6
e	2	1	3	1	-

Le tableau représente le résultat ligne par ligne des comparaisons d'une alternative avec les alternatives situées en colonne. On constate qu'il n'existe pas de choix social pour cet exemple. En effet, aucune alternative ne gagne toutes ses comparaisons (aucune n'a que des nombres ≥ 4 sur toute sa ligne). Cela représente le *paradoxe de Condorcet* : b est préféré à c, d est préféré à b et c est préféré à d, on a donc $c < b < d < c$. Il est donc impossible de déterminer l'alternative préférée. En revanche, si l'on enlève l'alternative c de l'exemple, on obtient un choix social qui est d : d est mieux classée que a sur 4 listes, d est mieux classée que b sur 4 listes et d est mieux classée que e sur 6 listes.

La méthode de vote séquentiel par paires dans un ordre fixé Cette méthode reprend la notion de comparaison des alternatives deux à deux de la méthode de Condorcet, mais ces comparaisons sont effectuées dans un ordre préétabli à l'avance. Le perdant d'une comparaison est éliminé tout de suite tandis que le gagnant rencontre l'alternative suivante d'après l'ordre fixé. En cas d'égalité, les deux alternatives sont comparées à l'alternative suivante. Il se peut que plusieurs alternatives soient à égalité à la fin des comparaisons, le résultat de la méthode est alors un sous-ensemble d'alternatives. Un inconvénient de cette méthode est que le résultat dépend de l'ordre fixé pour les comparaisons.

Avec notre exemple, si l'on prend l'ordre a, b, c, d et e, on commence par comparer a et b : a perd avec 3 victoires contre 4, a est donc éliminée. On poursuit avec la comparaison b contre c. b est mieux classée que c sur 4 listes, c est donc éliminée et on considère la comparaison b contre d. Cette fois -ci, b perd la comparaison et est donc éliminée. La dernière comparaison a lieu entre d et e. d remporte la comparaison et est donc déclarée choix social. En revanche, si l'on prend l'ordre e, d, c, b et a, c'est l'alternative b qui est le choix social.

La dictature Cette méthode est présentée afin de montrer qu'une procédure de choix social n'est pas forcément démocratique. Cette méthode consiste à désigner une personne parmi l'ensemble P et de l'appeler dictateur. On ignore simplement les listes de toutes les autres personnes. Le choix social est alors l'alternative classée première sur la liste du dictateur. Comme les personnes classent toutes les alternatives sans égalité, on est sûr d'obtenir un seul choix social. L'inconvénient de cette méthode provient du fait que la liste d'une seule personne est prise en compte dans la recherche du choix social. Cette méthode ne fournit pas un choix social démocratique alors que les autres méthodes présentées ont ce but.

Avec notre exemple, si l'on choisit le votant p_7 comme dictateur, le choix social est alors e.

Nous avons présenté plusieurs méthodes de vote qui donnent un choix social différent sur un même exemple. Ces méthodes présentent chacune divers inconvénients, comme la possibilité de ne pas trouver le choix social par la méthode de Condorcet, ou l'incitation pour les votants de ne pas classer les alternatives selon leurs préférences réelles pour la méthode de Borda. Borda et Condorcet clamaient que leurs méthodes étaient plus équitables, car elles prennent en compte l'ensemble des préférences des votants et pas seulement leur premier choix. Leur but était de trouver l'alternative réellement désirée par l'ensemble des votants. Nous allons maintenant définir quelques propriétés souhaitables pour une procédure de choix social.

Propriétés souhaitables pour une procédure de choix social

Les différentes procédures présentées ci-dessus peuvent donner des résultats différents pour les mêmes listes de préférences des personnes. Nous présentons ici différentes propriétés qui devraient être respectées par toute procédure de choix social par critère de rationalité.

La condition Pareto Une procédure de choix social satisfait la condition Pareto si pour tout couple d'alternatives x et y :

Si tout le monde préfère x à y alors y n'est pas un choix social.

Le critère du gagnant de Condorcet Une procédure de choix social satisfait le critère du gagnant de Condorcet pourvu que si le gagnant de Condorcet existe, alors lui seul est le choix social.

Monotonie Une procédure de choix social est dite monotone si pour toute alternative x :

Si x est le choix social et que quelqu'un change sa liste de préférences en déplaçant x d'une position vers le haut, alors x doit toujours être le choix social.

Indépendance des alternatives non pertinentes Une procédure de choix social satisfait le critère d'indépendance des alternatives non pertinentes (IIA) si pour tout couple d'alternatives x et y :

Si l'ensemble des choix sociaux contient l'alternative x mais pas l'alternative y et que un ou plusieurs votants changent leur liste de préférences sans modifier la préférence relative de x et de y, alors l'ensemble des choix sociaux ne doit pas changer de façon à inclure y.

Ces propriétés ont été définies afin de garantir la rationalité des procédures de choix social. Le tableau suivant reprend les différentes méthodes de choix social présentées et indique si elles respectent les propriétés énoncées.

	Pareto	Condorcet	Monotonie	IIA
Pluralité	oui	non	oui	non
Borda	oui	non	oui	non
Hare	oui	non	non	non
Paires	non	oui	oui	non
Dictateur	oui	non	oui	oui

Les preuves de ces résultats peuvent être trouvées dans [Taylor, 1995]. On constate qu'aucune méthode ne satisfait toutes les propriétés.

Le théorème d'impossibilité d'Arrow, également appelé « paradoxe d'Arrow », est une confirmation mathématique dans certaines conditions précises du paradoxe évoqué par Condorcet selon lequel il n'existerait pas de fonction de choix social indiscutable, permettant d'agréger des préférences individuelles en préférences sociales. Pour Condorcet, il n'existait pas de système *simple* assurant cette cohérence. Arrow démontre, sous réserve d'acceptation de ses hypothèses, qu'il n'existerait pas de système *du tout* assurant la cohérence, hormis celui - non démocratique - où un dictateur seul imposerait ses choix à tout le reste de la population.

Le théorème d'impossibilité d'Arrow C'est en 1950 que Kenneth Arrow [4] a publié dans le *Journal of Political Economy* un résultat connu maintenant sous le terme de théorème d'impossibilité d'Arrow.

[4] Prix Nobel d'économie en 1972

Définition 4 *Une* fonction de bien-être social *est une fonction qui*

1. *accepte en entrée une séquence de listes de préférences individuelles d'un ensemble A (l'ensemble des alternatives) et*
2. *produit en sortie un listing (avec peut-être des égalités) de l'ensemble A. Cette liste est appelée la* liste de préférences sociales.

Théorème 1 (Arrow, 1950) *Si A contient au moins trois alternatives et l'ensemble P des personnes est fini, alors la seule fonction de bien-être social pour A et P satisfaisant la condition de Pareto, l'indépendance des alternatives non pertinentes et la monotonie est une dictature.*

En d'autres termes, ce théorème indique qu'il est impossible de trouver une fonction de bien-être social sur A qui satisfasse la condition de Pareto, l'indépendance des alternatives non pertinentes et la monotonie et qui ne soit pas une dictature.

Nous avons présenté dans cette sous-section différents systèmes de vote ainsi que différentes propriétés souhaitables pour un système de choix social par critère de rationalité. Le théorème d'Arrow montre l'impossibilité de trouver un système de vote qui satisfasse ces propriétés. Les systèmes de vote sont des systèmes de négociation à un seul tour de parole (les participants envoient leur liste de préférences et l'initiateur réalise la procédure choisie) qui permettent de choisir parmi toutes les alternatives possibles. Il faut donc que chacun évalue l'ensemble des alternatives et les classe par ordre de préférence avant que le vote n'ait lieu. Cela est alors très coûteux en temps et en espace. Le système de négociation que nous proposons n'a quant à lui pas besoin de générer toutes les alternatives possibles et ne demande pas aux participants de toutes les évaluer, mais les propositions sont faites au fur et à mesure, ce qui permet de ne pas explorer l'ensemble des solutions ni de toutes les évaluer.

Les systèmes d'enchères constituent un autre système apparenté à la négociation, très utilisé à l'heure actuelle. La section suivante en fait une brève description.

2.2.2 Les enchères

Les enchères se sont démocratisées sur Internet grâce aux sites comme *eBay*, *Yahoo enchères*, *onSale*...C'est la forme de négociation la plus connue, bien que peu de systèmes d'enchères offrent de réelles possibilités de négociation. Elles sont utilisées pour la vente d'objets d'art, d'antiquités, de produits périssables, mais également pour l'immobilier et les voitures, et elles sont même utilisées dans certains jeux comme le bridge. Parmi les différentes enchères, nous décrivons les enchères ascendantes, descendantes, offres scellées au meilleur prix et offres scellées au second meilleur prix, qui sont parmi

les plus répandues sur Internet. Elles permettent à des personnes de vendre leurs biens au meilleur prix possible. Chacune de ces enchères implique un vendeur et une assemblée d'acheteurs, mais le déroulement de l'enchère diffère selon le type utilisé. Le vendeur fixe son prix de départ et son prix de réserve, c'est-à-dire le plus petit prix acceptable pour vendre le bien. Nous décrivons ici le déroulement de ces quatre types d'enchères.

type d'enchères	**description**
ascendantes	enchères ascendantes, publiques
descendantes	enchères descendantes, publiques
offres scellées au meilleur prix	enchères en un seul tour de parole, privées, le vainqueur est celui qui propose le meilleur prix
offres scellées au second meilleur prix (Vickrey)	enchères en un seul tour de parole, privées, le vainqueur est celui qui propose le meilleur prix mais ne paie que le second meilleur prix.

FIG. 2.4 – *Descriptions des différentes enchères les plus couramment utilisées.*

Les enchères ascendantes

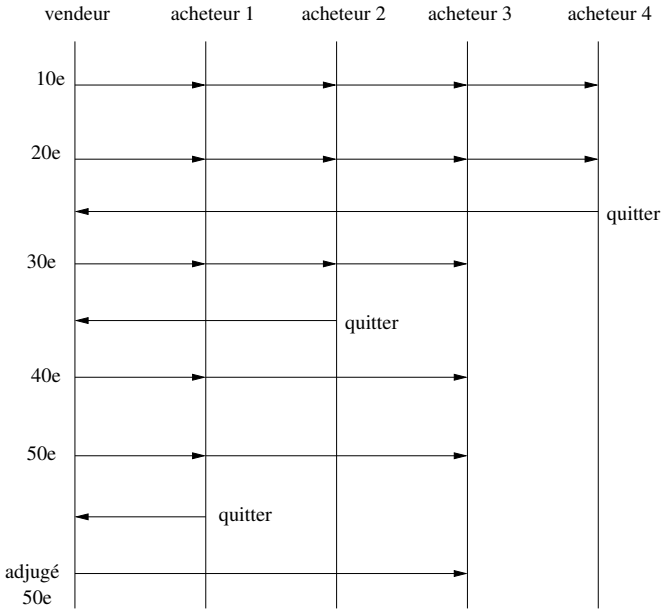


FIG. 2.5 – *Exemple d'enchères ascendantes. Quatre acheteurs sont en compétition pour obtenir l'article. Le vendeur augmente continuellement son prix jusqu'à ce qu'il ne reste qu'un acheteur. Ici, l'acheteur 3 remporte l'enchère pour 50 euros.*

Ces enchères sont dites ascendantes car le prix proposé pour le bien mis en vente augmente avec le temps. Elles sont aussi appelées enchères ouvertes, orales ou anglaises. Ces enchères se déroulent sur plusieurs tours. Lors d'enchères ascendantes, le prix est successivement augmenté jusqu'à ce qu'il ne reste qu'un seul acheteur, qui gagne le bien au prix final. Dans le modèle le plus couramment utilisé, le prix est continuellement augmenté par le vendeur et les acheteurs quittent les enchères au fur et à mesure que le prix dépasse leur budget. Les autres acheteurs observent les départs et une fois qu'un acheteur a quitté l'enchère, il ne peut plus y revenir. La Figure 2.5 illustre ces enchères en prenant un vendeur et quatre acheteurs. Le vendeur annonce un prix de 10 euros, tous les acheteurs restent en lice. Il annonce alors un prix de 20 euros et l'acheteur 4 quitte l'enchère. Le vendeur continue à augmenter son prix et annonce 30 euros. Cette fois, l'acheteur 2 quitte l'enchère. Le vendeur annonce alors 40 euros, puis 50 euros. A ce prix, l'acheteur 1 quitte l'enchère et donc l'acheteur 3 remporte le bien mis en vente à 50 euros.

Ces enchères sont implémentées dans *AuctionBot* [Wurman et al., 1998], un serveur d'enchères en ligne et sont couramment utilisées pour la vente d'objets d'art ou d'antiquités.

Ces enchères se déroulent sur plusieurs tours, il y a donc discussion entre le vendeur et les acheteurs, ce qui est bien, à notre sens, de la négociation. Le protocole de négociation de ces enchères peut se formaliser ainsi : le vendeur propose son article et le prix qu'il veut en obtenir à un ensemble d'acheteurs. Ceux-ci acceptent ou refusent la proposition. S'ils refusent, ils quittent la négociation. S'ils acceptent, ils restent en course pour l'obtention de l'article. S'il ne reste qu'un seul acheteur, il gagne le contrat. Sinon, le vendeur propose à nouveau son article avec un prix plus élevé. Le processus se termine lorsqu'il ne reste plus qu'un acheteur. Comme ce sont des enchères, il n'est pas possible pour un acheteur de se rétracter : une fois qu'il a acquis l'objet, il ne peut pas le rendre.

Les enchères descendantes

Ces enchères sont dites descendantes car le prix proposé pour le bien mis en vente diminue avec le temps. Elles sont utilisées notamment aux Pays-Bas pour la vente des fleurs, c'est pourquoi les économistes les appellent aussi enchères hollandaises. Ces enchères fonctionnent de façon exactement opposée aux précédentes : le vendeur propose un très haut prix pour son bien et le diminue jusqu'à ce qu'un acheteur se manifeste pour acquérir le bien au prix alors mentionné. La Figure 2.6 illustre ces enchères en y faisant intervenir quatre acheteurs. Le vendeur annonce en premier lieu un prix de 60 euros. Aucun acheteur ne souhaite payer ce prix. Le vendeur baisse alors son prix de 10 euros, passant donc à 50 euros. Personne n'est intéressé à ce prix. Le vendeur continue donc de baisser son prix et lorsque celui-ci atteint 10 euros, l'acheteur 3 se manifeste pour acquérir le bien. Il remporte donc l'enchère pour 10 euros.

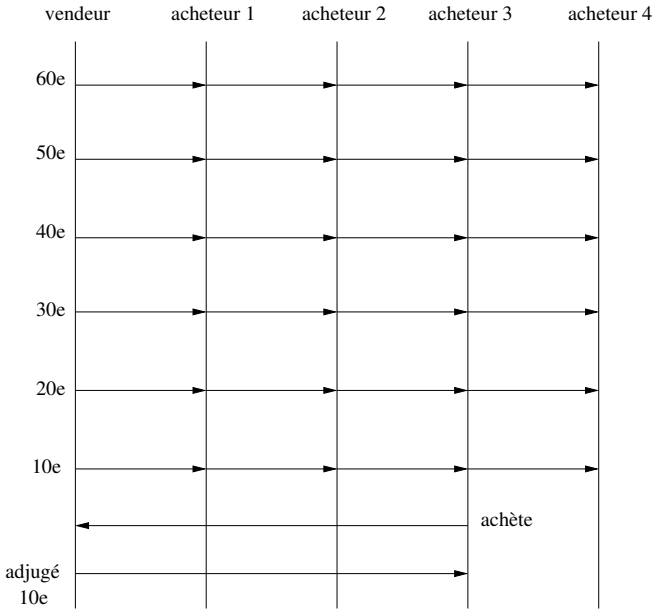


FIG. 2.6 – *Exemple d'enchères descendantes. Quatre acheteurs sont en compétition pour obtenir l'article. Le vendeur diminue continuellement son prix jusqu'à ce qu'un acheteur se manifeste pour acquérir le bien. Ici, l'acheteur 3 remporte l'enchère pour 10 euros.*

Ce sont ces enchères qui sont utilisées dans *Fishmarket* [Noriega, 1998], une place d'enchères électroniques pour la vente de poissons en Espagne développée par Noriega. Ces enchères sont surtout utilisées pour la vente de produits périssables. Les enchères au cadran sont des enchères descendantes automatisées où le prix est affiché sur un cadran, d'où leur nom. Elles sont fréquemment rencontrés sur des marchés au cadran pour la vente de bétail, de poisson ou autres denrées périssables.

Ces enchères sont aussi une négociation, puisqu'elles se déroulent sur plusieurs tours. Le protocole se définit comme suit : le vendeur propose son article à un prix très élevé à un ensemble d'acheteurs. Si un acheteur accepte la proposition, il gagne le contrat. Si tous les acheteurs refusent, le vendeur leur propose à nouveau son article à un prix moins élevé. Le processus se termine lorsqu'un acheteur accepte la proposition ou lorsque le prix de réserve est atteint et que personne n'accepte la proposition.

Les offres scellées au meilleur prix

Lors de ces enchères, qui se déroulent sur un seul tour, chaque acheteur propose un prix unique, sans connaître le prix proposé par les autres (d'où le terme offres scellées). Le vainqueur est celui qui a proposé le prix le plus élevé et paie son prix, d'où leur nom. La Figure 2.7 représente ces enchères dans le cas de quatre acheteurs en compétition.

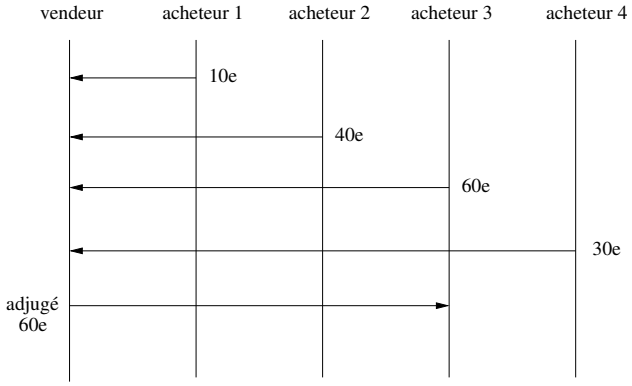


FIG. 2.7 – *Exemple d'enchères à offres scellées au meilleur prix. Quatre acheteurs sont en compétition pour obtenir l'article. Chaque acheteur propose secrètement au vendeur un prix pour le bien mis en vente. L'acheteur ayant proposé le prix le plus élevé remporte l'enchère et paie le prix qu'il a proposé. Ici, c'est donc l'acheteur 3 qui remporte l'enchère pour 60 euros.*

Chaque acheteur propose secrètement un prix au vendeur. Ainsi, l'acheteur 1 propose 10 euros, l'acheteur 2 propose 40 euros, l'acheteur 3 propose 60 euros et l'acheteur 4 propose 30 euros. C'est donc l'acheteur 3 qui a proposé le prix le plus élevé (60 euros), il remporte donc l'enchère et paie le prix qu'il a proposé, soit 60 euros.

Comme il n'y a pas de seconde proposition possible, ces enchères ne sont pas une négociation à proprement parler. Le protocole de ces enchères est très simple : le vendeur propose un article à un ensemble d'acheteurs. Ceux-ci peuvent soit accepter la proposition, auquel cas ils proposent un prix, soit refuser la proposition. Le gagnant de l'enchère est celui qui a proposé le prix le plus élevé (supérieur au prix de réserve). L'une des différences avec les précédentes enchères est qu'il n'y a pas de contre-proposition possible.

Les offres scellées au second meilleur prix

Ces enchères sont également appelées enchères de Vickrey. Le déroulement de ces enchères est exactement le même que celui des enchères aux offres scellées au meilleur prix, mis à part que le vainqueur paie le second meilleur prix proposé. La Figure 2.8 représente ces enchères dans le cas de quatre acheteurs en compétition. Chaque acheteur propose secrètement un prix au vendeur. Ainsi, l'acheteur 1 propose 10 euros, l'acheteur 2 propose 40 euros, l'acheteur 3 propose 60 euros et l'acheteur 4 propose 30 euros. C'est donc l'acheteur 3 qui a proposé le prix le plus élevé (60 euros), il remporte donc l'enchère mais ne paie que le second prix le plus élevé, soit 40 euros.

Ces enchères sont implémentées dans Magma [Tsvetovatyy et al., 1997], un marché virtuel pour le commerce électronique utilisant des agents.

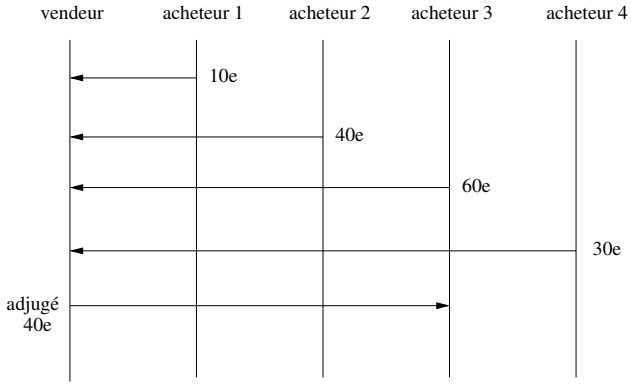


FIG. 2.8 – *Exemple d'enchères à offres scellées au second meilleur prix. Quatre acheteurs sont en compétition pour obtenir l'article. Chaque acheteur propose secrètement au vendeur un prix pour le bien mis en vente. L'acheteur ayant proposé le prix le plus élevé remporte l'enchère et paie le second prix le plus élevé qui a été proposé. Ici, c'est donc l'acheteur 3 qui remporte l'enchère pour 40 euros.*

Ici aussi, il n'y a pas de seconde proposition possible et donc pas de négociation à proprement parler. Le protocole est identique à celui des offres scellées au meilleur prix, la seule différence étant le prix à payer par le gagnant du contrat.

Les autres formes d'enchères

Il existe encore beaucoup d'autres formes d'enchères comme les enchères à la bougie, doubles, inversées, etc. Le rituel des enchères à la bougie est à l'identique depuis le XVème siècle : les acheteurs sont rassemblés dans une salle et l'article en vente est mis à prix par un commissaire priseur qui allume alors deux bougies. Les acheteurs potentiels portent alors leurs enchères. Lorsque celles-ci s'essoufflent, on allume le troisième feu. Si personne ne renchérit avant que la troisième bougie s'éteigne, l'adjudication est prononcée. Les enchères doubles se déroulent entre plusieurs vendeurs et plusieurs acheteurs. Les vendeurs publient une offre de vente à un prix donné tandis que les acheteurs publient des offres d'achat à un prix fixé. L'acheteur ayant fait la plus grosse offre remporte la vente avec le vendeur ayant proposé le prix le plus bas, et paie ce prix. Ce processus se déroule tant qu'il y a des vendeurs et des acheteurs dont les offres concordent. Ces enchères sont néanmoins moins utilisées de façon électronique. Les enchères inversées (ou dégressives) consistent à proposer un prix de plus en plus bas par les participants pour remporter le marché. Elles sont employées par les grands groupes de l'industrie agro-alimentaire pour faire baisser les prix de leurs fournisseurs, les poussant à vendre en dessous de leur prix de revient [Mathiot, 2004]. Plusieurs variations de ces enchères existent, dont la version dite *tournante*, où les participants sont obligés de baisser leur prix sous peine d'être éliminés et l'enchère dite *au ranking*, où les

participants n'ont comme information que le rang auquel leur dernière offre les classe. Cette utilisation des enchères par de grands groupes peut ne pas aboutir à un contrat avec le fournisseur ayant proposé le prix le plus bas, mais avec un fournisseur choisi à l'avance qui aura alors baissé son prix au maximum.

D'autres systèmes d'enchères électroniques définissent leur propre protocole d'enchère, c'est le cas de *Kasbah* [Chavez and Maes, 1996], une place de marché pour l'achat et la vente de biens. *Kasbah* utilise un mécanisme de classification d'annonces et des agents acheteurs ou vendeurs. Chaque agent acheteur (resp. vendeur) consulte l'ensemble des agents vendeurs (resp. acheteurs) potentiels avant de consulter à nouveau l'un d'entre eux. Chaque agent peut faire une proposition à un autre agent, ou lui demander le prix qu'il offre pour l'item, ou encore demander quel est l'item. Lors d'une proposition, l'agent décide alors d'accepter ou non l'offre, mais ne peut formuler de contre-proposition avant le prochain tour de parole. Le vendeur adopte une stratégie de réévaluation du prix demandé qui dépend du temps afin de réussir à vendre son bien.

Les enchères sont une forme très répandue de négociation, bien que nous ayons vu que certaines d'entre elles ne puissent pas vraiment être considérées comme telles. Pour ma part, je considère que les enchères à offres scellées ne constituent pas une négociation. En effet, une seule proposition de prix est émise par les acheteurs, il n'y a donc pas de discussion entre le vendeur et les acheteurs, ce qui est l'une des caractéristiques de la négociation. De plus, le but de la négociation est de satisfaire au mieux les participants, or, lors d'enchères, seuls le vendeur et le gagnant de l'enchère sont satisfaits. Aucun autre acheteur ne l'est. En ce qui concerne les enchères ascendantes et descendantes, le même reproche concernant la satisfaction des participants peut être fait. Mais je trouve ces enchères plus proches de la négociation que les précédentes car plusieurs tours de parole sont donnés aux acheteurs afin d'aboutir à un prix satisfaisant à la fois le vendeur et l'acheteur. Nous avons décrit le protocole de chacune de ces enchères, ce qui nous permettra de proposer un protocole unique dans le chapitre suivant. Les enchères ne sont pas les seules formes de négociation existantes, notamment pour le commerce électronique, comme le montre Kersten dans [Kersten et al., 2001]. La principale différence entre les enchères et les négociations est que les enchères n'impliquent qu'un seul attribut de négociation, qui est le prix, tandis que les négociations peuvent impliquer plusieurs attributs tels qu'une qualité de service, un délai d'expédition, etc. Nous présentons maintenant d'autres formes de négociation, plus complexes à mettre en œuvre.

2.2.3 Les autres formes de négociation

Moins connues du grand public, ces négociations sont tout aussi nombreuses et variées. Parmi ces négociations, on trouve le *take it or leave it offer*, les négociations multi-

attributs, les négociations multi-niveaux, les négociations combinées et la négociation à base d'argumentation.

Le *take it or leave it offer*

Cette forme de négociation est très primaire, puisqu'elle consiste à formuler une proposition qui est *à prendre ou à laisser* par le ou les participants. Cette négociation se déroule sur un seul tour, sans contre-proposition ni renégociation. C'est celle que l'on rencontre tous les jours pour acheter son pain, par exemple. Elle ressemble aux offres scellées au meilleur et au second meilleur prix, la différence vient du fait que dans les enchères, le prix est proposé par les acheteurs, tandis que dans le *take it or leave it offer* c'est le vendeur qui propose un prix ferme. Le protocole est donc très simple : le vendeur propose sa ressource (bien, service, etc.) à un prix ferme à un acheteur qui soit accepte, soit refuse. S'il accepte, l'acheteur paie le prix pour obtenir la ressource. Le *take it or leave it offer* n'est pas, pour nous, de la négociation.

Les négociations multi-attributs

Les négociations multi-attributs, comme leur nom l'indique, sont des négociations qui impliquent différents attributs devant être négociés. Elles sont directement opposées aux enchères qui n'impliquent qu'un seul attribut : un prix. Cette forme de négociation est cependant très répandue et à la base de nombreuses variantes de négociation. Un exemple de négociation multi-attributs est la négociation d'une voiture chez un concessionnaire. Le modèle, la motorisation, la couleur et de nombreuses options comme la climatisation, la direction assistée ou encore la présence d'airbags, en plus du prix, seront négociés.

Une fonction d'utilité est utilisée pour comparer les différentes offres. Cette fonction permet de pondérer l'importance de chaque attribut et chaque attribut est associé à une autre fonction d'utilité déterminant la satisfaction pour cet attribut. Si l'on reprend l'exemple de l'achat d'une voiture, on peut utiliser la même fonction d'utilité pour les options qui vaut 1 si l'option est présente, 0 sinon. De manière générale, la fonction d'utilité d'une offre o_j est de la forme :

$$U(o_j) = \sum_{i=1}^{n} \omega_i . u_i(o_j^i)$$

où la négociation comprend n attributs, ω_i le poids du ième attribut, u_i la fonction d'utilité du ième attribut et o_j^i la valeur du ième attribut dans l'offre o_j.

Le but est de maximiser la valeur de cette fonction d'utilité. Dans [Bichler et al., 1999], Bichler et ses collègues proposent d'utiliser les enchères pour

les négociations multi-attributs en utilisant la fonction d'utilité comme seul attribut d'évaluation de l'enchère. Dans [Jonker and Treur, 2001], Jonker et Treur proposent une architecture d'agent pour la négociation multi-attributs et décrivent le modèle de négociation qu'ils proposent et l'illustrent avec l'exemple de l'achat d'une voiture.

Les négociations multi-niveaux

C'est un type de négociation où le contrat se décompose en plusieurs "sous-contrats", dépendants les uns des autres [Meyer et al., 1988]. La négociation s'effectue séquentiellement pour chaque sous-contrat, la réussite globale est atteinte lorsque tous les sous-contrats ont été négociés avec succès. Lorsqu'un sous-contrat est en cours de négociation et qu'aucune solution n'est possible, on effectue un backtrack pour la négociation du sous-contrat précédent. Par exemple, pour la prise de rendez-vous, on peut décomposer le contrat en recherche d'un jour, puis d'une heure dans ce jour.

Dans [Conry et al., 1992], Conry et ses collègues utilisent la négociation multi-niveaux pour la satisfaction de contraintes distribuées et plus particulièrement pour le cas d'agents ayant des buts locaux qui nécessitent l'obtention de ressources communes. La négociation est alors utilisée pour que tous les agents puissent accomplir leurs buts locaux. Ils utilisent également la négociation multi-niveaux pour la restauration de canaux de communication dans un système de communications complexe.

Les négociations combinées

Les négociations combinées sont utilisées lorsqu'une personne a besoin d'un ensemble d'objets non disponibles auprès d'un unique vendeur. Il faut alors négocier chaque objet (ou sous-ensemble d'objets) séparément et avoir un mécanisme de liaison entre les négociations, car si l'ensemble des objets ne peut être acquis, aucun objet ne doit l'être. De plus, il ne faut obtenir chaque item qu'en un seul exemplaire. Les différentes négociations sont indépendantes les unes des autres, alors que l'ensemble des objets négociés sont typiquement interdépendants. Les négociations peuvent être de type différent pour chaque objet ou sous-ensemble d'objets. On peut donc rencontrer des enchères anglaises, des négociations de type *take it or leave it*, ou autres. Un exemple type de négociation combinée est la composition d'un voyage. Pour obtenir un voyage, il faut un moyen de transport (par exemple, un vol aller-retour vers la destination choisie) et les nuits d'hôtel pour la période du voyage. Le vol aller-retour se négocie avec une compagnie aérienne, tandis que les nuits d'hôtel se négocient avec une compagnie hôtelière. Cet exemple est celui retenu pour la *Trading Agent Competition (TAC)*, une compétition où différents agents sont chargés de composer des voyages pour 8 clients ayant des préférences sur les dates de départ et d'arrivée, ainsi que sur le type d'hôtel où ils vont résider. Cette compétition, qui a lieu tous les ans depuis trois ans, utilise

AuctionBot comme support logiciel pour les différentes enchères. Nous avons participé cette année à la compétition, mais malheureusement, une panne technique ne nous a pas permis de participer à la finale. Benyoucef [Benyoucef et al., 2001] étudie ces négociations et propose un système de support aux négociations combinées (CNSS), appelé *CONSENSUS*, qui permet à un utilisateur de s'engager dans différentes négociations en même temps. Ce CNSS est basé sur un système de workflow pour gérer le séquencement des négociations et leurs dépendances. Il utilise également un système de règles pour définir les stratégies de négociation. Enfin chaque négociation est confiée à un agent. Sandholm [Sandholm, 2002] et Aknine [Aknine, 2002] étudient également cette forme de négociation et plus particulièrement les stratégies à utiliser pour mener à bien cette négociation avec les meilleurs résultats possibles. Le problème de la recherche du gagnant étant NP-Complet, Sandholm a élaboré dans [Sandholm, 2002] un algorithme optimal pour la détermination du gagnant dans des enchères combinatoires se basant sur l'algorithme IDA* de Korf [Korf, 1985]. Aknine a ensuite amélioré cet algorithme dans [Aknine, 2004]. Ces algorithmes construisent des graphes à partir des offres reçues par l'acheteur.

Prenons l'exemple suivant : un acheteur désire obtenir sept items $\{1.2.3.4.5.6.7\}$. La requête formulée pour les vendeurs est sous la forme conjonctive. Supposons qu'en retour, l'acheteur reçoit les offres suivantes : $\{1, \$2\}$, $\{2, \$1\}$, $\{4, \$3\}$, $\{5, \$2\}$, $\{6, \$2\}$, $\{1.2, \$4\}$, $\{1.5, \$3\}$, $\{2.5, \$3\}$, $\{2.4, \$3\}$, $\{3.7\$4\}$, $\{4.5, \$3\}$, $\{1.4.5, \$5\}$, $\{2.3.4, \$6\}$, $\{2.4.5, \$5\}$, $\{2.4.5.7, \$7\}$, $\{1.3.4.5, \$7\}$, $\{1.2.4.7, \$6\}$, $\{1.2.3.4.5, \$9\}$.

Par exemple, $\{1.2.3.4.5, \$9\}$ signifie que le prix des cinq items ensemble $< 1.2.3.4.5 >$ est de $\$9$.

L'algorithme proposé par Aknine dans [Aknine, 2004] consiste dans un premier temps à calculer un score associé à chaque item g en utilisant la fonction f définie par $f(g) = \sum_{i=1}^{n} 1/T_i$, telle que n est le nombre d'offres contenant l'item g et T_i est la longueur de l'offre i contenant g.

Cette fonction permet de répartir les différents items et les offres qui leurs correspondent en plusieurs partitions.

Dans la seconde étape de l'algorithme, on commence par la création de différentes partitions. Dans la première partition, on isole les offres qui possèdent l'item de plus faible score, et ainsi de suite. L'étape suivante consiste à construire deux graphes : le premier est pour les items partagés par les offres et le second pour les items non-partagés. Une fois ces graphes construits, la recherche de la solution optimale peut commencer. Dans la troisième étape de résolution, on construit le graphe des solutions optimales. La Figure 2.9 représente l'arbre des solutions optimales pour l'exemple.

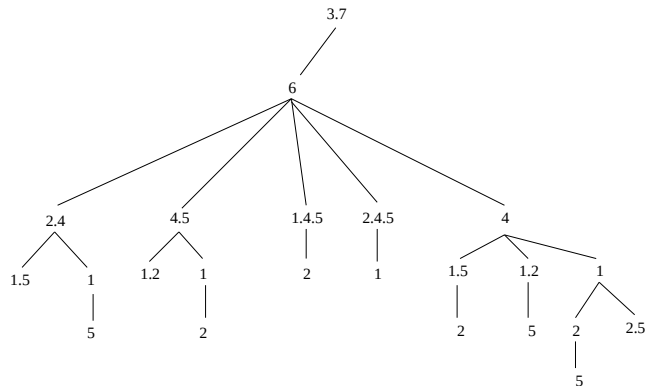


FIG. 2.9 – *Graphe des solutions optimales obtenu par l'algorithme d'Aknine (Figure issue de [Aknine, 2004])*

2.2.4 La négociation à base d'argumentation

La négociation à base d'argumentation est utilisée chez les agents logiques qui possèdent une base de connaissances avec des prédicats et des règles d'inférence. L'argumentation a alors pour but de modifier les croyances des autres agents afin qu'ils adoptent le même point de vue, les mêmes croyances et intentions que l'agent argumentant. Jennings [Parsons and Jennings, 1996] a présenté un rapport préliminaire sur cette forme de négociation en 1996, dans lequel il utilise des agents BDI (Belief, Desire, Intention) ainsi que des arguments et des règles d'inférence utilisant la notation Prolog et une extension des arguments pour indiquer le soutien ou le doute dans les propositions et pour les prouver. Amgoud et Prade proposent une approche possibiliste de la négociation par argumentation dans [Amgoud and Prade, 2004b] et un framework logique qui comprend de nouveaux types d'arguments dans [Amgoud and Prade, 2004a]. Dans [Carbogim and Robertson, 1999], Carbogim et Robertson proposent un framework général d'argumentation. Dans ce qui suit, nous présentons les arguments utilisés dans un système d'argumentation nommé *ANA* (l'Agent de Négociation Automatisé) et développé par Kraus, Sycara et Evenchik [Kraus et al., 1998] [5]. Les agents *ANA* sont des agents egocentrés et utilisent une méthode de négociation pour essayer de convaincre les autres d'accepter leurs propositions en cas de refus. Dans ce but, les agents doivent être capables de représenter leurs propres croyances, désirs et buts, de raisonner sur les croyances, désirs et buts des autres agents et d'essayer d'influencer les croyances et les intentions des autres agents du système.

Le modèle d'agent dans le système est un modèle BDI et la décision pour choisir le bon argument dépend des propres buts, des rapports entre ces buts , des croyances de l'agent, et de ce que l'agent croit concernant l'autre agent.

[5] Cet exemple est issu du cours en ligne sur les agents intelligents

`http://turing.cs.pub.ro/auf2/html/chapters/chapter5/chapter_5_5_2.html`

Dans la négociation, les agents peuvent utiliser différents types d'arguments. Chaque type d'argument est défini par des pré-conditions pour son utilisation. Si ces conditions sont satisfaites, alors l'agent peut utiliser l'argument. Évidemment, parmi les arguments possibles dans une certaine situation, l'agent doit choisir le bon argument ; par conséquent l'agent a besoin d'une stratégie pour décider quel argument utiliser.

Les types d'arguments sont les suivants :

Appels à une promesse passée le négociateur A rappelle à B une promesse passée concernant l'objet de négociation, autrement dit l'agent B a promis dans une négociation antérieure à l'agent A d'offrir ou effectuer un objet de négociation. *Pré-conditions* : l'agent A doit vérifier si une promesse concernant un objet de négociation a été reçue dans le passé à l'occasion d' une négociation conclue avec succès.

Promesse d'une récompense future le négociateur A promet de faire l'objet de négociation pour un autre agent B à un moment futur. *Pré-conditions* : l'agent A doit trouver un désir de l'agent B pour un moment futur, si possible un désir qui peut être satisfait par une action (service) que A peut effectuer mais que B ne peut pas effectuer.

Appels au propre intérêt l'agent A croit que la conclusion d'un accord sur l'objet de négociation est dans l'intérêt de B et essaye de convaincre B de cela. *Pré-conditions* : l'agent A doit trouver (ou déduire) un des désirs de B qui sera satisfait si l'agent B a l'objet de négociation ou A doit trouver un autre objet de négociation ON' qui a été offert auparavant sur le marché et démontrer que l'objet de négociation proposé est plus intéressant que ON'.

Appel à une pratique fréquente l'agent A croit que B a refusé la proposition parce que B croit que la proposition contredit un de ses buts. Dans ce cas, l'agent A donne à B l'exemple d'une pratique fréquente qui démontre que l'acceptation de la proposition ne contredit pas le but de B. *Pré-condition* : l'agent A doit trouver un autre agent ayant le même but que B, qui a été déjà d'accord avec une proposition semblable et qui a vu que l'accord n'a pas nui à ses buts.

Menace le négociateur menace de refuser à faire/offrir quelque chose à B ou il menace de faire quelque chose qui contredit les désirs de B. *Pré-conditions* : l'agent A doit trouver un des désirs de B directement satisfiable par un objet de négociation que l'agent A peut offrir ou A doit trouver une action qui est contradictoire avec ce qu'il croit être un des désirs de B.

Le système *PERSUADER* [Sycara, 1990] est un autre système bien connu qui utilise la négociation par argumentation. Les agents du système sont conçus pour résoudre les problèmes existant entre une compagnie et l'organisation syndicale de ses employés mais le modèle a été aussi appliqué dans le domaine de l'ingénierie concurrente.

En conlusion, nous avons présenté dans cette section la définition de la négociation que nous considérons dans cette thèse ainsi qu'une taxonomie des négociations. Cette taxonomie n'est bien sûr pas exhaustive, mais présente les négociations automatiques les plus répandues actuellement.

négociation par argumentation
négociations multi-niveaux, négociations combinées
négociations multi-attributs
Contract Net Protocol
enchères
systèmes de vote
take it or leave it offer

FIG. 2.10 – *Hiérarchie des différents types de négociation. La négociation sous sa forme la plus basique est en bas, la plus évoluée en haut.*

La Figure 2.10 représente une hiérarchie parmi les différentes formes de négociation. La forme la plus primaire de négociation est le *take it or leave it offer*, qui ne consiste pas pour nous en de la négociation. Puis un niveau supérieur de négociation est représenté par les systèmes de vote. Viennent ensuite les enchères, le *Contract Net Protocol*qui se déroule sur un seul tour de parole, les négociations multi-attributs, les négociations multi-niveaux et combinées, et enfin la négociation par argumentation. Les négociations multi-attributs sont situées entre le *Contract Net Protocol*et les négociations multi-niveaux car elles peuvent se dérouler en un ou plusieurs tours.

Nous allons maintenant présenter différentes plates-formes de négociation et le modèle sur lequel elles s'appuient : le *Contract Net Protocol*.

2.3 État de l'art

Les différentes formes de négociations sont étudiées par de nombreux chercheurs du monde entier et différentes plates-formes de négociation ont vu le jour. Tous ces travaux s'appuient sur le modèle général du *Contract Net* qui a sans aucun doute été le précurseur des travaux en négociation automatique. Dans cette section, nous présentons les principaux travaux en la matière. Nous commencerons bien évidemment par le *Contract Net* qui est sans aucun doute le plus utilisé par la communauté qui s'intéresse à ce sujet.

2.3.1 Le *Contract Net Protocol*

Le *Contract Net Protocol* [Smith, 1980], proposé par Smith en 1980, est un protocole de haut niveau pour la communication entre les nœuds d'un résolveur de problèmes distribué. Il facilite le contrôle distribué de l'exécution de tâches coopératives avec une communication efficace entre les nœuds. Pour Smith, la négociation comporte quatre composantes importantes :

- c'est un processus local qui n'implique pas un contrôle centralisé,
- l'échange d'information a lieu dans les deux sens,
- chaque partie dans la négociation évalue les informations de sa propre perspective,
- l'accord final est réalisé par une sélection mutuelle (le contractant a choisi de répondre au manager et le manager a choisi d'affecter la tâche au contractant).

Le manager va :	Un contractant va :
1. annoncer la tâche dont il veut qu'elle soit accomplie, 2. recevoir et évaluer les propositions des contractants, 3. choisir l'une de ces offres, 4. recevoir et synthétiser les résultats.	1. recevoir la demande de travail, 2. évaluer sa capacité à y répondre, 3. répondre (je ne peux pas, faire une offre), 4. si son offre est retenue, faire le travail, 5. renvoyer ses résultats.

FIG. 2.11 – *Description des rôles de manager et de contractant*

L'ensemble des nœuds est dénommé un *Contract Net* et l'exécution d'une tâche est traitée par un contrat entre deux nœuds. Chaque nœud dans le réseau endosse l'un des deux rôles relatifs à l'exécution d'une tâche individuelle : manager ou contractant (Figure 2.11). Un manager est responsable de la surveillance de l'exécution d'une tâche et du traitement des résultats de son exécution. Un contractant est responsable de l'exécution effective de la tâche. Les nœuds dans leur individualité ne sont pas conçus a priori comme étant managers ou contractants, ce sont seulement des rôles et chaque nœud peut prendre dynamiquement l'un des rôles pendant la durée de la résolution de problème. Typiquement, un nœud endossera les deux rôles, souvent simultanément, pour différents contrats. Finalement, les nœuds ne sont pas assujettis statiquement à une hiérarchie de contrôle. Cela conduit également à une utilisation des nœuds plus efficace, en comparaison, par exemple, avec les schémas qui n'autorisent pas les nœuds qui ont délégué des tâches à en contracter d'autres pendant qu'ils attendent les résultats.

La Figure 2.12 décrit la spécification BNF du protocole du *Contract Net*. Les symboles non terminaux sont entourés par des chevrons, les terminaux sont écrits sans dé-

```
<message> => <header> <addressee> <originator> <text> <trailer>
<header> => [line-header] [identifier] [time] [acknowledge]
<trailer> => [error-control] [line-trailer]
<addressee> => [net-address] | [subnet-address] | [node-address]
<originator> => [node-address]
<text> => [cctext] | <pstext>
<pstext> => <task-announcement> | <bid> | <announced-award> |
            <directed-award> | <acknowledgement> | <report> |
            <termination> | <node-available-message> |
            <request-message> | <information-message>
<task-announcement> => TASK-ANNOUNCEMENT [name]
                       {task-abstraction} {eligibility-specification}
                       {bid-specification} [expiration-time]
<bid> => BID [name] {node-abstraction}
<announced-award> => ANNOUNCED-AWARD [name] {task-specification}
<directed-award> => DIRECTED-AWARD [name] {task-abstraction}*
                    {eligibility-specification} {task-specification}
<acknowledgement> => ACCEPTANCE [name]*
                  => REFUSAL [name] {refusal-justification}
<report> => INTERIM-REPORT [name] {result-description}
         => FINAL-REPORT [name] {result-description}*
<termination> => TERMINATION [name]
<node-available-message> => NODE-AVAILABLE {eligibility-specification}*
                            {node-abstraction} [expiration-time]
<request-message> => REQUEST [name] {eligibility-specification}*
                     {request-specification}
<information-message> => INFORMATION [name] {eligibility-specification}*
                         {information-specification}
```

FIG. 2.12 – *Spécification BNF du protocole du Contract Net (Figure issue de [Smith, 1980])*

limiteurs et les non terminaux dont l'expansion n'est pas utile pour la discussion sont entourés par des crochets. Les termes devant être remplis par des informations codées dans le langage commun entre les nœuds sont entourés d'accolades. Les types de message qui n'ont pas besoin d'être inclus dans une implémentation basique sont suivis d'une étoile.

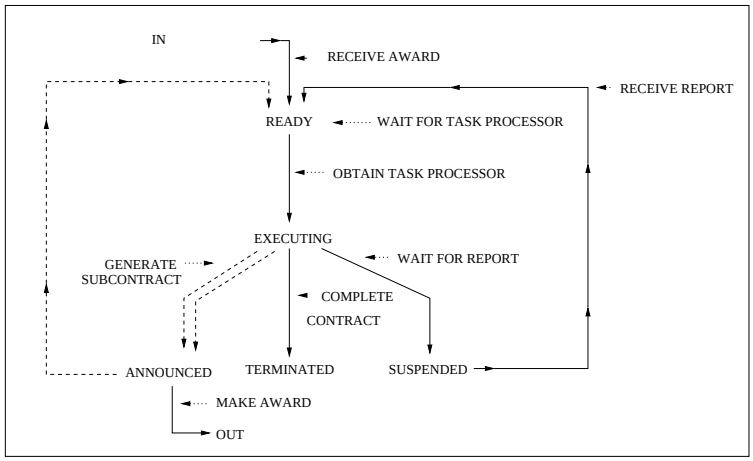


FIG. 2.13 – *États de traitement d'un contrat (Figure issue de [Smith, 1980])*

Les contrats sont traités selon le diagramme de transition d'états présenté en Figure 2.13, qui utilise la terminologie des systèmes d'exploitation. Le processeur de contrat d'un nœud est responsable de la transition du contrat à travers le diagramme. La progression standard entre les états est en lignes solides. Les progressions optionnelles sont en lignes pointillées.

Lorsqu'un contrat est affecté à un nœud (`IN`), il est placé dans l'état `READY` où il attend que le processeur de tâches soit libre. Il est alors placé dans l'état `EXECUTING` et son exécution est entamée. Si des sous-contrats sont générés, ils sont placés dans l'état `ANNOUNCED`. Un sous-contrat peut être soit affecté à un autre nœud (`OUT`), soit au même nœud (`READY`).

Si l'exécution d'un contrat ne peut continuer jusqu'à ce qu'un évènement se produise (réception d'un rapport de sous-contrat, par exemple), le contrat est alors placé dans l'état `SUSPENDED`, et un autre contrat dans l'état `READY` est choisi pour être traité. Lorsque l'évènement attendu se produit, le contrat est transféré de retour dans l'état `READY` où il attend la suite de son traitement.

Lorsque le traitement d'un contrat est terminé, il est placé dans l'état `TERMINATED` et un autre contrat dans l'état `READY` est choisi pour être traité. Les contrats récemment traités sont maintenus dans l'état `TERMINATED` de façon à faciliter le rattrapage d'une panne du manager.

Si un contrat dans l'état `EXECUTING` est déplacé vers l'état `SUSPENDED` ou `TERMINATED` et qu'aucun autre contrat n'est dans l'état `READY`, le nœud tente alors d'acquérir un nouveau contrat, soit en proposant une offre pour une annonce de tâche récente, soit en transmettant un message indiquant qu'il est libre.

```
<contract> => [name]
              [manager]
              [report-recipients]
              [related-contractors]
              <task>
              [results]
              <subcontract-list>
```

FIG. 2.14 – *Structure d'un contrat (Figure issue de [Smith, 1980])*

La spécification d'un contrat est présentée en Figure 2.14. Le `nom` est l'unique identifiant du contrat, le `manager` est le nœud qui a généré la tâche, les `destinataires de rapport` sont les nœuds à qui les rapports du contrat doivent être envoyés, le destinataire par défaut est le manager. Les `contractants en relation` avec le contrat sont les nœuds qui sont en train de travailler sur des contrats en relation avec celui-ci (par exemple, des sous-contrats du même contrat). La `tâche` se compose du type de tâche, de sa spécification et des procédures requises pour mener à bien la tâche dans le réseau de contrats. Les `résultats` sont ce qui est produit par l'exécution de la tâche, ils sont transmis aux destinataires de rapport du contrat. La `liste des sous-contrats` est la collection des sous-tâches générées depuis le contrat initial. Un sous-contrat contient l'information gardée par un manager pour une sous-tâche.

Le *Contract Net* est un protocole basé sur l'échange de contrats, qui met en relation un agent, le manager, avec plusieurs autres agents, les contractants. C'est donc de la négociation de 1 vers n agents. Comme avec notre modèle, plusieurs négociations pour des contrats différents peuvent avoir lieu simultanément et un agent peut tenir les deux rôles simultanément pour différentes négociations. Mais le *Contract Net* est un modèle où seul le manager émet des propositions, les contractants ne peuvent que faire une offre et pas de contre-proposition. En revanche, notre modèle inclut un processus de contre-propositions afin de prendre en compte l'avis des contractants, de manière à aboutir plus rapidement à une solution acceptable par tous en comparaison avec un modèle où le manager est le seul à décider quelle nouvelle proposition envoyer sans savoir si celle-ci va dans le sens des contractants. De plus, il n'y a pas de message indiquant à un contractant qu'il n'a pas été retenu pour la tâche. Un contractant n'a donc pas le moyen de savoir si l'offre qu'il a soumise n'a pas été retenue ou si le manager n'a pas encore pris sa décision, ce qui peut entraîner qu'un contractant ne fasse pas d'offre pour une autre tâche alors qu'il n'a pas été choisi pour la précédente. Dans notre modèle, un tel message existe et permet à un agent de savoir si la négociation est terminée ou non.

Le *Contract Net* fait partie des spécifications de la *FIPA* (Foundation for Intelligent Physical Agents) et une version itérée du protocole est également spécifiée. Bien que le *Contract Net* soit une référence parmi les protocoles de négociation, celui-ci ne convient pas pour de nombreux problèmes, ce qui explique les nombreux travaux sur des extensions de celui-ci, parmi lesquels les travaux de Tuomas Sandholm sur le *Traconet*.

2.3.2 *Traconet*

Ce projet de Tuomas Sandholm [Sandholm, 1993] est destiné à résoudre le problème de routage de véhicules (VRP). Pour cela, il propose une extension du *Contract Net* qui formalise le processus de décision d'enchérissement et d'attribution de tâches. Cette formalisation se base sur le calcul de coûts marginaux relativement aux critères locaux de l'agent. De cette façon, les agents ayant des critères locaux très différents (basés sur leur propre intérêt), peuvent interagir afin de distribuer les tâches pour que le système global fonctionne plus efficacement. Dans ce modèle, des agents compétitifs aussi bien que coopératifs peuvent interagir. De plus, le *Contract Net Protocol* est étendu afin de permettre, pour un regroupement de tâches, de traiter un grand nombre possible de messages d'annonce et d'enchère et d'effectivement traiter les situations où de nouvelles enchères et attributions se déroulent alors que les résultats des enchères précédentes sont inconnus. Le protocole est vérifié par le système *Traconet* (TRAnsportation COoperation NET), où les centres de distribution de différentes compagnies coopèrent automatiquement pour le routage de véhicules. L'implémentation est asynchrone et distribuée et elle procure aux agents une autonomie étendue. Comme le *Contract Net*, *Traconet* permet de réaliser une négociation de 1 vers m agents.

Les insuffisances du *Contract Net* que Sandholm a identifiées et traitées dans cette implémentation sont les suivantes :

- il n'y a pas de modèle formel pour la prise de décision concernant l'annonce des tâches, les offres et les attributions. Dans cette implémentation, un modèle basé sur les coûts marginaux pour exécuter des ensembles de tâches est utilisé.
- la prise de risque d'un agent pour se voir attribuer des activités qu'il ne pourrait pas honorer, ou que l'accomplissement de celles-ci ne lui serait pas bénéfique n'est pas pris en compte.
- une seule tâche est annoncée à la fois, ce qui peut ne pas être suffisant si l'effort de déléguer une tâche dépend de la délégation d'autres tâches. Le framework a donc été étendu pour gérer l'interaction entre les tâches par le regroupement de tâches dans des ensembles qui seront négociés comme des entités élémentaires.
- le problème pratique de la congestion des messages d'annonce est résolu dans *Traconet*.

De plus, dans cette extension du *Contract Net*, un message indiquant aux contractants qu'ils n'ont pas été retenus a été ajouté au protocole.

Dans une autre extension du *Contract Net* [Sandholm and Lesser, 1995], Sandholm et Lesser ont ajouté la possibilité d'effectuer des contre-propositions et des niveaux d'obligation vis à vis des contrats attribués. Ils ont également introduit le concept d'allocation de plusieurs tâches en même temps à un même agent, afin de réduire le temps d'exécution des tâches lorsque celles-ci sont interdépendantes. Afin d'améliorer la performance des agents, Sandholm et Lesser considèrent que les agents qui traitent leurs tâches séparément peuvent parfois s'échanger certains contrats.

2.3.3 *Kasbah*

Kasbah [Chavez and Maes, 1996] a été développé au MIT Media Lab par Pattie Maes. C'est un système où les utilisateurs créent des agents pour négocier la vente et l'achat de biens pour leur compte sur Internet. Ces biens sont classifiés, reprenant ainsi l'idée des petites annonces classées par type.

Lors de la création d'un agent, pour la vente ou l'achat, l'utilisateur spécifie le type de bien à négocier, la date à laquelle il souhaite que la transaction soit effectuée, le prix désiré, le plus petit (resp. grand) prix acceptable et la stratégie de négociation à choisir parmi les 3 proposées qui correspondent aux fonctions linéaire, quadratique et exponentielle pour le calcul de l'évolution du prix selon le temps. L'utilisateur précise également si l'agent doit demander son accord avant de conclure la vente et s'il veut être averti par mail lorsqu'un accord est trouvé. Une fois l'accord atteint, la transaction physique peut avoir lieu, ce qui doit être réalisé par les agents humains.

Du point de vue de l'implémentation, *Kasbah* met en relation les agents ayant des buts communs, les communications entre agents se font de 1 vers 1. Le fonctionnement en parallèle des agents est simulé en accordant un *time-slice* à chacun à tour de rôle. Durant ce time-slice, l'agent détermine le prix courant désiré, décide avec quel agent communiquer et enfin communique avec celui-ci. Les agents communiquent via des actes de langages spécifiques à *Kasbah*. Si une évolution vers KQML a été prévue, ce système n'en reste pas moins axé sur l'e-commerce, qui est un aspect très spécifique de la négociation.

2.3.4 *AuctionBot*

AuctionBot [Wurman et al., 1998] est un serveur d'enchères expérimental développé et opérationnel au laboratoire d'intelligence artificielle de l'université du Michigan par Michael P. Wellman et Peter R. Wurman. Son but est de permettre à n'importe quel internaute de participer aux enchères sur le net.

AuctionBot est un framework utile à la fois pour le commerce et pour la recherche au sens où il propose une large variété de types d'enchères (English, Vickrey, CDA,

etc.) et une API pour créer ses propres agents qui participeront à la place de marché d'*AuctionBot*. Son architecture est asynchrone, il stocke les enchères dans une base de données et il peut en gérer plusieurs simultanément. Afin de participer aux enchères, il faut s'enregistrer. Les utilisateurs humains peuvent consulter leurs comptes via une page web ou choisir d'être avertis de l'avancement des enchères par mail. Comme *AuctionBot* répertorie les enchères proposées dans un catalogue organisé de façon hiérarchique, un vendeur peut placer son enchère n'importe où dans le catalogue existant ou étendre celui-ci. Il peut également choisir de mettre son enchère dans le catalogue public ou de la proposer à un groupe privé.

Du point de vue de l'implémentation, les agents placent les enchères dans la base de données, tandis qu'*AuctionBot* collecte les enchères, détermine une allocation d'après un ensemble de règles d'enchères bien défini et avertit les participants. En revanche, il n'exécute pas les transactions, il n'impose pas les échanges ni ne vérifie la crédibilité des participants. Ce n'est donc pas l'agent initiateur du contrat qui gère l'enchère mais un programme d'*AuctionBot*. *AuctionBot* ne propose un support que pour les mécanismes d'enchères, tandis que notre objectif est de fournir une plateforme générique de négociation qui permet d'effectuer des enchères mais aussi d'autres types de négociation tels que la prise de rendez-vous, le choix d'un restaurant, etc. De plus, nous laissons aux agents l'autonomie de la gestion de la négociation, chaque agent initiateur (vendeur dans le cas des enchères) gère le déroulement de sa négociation, il n'y a pas de centralisateur d'offres de vente et d'achat qui effectue l'appariement selon les règles d'enchères définies comme dans *AuctionBot*.

Le serveur d'*AuctionBot* est utilisé pour TAC (Trading Agent Competition)(cf. page 31, une compétition qui met en concurrence des agents chargés de négocier les composants d'un voyage (vol aller, nuits d'hôtel, sorties et vol retour) pour un ensemble de 8 clients, créée elle aussi par Michael P. Wellman et Peter R. Wurman. Les vols, les nuits d'hôtel et les sorties sont négociés par des enchères de différents types. Chaque client a des préférences pour le jour d'arrivée, le jour de départ, le type d'hôtel (luxueux ou standard) et les sorties possibles. Chaque agent est ensuite noté en fonction de ses dépenses, de ses revenus et de la satisfaction de ses clients. Cette compétition rassemble une trentaine de participants chaque année depuis 2001.

2.3.5 Le projet *ADEPT*

Le projet *ADEPT* (Advanced Decision Environment for Process Tasks) [Alty et al., 1994, Jennings et al., 1998], de Nick Jennings et al., recherche à la fois la technologie et les méthodes nécessaires à l'amélioration de la manière de recueillir des informations, de les gérer, de les distribuer et de les présenter aux personnes, dans des fonctions et des opérations clés de l'entreprise. Ce projet crée un *Environnement d'Information Concourant*, qui doit fournir un support pour la création d'accords, la négociation et la gestion de conflits. Ce projet comprend donc un module de *négociation de services*. Un service est défini par les caractéristiques suivantes :

- Chaque service possède un **nom** qui est unique.
- Chaque service possède une **garde**. Une garde déclare les informations qui sont requises pour lancer le service. Des contraintes peuvent également être exprimées, comme par exemple : un service ne peut être lancé que lorsque une certaine valeur `X` est disponible et supérieure à 5, etc.
- Chaque service peut avoir des **hypothèses**. Une hypothèse peut imposer des contraintes supplémentaires sur l'information, ou assigner des valeurs par défaut (qui sont alors crues, mais peut-être encore inconnues).
- Chaque service possède un **corps** qui décrit la façon dont le service peut être exécuté.

Un service peut être défini récursivement en termes de services fournis par d'autres agents. La communication entre les agents s'effectue selon le *modèle générique de négociation de services/d'informations* qui a été développé. Ce modèle est le suivant :

1. `Trouver/Localiser` (un service ou une information) - établir le contact.
2. `Demander/Expliquer` (quelle est la forme ou la nature du service ou de l'information) - obtenir des détails.
3. `Établir/Négocier` (en mettant en place un service ou en recevant des informations) - se mettre d'accord sur ce qui doit être fait.
4. `Exécuter/Réviser/Informer` (faire le service ou délivrer l'information, avec la possibilité de réviser la progression et d'informer si nécessaire) - faire ce qui a été conclu.
5. `Terminer`.

Il est important que ce modèle doive s'appliquer aussi bien aux requêtes de service explicites qu'au plus général et implicite service d'exigence de trouver, localiser et gérer les informations au sein d'une entreprise. Ce modèle a été développé dans un ensemble de primitives spécifiques qui pourraient être utilisées comme une base pour explorer plus finement la nature précise de la sémantique de la négociation. Nous ne présentons ici que les primitives concernant le troisième point du modèle : la négociation.

```
PROPOSE<agent a, service s,conditions c,optionale rationale r>
MODIFY
REJECT
ACCEPT
--> return<agreement(working conventions, reporting level,
                                        penalties, etc)>
```

Ces primitives de base permettent aux agents de se mettre d'accord sur la façon dont un service sera fourni. Les conditions seront séparées en *obligatoires* et *désirables*. Le raisonnement optionnel et les détails de la provision de service devraient fournir aux agents un mécanisme pour casser les deadlocks qui proviendraient de requêtes obligatoires (non nécessaires), ou de fournir plus de formes nouvelles, d'alternatives de provision de service. Cela devrait permettre de développer, rechercher beaucoup plus

de formes sophistiquées de processus de négociation. La chose importante à noter est que l'acceptation d'une proposition, après de possibles modifications (et peut-être des demandes, des explications), résulte en un accord de niveau de service spécifique qui peut alors être planifié pour être exécuté. Si un problème survient lors de l'exécution du service, celui-ci peut être renégocié automatiquement s'il n'y a pas d'autre solution (comme le redémarrer, par exemple).

Dans [Sierra et al., 1997], les auteurs détaillent le modèle de raisonnement des agents pour la négociation orientée service où ils se concentrent sur les négociations multi-parties, multi-issues, simple rencontre dans un environnement où les ressources (parmi lesquelles le temps) sont limitées. Pour eux, un modèle détaillé et complet de raisonnement pour la négociation orientée services devrait déterminer :

- quels serveurs potentiels devraient être contactés,
- la négociation devrait-elle être exécutée en parallèle sur tous les serveurs ou séquentiellement,
- quelles offres initiales devraient être envoyées,
- quel est l'intervalle des accords acceptables,
- quelles contre-propositions devraient être générées,
- quand la négociation devrait être abandonnée,
- quand un accord est atteint.

Le cadre de l'application traitée dans cet article est basé sur les études de British Telecom pour évaluer le coût de la mise en place d'un service demandé par les usagers.

Pour cela, les agents seront les différents acteurs de la mise en place d'un service. Les agents utilisent un modèle de négociation bilatérale qui leur permet de formuler des offres, de les évaluer et de formuler des contre-propositions. Les offres sont caractérisées par un certain nombre de paramètres chiffrés. Les tactiques des agents définissent leurs réactions à ces paramètres, alors que la stratégie des agents gère l'importance des aspects paramétrés de la négociation au cours du temps.

Plusieurs tactiques ont été développées et mises en œuvre, insistant sur la valeur du temps (qui est lui aussi une ressource limitée), les ressources matérielles,etc. La convergence du système n'est pas automatique.

Ces travaux ont ensuite beaucoup inspirés leurs concepteurs pour le développement d'une application *FIPA* (Foundation of Intelligent Physical Agent) dans le domaine des télécommunications de la technologie agents [Faratin et al., 1999].

Comme notre modèle, *ADEPT* propose une séquence de contre-propositions lors de la négociation afin de parvenir à un accord mutuel. Le modèle de raisonnement proposé tient également compte des préférences de l'agent et utilise une base qui mémorise les services de l'agent et une pour les services supposés des autres agents. Ce modèle de raisonnement doit également définir si les négociations se font séquentiellement ou en parallèle et la définition d'un service doit permettre d'éviter les deadlocks. De plus,

lorsque l'agent s'aperçoit de l'échec de l'exécution d'un service, il est possible de le renégocier. Mais *ADEPT* est dédié à la négociation de service et utilise de la négociation bilatérale (1 initiateur, 1 participant), tandis que notre modèle permet de la négociation de n initiateurs vers m participants et a pour but d'être générique, donc de s'appliquer à différents types d'applications de négociation et pas uniquement de la négociation de services, par exemple.

2.3.6 *Fishmarket*

Fishmarket [Noriega, 1998] est une agence d'enchères électroniques pour vendre du poisson, où les agents peuvent être soit humains, soit virtuels et qui a été développée à l'Institut de Recherche en Intelligence Artificielle en Espagne par Pablo Noriega.

Le marché au poisson est une place où se déroulent différentes scènes qui ont lieu en même temps mais en des pièces différentes et qui ont une continuité causale. Chaque scène implique divers agents qui agissent selon des règles bien définies à ce moment précis. La scène principale est l'enchère où les acheteurs font des offres pour des caisses de poisson présentées par un commissaire-priseur qui annonce les prix dans l'ordre décroissant (protocole d'enchères descendantes). Cependant, avant que ces caisses de poisson soient vendues, les pêcheurs ont dû livrer le poisson au marché (dans la scène d'admission des vendeurs) et les acheteurs ont dû s'enregistrer dans le marché (dans la scène d'admission des acheteurs). De la même façon, une fois une caisse de poisson vendue, l'acheteur doit la retirer en passant par la scène des règlements des acheteurs, pendant que les vendeurs doivent retirer leur paiement dans la scène des règlements des vendeurs une fois leur lot vendu.

La Figure 2.15 montre le protocole utilisé pour la vente des caisses de poisson. Le commissaire-priseur annonce le prix de vente du poisson et collecte les offres des acheteurs. L'offre d'un acheteur est une promesse d'achat au prix indiqué par le commissaire-priseur. Une fois les offres collectées pendant la période de temps allouée, le commissaire-priseur compte les offres reçues. S'il n'y en a pas, il décrémente le prix et annonce à nouveau le prix de vente du poisson. S'il y en a une, l'acheteur ayant fait l'offre remporte la vente si son crédit le permet, sinon, il doit payer une amende et le commissaire-priseur remet en vente le poisson au prix incrémenté de 25%. Si plusieurs offres sont reçues, le commissaire-priseur déclare qu'il y a eu collision et remet en vente le poisson à un prix incrémenté de 25%. Lorsque le poisson a été vendu, le crédit de l'acheteur est mis à jour, le vendeur est payé et l'acheteur peut chercher son poisson.

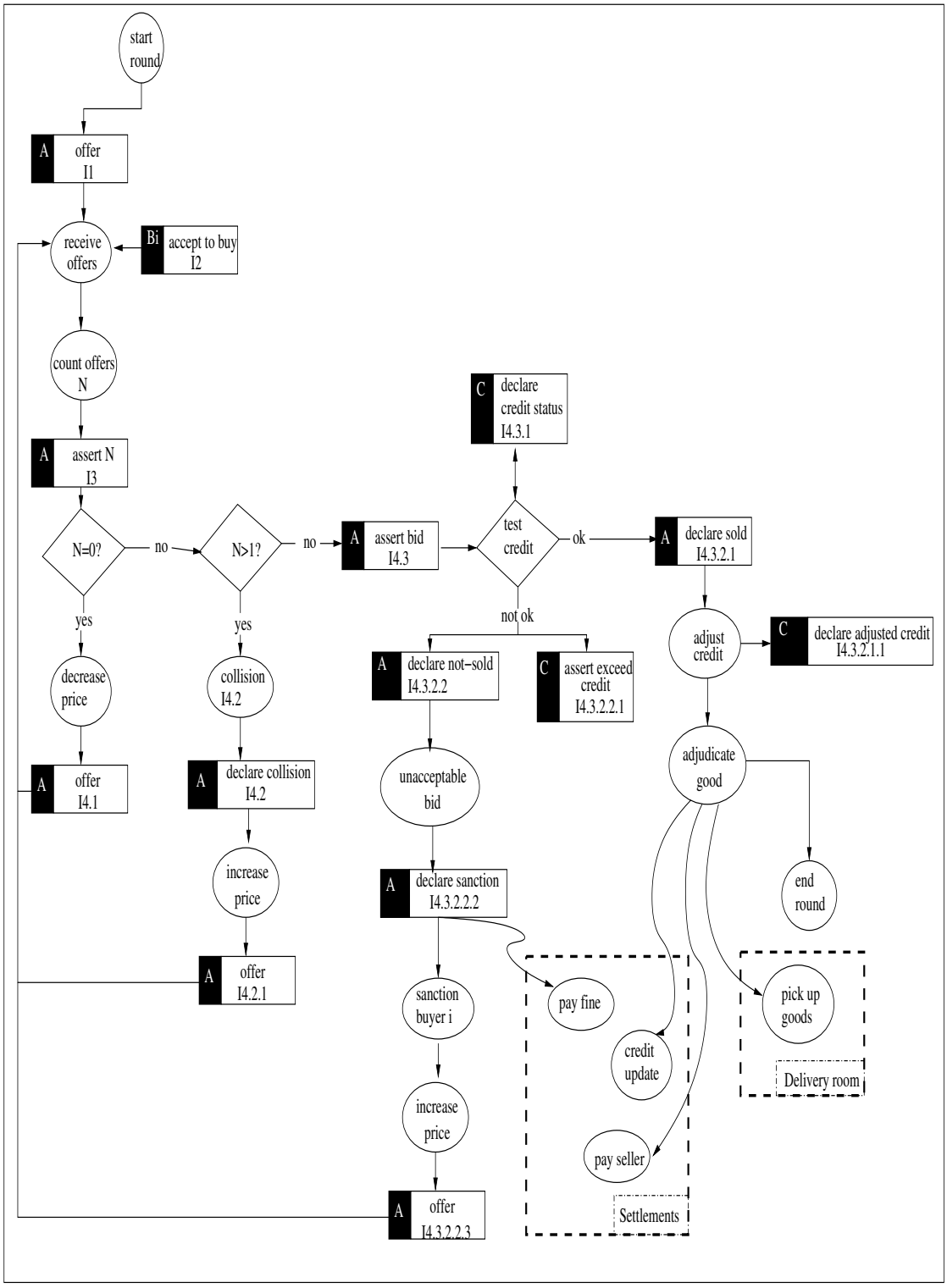


FIG. 2.15 – *Protocole d'enchères descendantes de Fishmarket (Figure issue de [Rodriguez-Aguilar et al., 1997])*

Un aspect important du marché au poisson, qui peut être directement transféré à la version électronique, est la présence d'intermédiaires du marché : le commissaire-priseur, un patron du marché, un réceptionniste et un officier de crédit. Ces intermédiaires interagissent avec les acheteurs et les vendeurs pour le compte du marché et ont donc l'autorité de demander, reconnaître, rejeter ou accepter toutes les actions que les acheteurs et les vendeurs doivent exécuter au sein du marché. De plus, toutes ces interactions entre les intermédiaires du marché et les agents extérieurs (acheteurs ou vendeurs) peuvent en fait être associées à des allocutions standardisées, dont certaines sont probablement tacites dans le marché, mais néanmoins rendues explicites dans le modèle computationnel.

Fishmarket a été conçu pour montrer la complexité de ces interactions tout en gardant aussi fortement que possible une similarité avec l'ontologie des éléments du marché de poisson. Chaque agent logiciel de *Fishmarket* représente soit un intermédiaire du marché, soit un vendeur ou encore un acheteur. De même les allocutions (tacites ou explicites) utilisées dans le marché ont été reprises en allocutions toujours explicites pour les agents logiciels.

Fishmarket utilise un cadre dialectique et peut être étendu afin de convenir aux différents types d'enchères (anglaises, hollandaise, Vickrey, etc.). Les enchères y sont traitées séquentiellement, tandis que notre modèle permet d'effectuer plusieurs négociations en parallèle lorsqu'il n'y a pas de conflit sur les ressources.

2.3.7 *Zeus*

Zeus [Nwana et al., 1999] est une API Java qui permet de programmer une application multi-agents, développée par British Telecom. Le but de *Zeus* est de proposer une boîte à outils qui puisse être appliquée à une large variété de problèmes et non à des variations sur une application particulière. Elle a donc le même but que nous.

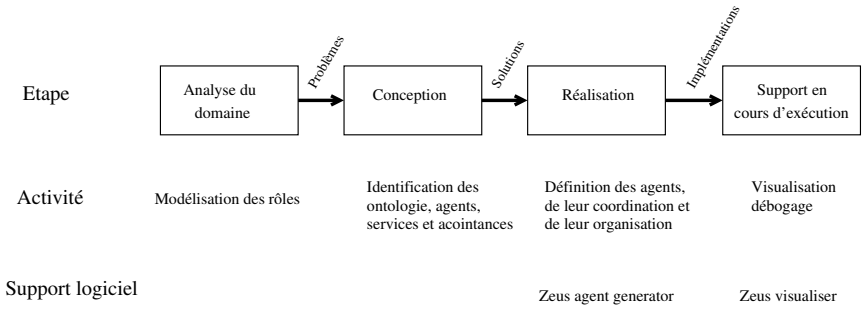


FIG. 2.16 – *Méthodologie pour la réalisation d'applications avec Zeus (Figure issue de [Collis and Ndumu, 1999a])*

Afin de réaliser une application à l'aide de *Zeus*, la méthodologie représentée en Figure 2.16 est proposée.

Comme pour toute application, il faut commencer par analyser le sujet. Vient ensuite la phase de conception où il faut définir l'ontologie, c'est-à-dire les différents *faits* nécessaires (les faits correspondent aux ressources pour notre type d'applications), les différents agents qui vont intervenir, les services dont ils auront besoin ainsi que leurs accointances. La réalisation de l'application avec *Zeus* est facilitée par un support logiciel : le *Zeus Agent Generator*. Il permet de définir l'ontologie, les agents et les tâches. Puis le *Code Generator* permet de générer le code correspondant aux définitions réalisées avec l'*Agent Generator*.

Une suite de logiciels support pendant l'exécution est également fournie avec *Zeus*. Elle comprend le visualiseur de la société, l'outil de rapport, l'outil de statistiques, le visualiseur d'agent et l'outil de contrôle.

Zeus propose une modélisation de rôles pour 5 applications dans 3 domaines :
- Le domaine de gestion d'informations :
  - L'espace d'informations partagées.
- Le domaine du commerce entre plusieurs agents :
  - La place de marché distribuée.
  - Les enchères institutionnelles.
- Le domaine des processus industriels :
  - La chaîne de production simple.
  - La chaîne de production contractuelle.

Chaque agent possède un objet appelé *Co-ordination Engine* dont le but est de gérer les comportements lors de résolution de buts et en particulier ceux relatifs à la collaboration entre agents. En plus des besoins généraux pour la déclaration du comportement à adopter, il a fallu tenir compte du fait qu'un agent doit être capable d'engager plusieurs tâches simultanément. Cela signifie que le *Co-ordination Engine* doit supporter une forme de multi-tasking. Cependant, en raison des coûts impliqués, le multithreading a été jugé inapproprié, sachant que le nombre de tâches indépendantes pouvait atteindre des centaines. C'est pourquoi les comportements de résolution de but sont représentés par des graphes de réseau de transitions récursifs, qui sont interprétés par une machine à états finis récursive.

Fonctionnement général

Le traitement d'un graphe consiste à partir d'un nœud initial et tenter de traverser le graphe jusqu'à ce qu'un nœud terminal soit atteint. Les nœuds du graphe implémentent des traitements à réaliser et les arcs déterminent si le passage d'un nœud à un autre est valide. Les nœuds et les arcs acceptent en entrée un argument sur lequel ils agissent pour retourner un argument en sortie. Ainsi l'information circule en même

temps que le graphe est traversé. Pour permettre la récursivité, les graphes sont aussi des arcs.

Tous les nœuds doivent implémenter 2 fonctions : `exec()` et `reset()`.
- `exec()` : exécute le traitement associé au nœud et retourne l'une des valeurs OK, FAIL ou WAIT. La valeur de retour OK signifie que le traitement a réussi, alors que FAIL indique un échec. La valeur WAIT indique que le traitement doit être interrompu jusqu'à ce qu'un timeout expire ou qu'un message spécifié par une valeur de clé de réponse soit reçu.
- `reset()` : retire tous les changements qui ont été faits sur la donnée d'entrée par la fonction `exec()`, ceci afin de permettre le backtracking.

Les arcs, quant à eux, doivent implémenter la fonction `test()` qui retourne un booléen indiquant si sa traversée est valide ou non.

Multi-tasking Il est réalisé grâce à une file d'attente pour les nœuds de type FIFO.

Parallèlisme Certains arcs ou graphes peuvent être désignés comme étant parallèles. Des copies (non parallèles) du graphe/arc sont alors créées pour chaque élément du tableau d'entrée et ces copies sont gérées d'une façon k parmi n, c'est-à-dire que pour que l'arc/le graphe soit traversé, k copies parmi les n doivent avoir réussi.

Backtracking Prenons l'exemple d'un nœud ayant 2 arcs sortant. Si ce nœud retourne OK, mais que le premier arc retourne faux, alors le second arc est testé. Si celui-ci échoue également, la méthode `reset()` de ce nœud est appelée pour défaire tout changement réalisé par ce nœud et le nœud prédécesseur tente de traverser le graphe par un autre arc sortant de celui-ci. Le backtracking est aussi réalisé lorsque la fonction `exec()` d'un nœud retourne FAIL.

Communication Dans le cadre d'un graphe, le support pour la communication entre agents est réalisée par l'utilisation de la valeur de retour WAIT de la fonction `exec()`. Par exemple, un nœud engagé dans une communication enverra un message puis demandera à être suspendu jusqu'à ce qu'une réponse arrive ou qu'un timeout expire. Ce nœud ne sera alors remis dans la file d'attente que lorsque le message sera arrivé ou que le délai aura expiré.

Le graphe du *Contract Net Protocol*

Il y a un graphe pour l'initiateur et un pour le participant. La Figure 2.17 et la Table 2.1 décrivent l'implémentation du *Contract Net Protocol* réalisée par *Zeus*.

Dans cette implémentation, on remarque que les agents qui sont considérés comme des *collègues* sont préférés aux agents *pairs* par l'initiateur de la négociation (l'arc A_1 précède l'arc A_2 dans la spécification).

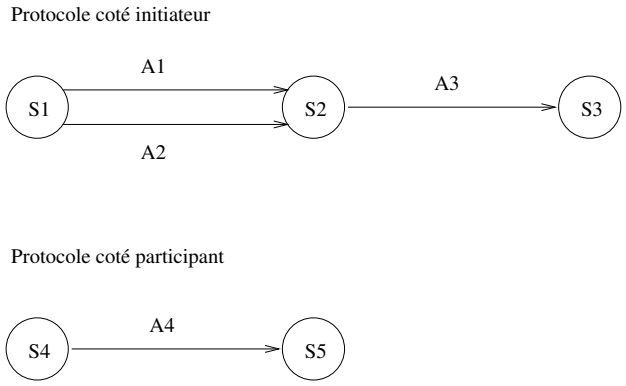


FIG. 2.17 – *Graphe d'implémentation du Contract Net Protocol (Figure issue de [Collis and Ndumu, 1999b])*

Comportement du contract − net côté initiateur	
Nœud/Arc	**Description/Condition de transition**
S1	Indentifie les agents capables de réaliser le but.
A1	Sélectionne un sous-ensemble d'agents capables de réaliser le but et qui sont des collègues (vérifie $\neq \emptyset$).
A2	Sélectionne un sous-ensemble d'agents capables de réaliser le but et qui sont des pairs (vérifie $\neq \emptyset$).
S2	Envoie un appel d'offres pour sélectionner les agents et attend les réponses.
A3	Vérifie qu'une réponse positive a été reçue.
S3	Fait

Comportement du contract − net côté participant	
Nœud/Arc	**Description/Condition de transition**
S4	Evalue le coût. Envoie un message d'accord. Attend la réponse.
A4	Message de confirmation de contrat reçu.
S5	Fait.

TAB. 2.1 – *Description des nœuds et arcs de la Figure 2.17 (Tableaux issus de [Collis and Ndumu, 1999b])*

Dans l'état actuel des choses, le protocole fourni ne traite que des buts portant sur un unique fait (ie. sur une unique ressource). Ceci constitue un point négatif pour *Zeus*. En effet, lors de négociation pour la prise de rendez-vous, les ressources sont des tranches horaires. Or, un rendez-vous peut nécessiter plusieurs tranches horaires, par exemple les tranches horaires peuvent être d'une durée d'une heure et le rendez-vous de 2 heures. Notre application, quant à elle, permet de gérer des contrats sur plusieurs ressources.

De plus, dans *Zeus*, une fois le contrat accepté, il est impossible de se rétracter. Notre protocole permet cela, afin de prendre en compte les priorités définies sur les participants et sur les ressources. Ce qui veut dire que si un contrat plus important est proposé à l'agent, il sera rejeté dans le cas de *Zeus* mais accepté par notre protocole et le contrat en conflit sera annulé.

Un autre point important à remarquer est que la négociation s'effectue de 1 agent vers 1 autre agent. Il n'est pas possible avec *Zeus* de négocier de 1 vers n agents avec le protocole fourni. Ce protocole réalise des contrats entre 2 agents uniquement, il peut exécuter plusieurs négociations de 1 vers 1 simultanément, mais 1 seule sera confirmée, les autres annulées. Dans le cas de la prise de rendez-vous, une réunion réunit en général plusieurs participants et un pourcentage d'acceptation doit être atteint. Il n'y a donc pas n négociations simultanées dont l'une aboutira mais 1 négociation avec n participants qui aboutira si le pourcentage d'acceptation est atteint.

D'autre part, ce protocole permet de lancer des appels d'offres, mais il ne permet pas de renégocier les offres proposées. Elles sont acceptées ou rejetées telles quelles, il n'y a pas de retour de la part de l'initiateur pour essayer d'aboutir à un accord commun. Notre protocole fournit des primitives afin de permettre la renégociation de contrat si l'initiateur et les participants ne sont pas satisfaits. Ainsi, les agents tentent de trouver un contrat qui les satisfait au mieux.

De plus, les stratégies proposées sont basées sur l'évolution du coût du but à réaliser. Or, tous les types de négociation ne comportent pas forcément un coût (comme par exemple la prise de rendez-vous).

2.3.8 *Magnet*

Magnet (Multi AGent NEgotiation Testbed) [Collins et al., 1998] est un banc de tests pour la négociation multi-agents, implémenté sous la forme d'une architecture généralisée de marché et développé à l'université du Minnesota par John Collins et al. Il fournit un support pour un large panel de transactions, du simple achat/vente de biens à la négociation complexe de contrats entre agents. Le but des auteurs est de concevoir, implémenter et analyser une architecture de marché multi-agents généralisée, qui peut fournir un support explicite et intégré pour les interactions complexes entre agents,

comme dans la prise de contrats automatisée, tout comme pour d'autres types de protocoles de négociation, incluant les enchères à offres scellées et la vente et l'achat à prix public ou offres publiques. Les auteurs introduisent également un protocole de prise de contrats flexible qui peut prendre complètement avantage de l'architecture de marché proposée pour faciliter les interactions entre agents. *Magnet* introduit un intermédiaire explicite dans la négociation qui aide à contrôler les fraudes et décourage les contre-spéculations et est organisé autour de trois composants de base : l'échange, le marché et la session.

L'*échange* est une collection de marchés spécifiques à un domaine dans lesquels les services et les biens sont échangés, agrémentée par des services génériques requis par tous les marchés, comme la vérification des identités des participants à une transaction.

Chaque *marché* au sein d'un échange est un forum pour le commerce d'une commodité particulière ou d'un secteur d'activités. Il y a des marchés voués à la publication, à la construction, au transport, etc. Chaque marché inclut un ensemble de services et d'aménagements spécifiques à son domaine et chaque marché s'appuie sur les services communs de l'échange.

Un composant important de chaque marché est l'ensemble des sessions de marché courantes dans lesquelles les interactions effectives entre agents ont lieu. Les agents participant à un marché doivent le faire en tant qu'initiateur de session, de client ou les deux à la fois. Chaque session est initialisée par un agent pour un propos particulier et en général de multiples agents peuvent rejoindre une session existante en tant que clients. Les éléments importants du marché incluent une ontologie spécifique au domaine spécifiant les termes du discours pour ce domaine, une spécification de protocole qui définit les types de négociation supportés par le marché et un registre de clients au sein du marché.

Une *session* (de marché) est le moyen par lequel les services du marché sont délivrés dynamiquement aux agents participant. Il sert aussi bien d'encapsulation pour une transaction au sein du marché, que de répertoire persistant pour l'état courant de la transaction. La session entoure la vie entière d'un contrat ou d'un ensemble de contrats relatifs. Le mécanisme de session assure la continuité des transactions à demi réalisées, protège contre la fraude, limite la contre-spéculation et permet aux agents participants de ne pas avoir à garder localement une trace des statuts détaillés de la négociation. Ce mécanisme de session permet aux agents de lancer un appel d'offres et de conduire d'autres travaux simultanément.

Le protocole de négociation pour la planification par contrats est constitué de trois phases : un appel d'offres, la collecte des enchères et la notification des résultats.

Un appel d'offres contient un ensemble de sous-tâches avec pour chaque sous-tâche l'intervalle de temps durant lequel le travail doit être accompli. Ce message inclut également une date limite de réception des offres, la date à laquelle le client étudiera les

propositions, la date au plus tôt où les notifications d'acceptation seront envoyées, et pour chaque sous-tâche une fonction de pénalité indiquant aux fournisseurs la pénalité en fonction du temps, dont ils seront redevables si ils n'arrivent pas à (décident de ne pas) accomplir le travail.

Ce message, une fois créé, est donné à la session de marché qui le rend disponible à tous les fournisseurs appropriés (ceux qui sont enregistrés au sein du marché et qui peuvent fournir les tâches nécessaires). En ce sens, l'appel d'offres est public, tandis que tous les autres messages sont privés. Avant de faire suivre le message, la session peut le vérifier pour être sûr qu'il est conforme à toutes les règles de marché et d'échange qui peuvent exister.

Chaque fournisseur ayant reçu l'appel d'offres l'inspecte et décide de faire ou non une offre, selon ses ressources, ses contraintes de temps et ses connaissances sur le travail à accomplir, selon le catalogue des services fourni par l'agent du marché.

Une offre est un message privé, qui peut contenir une combinaison de sous-tâches, qui peut être un sous-ensemble des sous-tâches listées dans l'appel d'offres. Dans l'offre, le fournisseur doit indiquer le coût pour le client, l'intervalle de temps et la durée estimée du travail pour la combinaison entière des sous-tâches, la date limite pour l'acceptation de l'offre et une fonction de pénalité pour le client si celui-ci attribue le travail au fournisseur puis se rétracte.

Chaque fournisseur peut envoyer plusieurs offres différentes pour un même appel d'offres mais une seule de ces offres sera attribuée à chaque fournisseur. Si un fournisseur n'envoie pas d'offre avant la date limite fixée par le client, celui-ci considère que le fournisseur a décidé de ne pas faire d'offre, le refus est donc passif.

Une fois les offres reçues, le client doit décider lesquelles accepter, en utilisant ses connaissances sur les offres, les valeurs des tâches et des sous-tâches, ses propres contraintes temporelles et le fournisseur ayant proposé une offre. Une fois ce processus accompli, le client doit choisir parmi ces trois actions pour chaque offre :

1. accepter l'offre entière,
2. accepter une partie de l'offre,
3. rejeter l'offre (ce qui est fait de façon passive).

Le message d'acceptation de l'offre sera envoyé à la session de marché qui le vérifiera, le validera et l'estampillera avant de le faire suivre au fournisseur. Les deux premiers choix sont des engagements à attribuer le travail au fournisseur et à partir de la date d'envoi du message, les fonctions de pénalité du client et du fournisseur sont effectives. Si un fournisseur ne reçoit pas d'acceptation de son offre, cela veut dire que le client l'a rejetée. Une fois les engagements pris, un agent peut déterminer qu'il ne peut pas faire les tâches pour lesquelles il s'est engagé, ou qu'il lui serait désavantageux de les faire. Dans ces situations, l'agent doit envoyer un message de désengagement indiquant quelles parties du travail il ne peut pas satisfaire. Le message de désengagement

comprend également une acceptation de la pénalité à payer par l'agent du fait de ce désengagement.

Magnet reprend le protocole du *Contract Net*, il n'y a donc pas de mécanisme de contre-propositions afin de trouver une solution. En revanche, notre protocole de négociation permet à l'initiateur de l'appel d'offres et aux participants de formuler des contre-propositions jusqu'à ce qu'un accord soit atteint. Dans *Magnet*, il y a un intermédiaire explicite dans le processus de négociation et les agents interagissent par son intermédiaire, tandis que les agents interagissent directement entre eux dans notre processus de négociation. La rétractation est possible moyennant une pénalité à payer, mais la renégociation ne se fait pas automatiquement, alors que c'est le cas dans notre modèle.

2.3.9 *SilkRoad*

Le projet *SilkRoad* [Ströbel, 2000, Ströbel, 2001a, Ströbel, 2001b, Ströbel and Stolze, 2001] est développé au laboratoire de recherche d'IBM à Zurich. Ce projet a pour but de faciliter la conception et l'implémentation de systèmes de support de négociation pour des domaines d'applications spécifiques. *SilkRoad* facilite les négociations multi-attributs dans les scénarios d'e-business grâce à une méthodologie de conception spécifique et à une architecture de système générique avec des composants de support de négociation réutilisables. Un système de support de négociation construit sur la base du modèle d'architecture *SilkRoad* agit comme un intermédiaire entre les agents négociant réellement (qui peuvent être des agents humains ou logiciels) et ainsi fournit une communication basée sur des règles et un support de décision.

Les deux éléments au coeur de *SilkRoad* (Figure 2.18) sont le `ROADMAP` et le `SKELETON`. Le `SKELETON` fournit plusieurs composants de service de négociation modulables et configurables, qui peuvent être utilisés pour implémenter un médium de négociation électronique. Un médium est une plateforme où l'échange (une transaction) d'objets tangibles ou intangibles est coordonné grâce à l'interaction d'agents. Un médium peut être décrit en terme de trois composants principaux :

- Les canaux de communication : les agents accèdent au médium via des canaux de communication qui peuvent transporter les objets qui seront échangés.
- L'espace logique : la syntaxe et la sémantique définies pour les objets, que les agents échangent.
- L'organisation : les rôles décrivant les types d'agents et les protocoles spécifiant leurs interactions.

Un médium électronique, en particulier, est un médium avec des canaux de communication électroniques (digitaux) qui transportent les données. Cependant, les agents peuvent toujours être des humains ou des unités organisationnelles et ne doivent pas nécessairement être des agents logiciels.

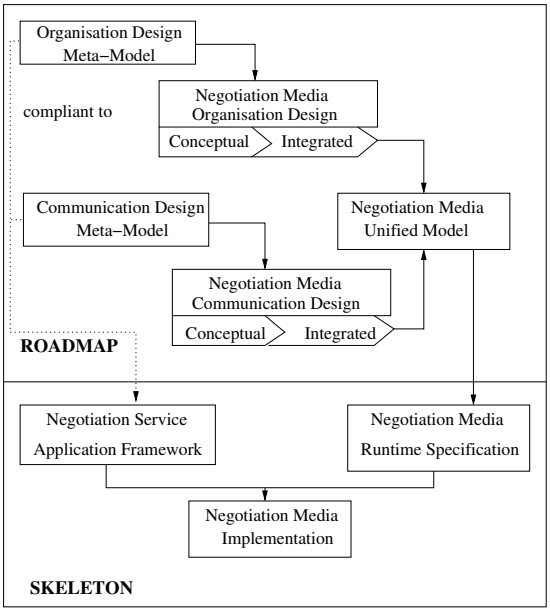


FIG. 2.18 – *Vue générale de SilkRoad (Figure issue de [Ströbel, 2001b])*

Le `SKELETON` peut être vu comme un framework d'applications (le squelette d'un médium de négociation électronique, qui peut être adapté à des scénarios d'accords spécifiques). L'usage le plus commun du framework est le déploiement d'instances des composants du framework, qui sont adaptées au moment de l'exécution sur les bases des spécifications générées dans la phase de conception.

Le processus de conception d'un médium de négociation électronique est structuré selon le modèle d'action de conception `SILKROAD ROADMAP` (Figure 2.19), qui fournit deux éléments principaux :

- Le méta-modèle de conception de l'organisation (ODMM). Ce méta-modèle facilite, mais aussi contraint, le processus de conception de modèles organisationnels concrets de média de négociation. Il permet la spécification des structures (rôles) et du comportement (protocole) des négociations électroniques avec la conception de blocs de construction qui représentent explicitement la fonctionnalité des composants de service de négociation sousjascents au niveau conceptuel.
- Le méta-modèle de conception de la communication (CDMM). Le but de ce méta-modèle est d'assister la conception de l'espace logique d'une négociation électronique. Il fournit le moyen d'exprimer les représentations syntaxique et sémantique des objets et les termes et conditions des transactions du marché.

La Figure 2.20 montre les principaux types d'entités et les relations entre ces types, aussi bien pour le ODMM que pour le CDMM. Le concept de graphes d'états est le paradigme de modélisation sous-jacent pour la conception de l'organisation et de la communication. Il a l'avantage d'être communément utilisé dans la conception de systèmes d'information et fait également partie d'UML.

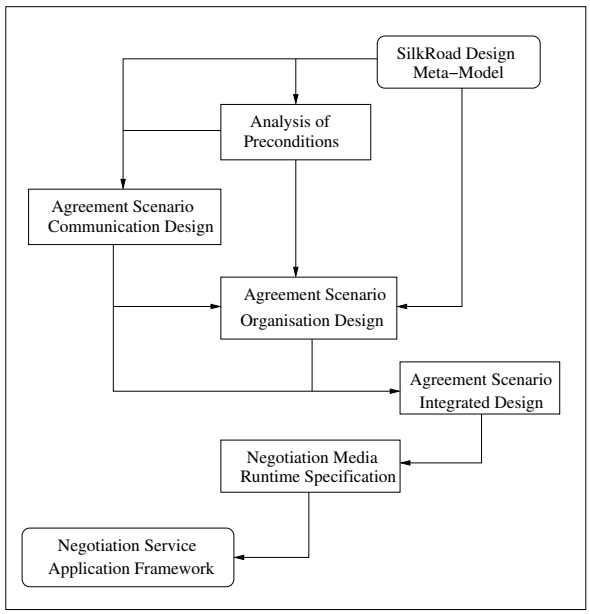


FIG. 2.19 – *L'élément ROADMAP de SilkRoad (Figure issue de [Ströbel, 2001a])*

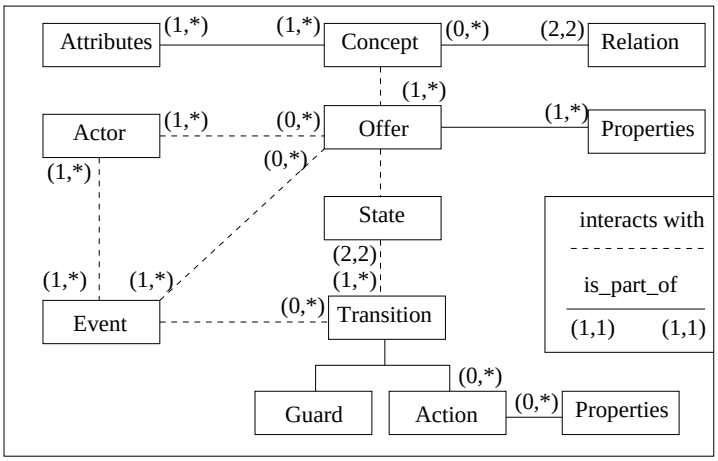


FIG. 2.20 – *Méta-modèle de conception de SilkRoad (Figure issue de [Ströbel, 2001b])*

La tâche de conception de l'organisation est de modéliser les différents états des types d'offres au sein d'une négociation électronique et par là même de découvrir les rôles d'acteur, les évènements, les transitions, les gardes et les actions associés. Cela résulte dans la conception des rôles et du protocole du médium de négociation électronique. Les rôles sont définis comme l'ensemble de tous les évènements possibles qu'un acteur peut déclencher. Un graphe d'entités et de relations constitue un scénario d'accord. Un scénario d'accord représente tous les rôles nécessaires et le protocole pour la spécification complète de la phase d'accord d'une transaction dans un médium de négociation électronique. Le protocole constitue toutes les règles d'un scénario, représenté par des états valides et des transitions, qui définissent comment les agents aboutissent à un accord. Un médium de négociation électronique peut caractériser plusieurs scénarios d'accord qui, au moment de l'exécution, peuvent être instanciés en parallèle.

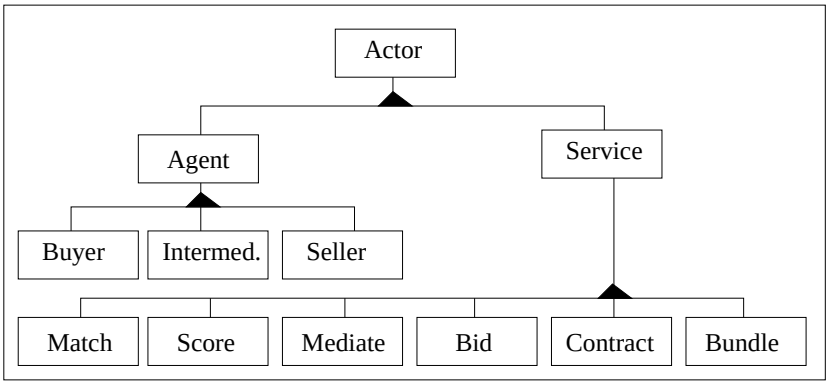


FIG. 2.21 – *Les acteurs dans SilkRoad (Figure issue de [Ströbel, 2001b])*

Les acteurs sont soit un type d'agent, soit un type de service. Un type d'agent peut assumer trois rôles dans un modèle d'organisation de médium de négociation (Figure 2.21) : acheteur (B), vendeur (S) ou intermédiaire (I). Le SKELETON fournit des interfaces pour des agents autonomes aussi bien que pour des agents humains.

Un service est défini comme un système possédant des entrées/sorties qui reçoit des entrées (par exemple, une offre), traite ces entrées en interne et ensuite produit des sorties (par exemple, une valeur d'utilité agrégée pour cette offre). Les types de service d'acteur dans le ODMM représentent les composants de service de négociation suivants dans le framework d'application de *SilkRoad* :

- `Match`. Le service d'appariement teste un ensemble d'offres de vente et d'offres d'achat pour compatibilité. Le résultat est un ensemble de $n >= 0$ paires d'offres compatibles.
- `Score`. Le service d'évaluation reçoit un ensemble d'offres candidates et une offre spécifiant les critères d'évaluation pour calculer un classement pour l'ensemble des offres candidates, dépendant de leurs notes d'évaluation, déterminant ainsi les meilleures offres pour la perspective d'un agent.

- `Mediate`. Pour deux offres non concordantes, ce service suggère un accord dépendant des éléments de l'offre mutuellement négociables et des préférences spécifiques à la situation des agents pour ces éléments négociables.
- `Bid`. Ce service persistant publie des offres, si elles obéissent à certaines règles d'enchère, dans une session de négociation, jusqu'à ce qu'une règle de terminaison s'applique et que l'offre gagnante puisse être déterminé.
- `Contract`. Ce service demande aux agents de signer les offres, vérifie les signatures et les autorisations et scelle les offres.
- `Bundle`. Ce service permet de collecter ou d'agréger des offres afin de former des offres cumulatives. Ainsi des agents acheteurs peuvent se regrouper afin d'accroître leur pouvoir de négociation, ou alors des offres complémentaires fournies par différents agents vendeurs peuvent être regroupées (transport de A vers B, B vers C, et C vers D)(négociation combinée).

Dans un scénario d'accord, ces services peuvent être combinés pour fournir un niveau de scénario spécifique d'un support de négociation.

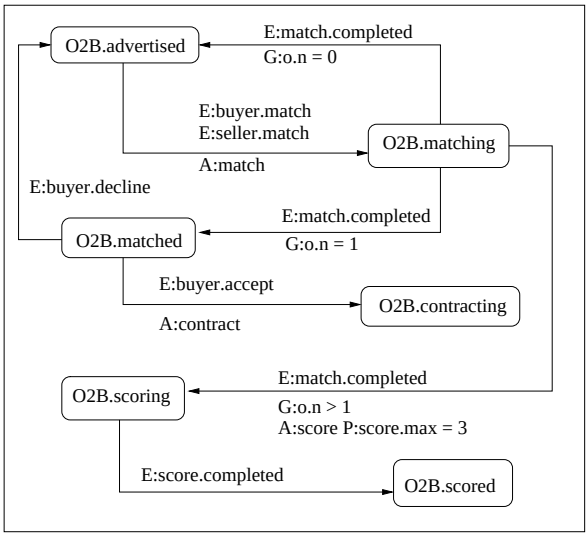


FIG. 2.22 – *Exemple de scénario d'accord fournissant un appariement et une évaluation des offres (Figure issue de [Ströbel, 2001b])*

Les Figures 2.22 et 2.23 montrent deux exemples de scénario de négociation modélisés avec les éléments du ODMM. La notation graphique suit les conventions UML pour les diagrammes d'états. Les états sont représentés par des rectangles aux bords arrondis. Le type d'offre relatif à un état est indiqué en lettres capitales devant l'identifiant d'état. Les transitions sont les flèches connectant les états et les évènements ('E :'), les gardes ('G :'), les actions ('A') et les propriétés ('P :') sont spécifiés comme des informations textuelles complétant les flèches de transition.

La Figure 2.22 illustre un exemple de scénario d'accord qui fournit un appariement

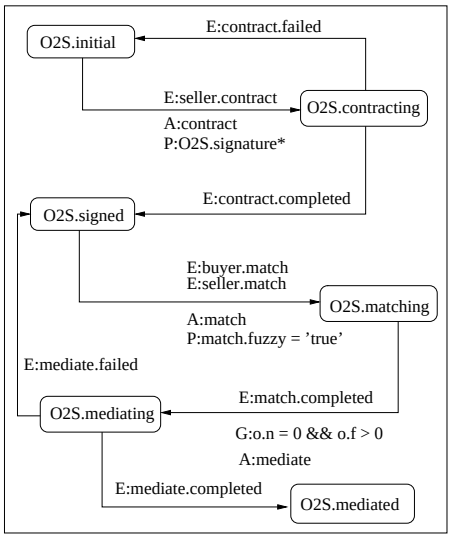


FIG. 2.23 – *Exemple d'utilisation des services de contractualisation et de médiation (Figure issue de [Ströbel, 2001b])*

basique des offres des acheteurs/vendeurs et classe les offres dans le cas où plusieurs d'entre elles correspondent à une requête. Dans cet exemple, un acteur de type acheteur déclenche un évènement d'*appariement* (`buyer.match`) pour une offre d'achat (`O2B`), ce qui active la transition de l'état *offre d'achat publiée* (`O2B.advertised`) à l'état service *appariement* (`O2B.matching`). L'action *appariement* (`match`) est associée à cette transition. Si l'ensemble des offres concordantes associé à l'évènement *appariement effectué* (`match.completed`) contient une seule offre, alors une transition vers l'état *offre d'achat appariée* (`O2B.matched`) a lieu. Dans cet état, seul l'acheteur peut déclencher des évènements (*accepter* : `buyer.accept` /*refuser* : `buyer.decline`) qui activent des transitions, par conséquent, l'exécution du framework s'arrête et attend qu'un agent déclenche un évènement (`agent.event`). Si l'acheteur choisit l'option d'*accepter* (`buyer.accept`), l'action *contractualiser* (`contract`) est invoquée et l'offre passe dans l'état *en cours de contractualisation* (`O2B.contracting`). Alternativement, lors du traitement de l'évènement *appariement effectué* (`match.completed`), la garde de condition $n > 1$ peut être évaluée à *vrai*. Dans ce cas, l'offre d'achat passe à l'état d'*évaluation* (`O2B.scoring`) avec l'action *évaluer* (`score`) et la propriété *score maximum égal 3* (`score.max = 3`). L'évènement *évaluation terminée* (`score.completed`) résulte ensuite en l'état *offre évaluée* (`O2B.scored`), où l'acheteur a de nouveau l'option d'*accepter* l'une des offres évaluées.

Le projet *SilkRoad* se concentre sur les négociations commerciales où sont échangées des offres de vente et des offres d'achat. Bien que le scénario d'une négociation puisse être spécifié par l'utilisateur, celui-ci reste relatif aux offres et il n'est pas possible de définir ses propres actions, seules celles fournies par le `SKELETON` peuvent être utilisées. La généricité de cette plateforme est donc relative, puisqu'elle est cantonnée

aux fonctionnalités fournies par celle-ci. Notre approche de la négociation et plus particulièrement d'une plateforme générique de négociation est plus ouverte. Pour nous, il doit être possible d'effectuer des négociations commerciales sous différentes formes (enchères hollandaises, anglaises, etc.) aussi bien que des négociations non commerciales (prise de rendez-vous, choix d'un restaurant, etc.). Dans *SilkRoad*, il n'y a pas, a priori, de communication entre les agents : un serveur regroupe les offres et s'occupe de les apparier. Notre approche est de fournir aux agents un cadre de négociation au sein duquel ils peuvent s'échanger des propositions de contrat. Chaque agent gère donc ses négociations et il est possible de définir de nouvelles stratégies de négociation.

2.3.10 *GNP*

La Generic Negotiation Platform (*GNP*) est réalisée par Morad Benyoucef, Rudolf Keller et leurs collègues ([Benyoucef et al., 2000, Gerin-Lajoie, 2000]) au département IRO de l'université de Montréal. C'est une plateforme générique de négociation d'entreprises à entreprises orientée enchères pour les places de marché. La négociation entre les agents s'effectue via le service de négociation qui s'occupe de l'appariement entre la demande des acheteurs, l'offre en bien et services des vendeurs et la fixation des prix. Cette plateforme reçoit les annonces des acheteurs ou des vendeurs et les mises qui répondent à ces annonces. Les règles de marché sont définies dans des fichiers XML.

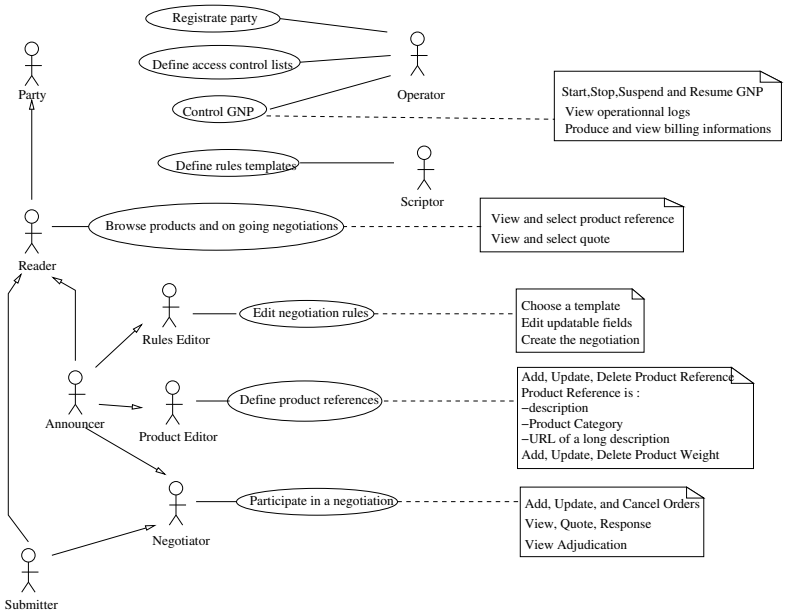


FIG. 2.24 – *Scénario d'utilisation de GNP (Figure issue de [Gerin-Lajoie, 2000])*

La Figure 2.24 illustre le scénario d'utilisation principal, les acteurs et leurs rôles dans le processus de négociation.

Lecteur - "Reader" Tous les participants peuvent a priori être des lecteurs. Ils peuvent consulter les produits, les annonces d'achats et de vente, voir les cotes publiques et finalement sélectionner celles sur lesquelles ils vont réagir et soumissionner.

Annonceur - "Announcer" Un annonceur crée une négociation. Il annonce un intérêt d'achat ou de vente d'un ou plusieurs produits ou services. Les 3 étapes de la création d'une négociation sont :
- Créer les règles de la négociation à partir de modèles existants. Fixer les paramètres éditables comme l'heure de fermeture ;
- Créer les références aux objets de la négociation, produits ou services ;
- Créer un ordre initial d'achat ou de vente, définissant le prix et la quantité de départ.

Négociateur et soumissionnaire - "Negotiator and Submitter" Un négociateur émet les ordres d'achats ou de vente. Un acheteur et un vendeur peuvent donc être un négociateur. Le négociateur émet l'ordre initial d'achat ou de vente. Le soumissionnaire est un négociateur répondant à une cote avec un ordre d'achat ou de vente.

Administrateur et opérateur - "Administrator and Operator" Les administrateurs et opérateurs gèrent les participants et leurs priorités, les marchés et les négociations. Sur les marchés et les négociations, ils peuvent non seulement les ajouter, modifier et enlever, mais aussi les démarrer, suspendre et repartir.

Déroulement d'une négociation L'initialisation d'une négociation sur *GNP* se fait par l'envoi d'un document XML initial : l'annonce, qui contient tous les éléments d'entrée nécessaires au démarrage d'une négociation.

Le participant désirant initialiser une négociation sur *GNP* dispose en premier lieu d'un choix de différents modèles de négociation (template). À un modèle correspond un formulaire d'annonce HTML unique. Le formulaire permet à l'usager de définir certains paramètres du modèle et ainsi de garantir une flexibilité d'implantation à l'intérieur de chaque modèle. Le formulaire complété et envoyé donnera naissance à l'annonce, document au format XML.

Il est également possible d'envoyer directement le document d'une annonce à *GNP*. Dans ce cas, le document de l'annonce doit être entièrement édité manuellement dans le format XML valide et attendu, dans un fichier.

L'annonce donne à *GNP* tous les éléments d'entrée nécessaires au démarrage d'une négociation. L'annonce apparaît à l'utilisateur dans un format HTML. Elle contient toujours une partie *règles* (rules) et, selon le mécanisme choisi, une partie *produit* (productReference) et une partie *ordre* (order). La partie *règles* établit précisément les paramètres

fixes de la négociation, telles que les heures d'ouvertures et de fermetures, l'incrément minimum, les conditions de fin de tours et les conditions d'équilibrage du marché. Si ceux-ci doivent varier, la négociation est divisée en plusieurs phases. La partie *produit* définit techniquement la nature du produit à négocier, renvoie à sa description précise. La partie *ordre* représente les données de la mise initiale faite par le participant, telles que son prix de départ, son prix de réserve, la quantité de produit mise en négociation et ses fractions permises. Lorsque l'annonce est remplie, elle est envoyée au système qui va déclencher une session d'annonce.

Les utilisateurs interagissent avec *GNP* uniquement via un navigateur web. Aucune installation de logiciel ne doit être effectuée.

D'un point de vue formel, les auteurs clament dans [Benyoucef et al., 2000] :

> *"we have identified a number of operations that are common to different negotiation processes. Some of these operations are :*
> – *define attributes and default values for the formalized concepts ;*
> – *setup the end conditions for rounds, phases and the whole negotiation ;*
> – *define the information to be displayed to or hidden from the players."*

Comme nous le montrerons par la suite avec nos propres travaux, il est possible d'identifier des points communs aux différents types de négociation et d'en tirer parti pour offrir une plateforme générique de négociation. Les auteurs pensent également qu'il faut séparer le processus de négociation des autres parties du logiciel et que les règles gouvernant la négociation ne devraient pas être codées en dur. Nous sommes donc très proches de ces travaux sur le *GNP*, mais notre plateforme se veut plus générique encore, car nous ne nous sommes pas préoccupés uniquement des marchés et des ventes aux enchères, mais aussi aux négociations non commerciales telles que la prise de rendez-vous, par exemple. Nous proposons également différents modes de communications entre les agents (SMA, e-mail, etc.) alors que *GNP* ne peut s'utiliser que par un navigateur web. Notre protocole de négociation est également plus large, puisqu'il permet de renégocier automatiquement les contrats qui doivent être déplacés. Un avantage du *GNP* est qu'il propose différents modèles de négociation, ce qui permet d'instancier facilement les différents types de négociation prédéfinis. C'est cet apport de bibliothèque de règles (protocoles) que nous voulons atteindre avec *GeNCA*.

2.3.11 A generic software framework for automated negotiation

Claudio Bartolini et ses collègues ([Bartolini and Preist, 2001, Bartolini et al., 2002b, Bartolini et al., 2002a]), des laboratoires HP, veulent créer 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 différents ensembles 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 utilisé 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 par un message destiné à l'hôte de négociation. Celui-ci détermine quels participants doivent voir le message et le leur envoie. Cela permet de modéliser la négociation de 1 vers 1 comme un cas particulier de la négociation de n vers m.

Afin de négocier, les participants doivent initialement partager un *template* de négociation. Il spécifie les différents paramètres de négociation qui peuvent être soit contraints, soit totalement ouverts. Une scène de négociation est associée à un *template* de négociation, ce qui définit l'objet de la négociation au sein de cette scène. Le processus d'*admission* à la négociation comprend entre autres l'acceptation du template, plus éventuellement d'autres critères possibles comme la provision d'un crédit suffisant pour participer à la négociation.

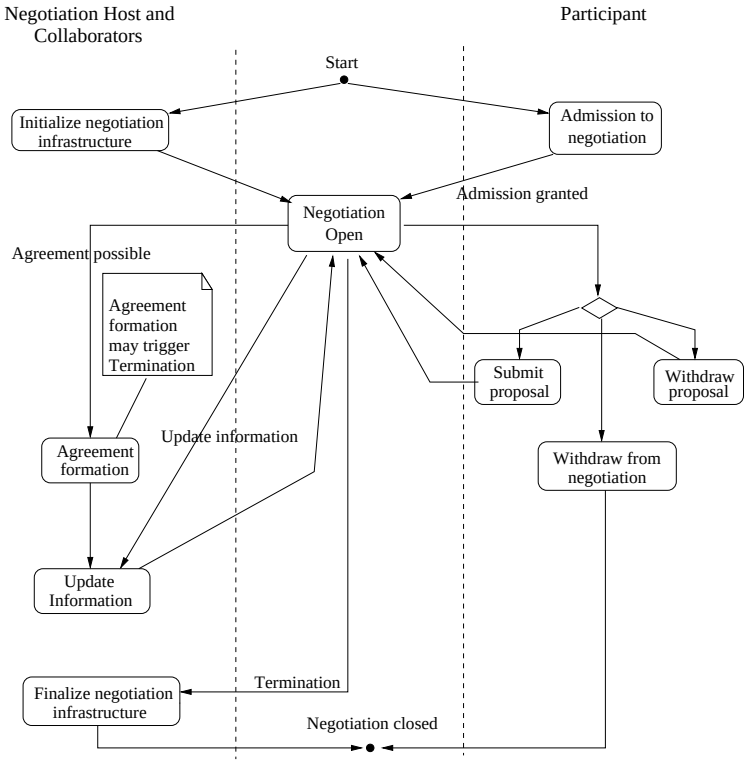


FIG. 2.25 – *Diagramme d'activité de négociation (Figure issue de [Bartolini et al., 2002b])*

Le processus de négociation (Figure 2.25) consiste à parvenir à un accord acceptable depuis le template de négociation. Au cours de la négociation, les participants échangent des *propositions* représentant des accords actuellement acceptables pour eux. Chaque proposition contient des contraintes sur tout ou partie des paramètres du tem-

plate de négociation. Ces propositions sont envoyées à l'hôte de négociation. Cependant, avant qu'une proposition soit acceptée dans la scène, elle doit être validée. Pour être *valide*, une proposition doit satisfaire deux critères :

- Ce doit être une restriction valide de l'espace des paramètres défini dans le template de négociation.
- La proposition doit être soumise conformément à l'ensemble des règles qui gouvernent le déroulement de la négociation. Ces règles spécifient entre autres qui peut faire une proposition, quand et quelle proposition peut être faite selon les propositions antérieures.

Un *accord* est formé conformément aux règles de formation d'accord associées à la scène de négociation. Lorsque les propositions dans la scène satisfont certaines conditions, elles sont converties par ces règles en accords et retournées aux participants. La fin de la négociation est déterminée par des règles de terminaison.

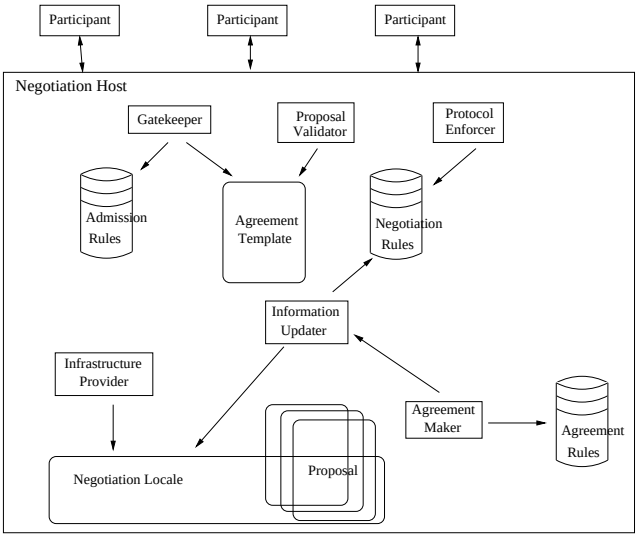


FIG. 2.26 – *Architecture abstraite : sous-rôles et relations (Figure issue de [Bartolini et al., 2002b])*

Il y a deux rôles principaux dans la négociation : *participant* et *hôte* (Figure 2.26). Les participants sont ceux qui veulent aboutir à un accord. L'hôte est chargé de faire respecter le protocole et les règles de négociation. L'hôte de la négociation peut ou non être également un participant. Le participant peut poster des propositions selon les règles fournies par l'hôte de négociation. L'hôte est chargé de la création et de faire respecter les règles gouvernant la participation, l'exécution, la résolution et la terminaison de la négociation. 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 selon ce qui est spécifié dans la règle de terminaison.

Ce framework a été réalisé en utilisant la plateforme Jade, les agents communiquent par envoi de messages au format FIPA ACL et Jess a été utilisé pour le traitement des règles. Les auteurs travaillent à utiliser DAML+OIL au lieu de Jess pour avoir une implémentation plus ouverte et indépendante de la plateforme et à incorporer d'autres formes de négociation telles que les accords multi-parties, le *Contract Net* et la négociation par argumentation. Ce framework est proche du notre par son protocole générique qui est paramétrable ici par des règles et qui fournit un mécanisme de contre-propositions. Cependant, notre protocole permet en plus de se rétracter d'un contrat et de le renégocier automatiquement. De plus, nous ne sommes pas liés à une plateforme multi-agents car nous avons séparé notre couche de communication afin de pouvoir s'adapter à différents moyens de communication comme par exemple l'envoi d'e-mails.

2.4 Conclusion

Nous avons présenté dans ce chapitre la définition de la négociation que nous considérons pour cette thèse, ainsi que différents types de négociation que nous avons étudié et différents systèmes de négociation réalisés par d'autres équipes de recherche. La plupart des systèmes présentés dans ce chapitre utilisent des protocoles propres à leur application. Parmi ces systèmes, certains sont spécialisés dans les enchères comme *Fishmarket*, *Kasbah*, *GNP* et *AuctionBot*, d'autres sur les services comme *ADEPT*, mais peu se présentent comme étant générique. De plus, ils n'offrent pas tous les mêmes possibilités de communication, seul *ADEPT* sépare la communication, pour les autres, elle est figée comme par exemple *SilkRoad* et *GNP* qui communiquent via une interface web. Certains utilisant un modèle bilatéral de négociation, par exemple *Kasbah*et *Zeus*, d'autres un modèle de 1 vers n agents, par exemple *Magnet* et *Traconet*. Enfin, ils utilisent des actes de langages spécifiques, ce qui limite leur portabilité. Le tableau 2.2 reprend les différentes plateformes présentées ainsi que *GeNCA* et apporte une comparaison sur différents critères qui nous paraissent importants pour une plateforme de négociation générique. Ce tableau laisse apparaître que *GeNCA* est la seule plateforme possèdant toutes les caractéristiques que nous avons définies et la seule à proposer de choisir la réponse par défaut pour une proposition.

Nous allons maintenant présenter notre modèle de négociation qui réifie les différents types de négociation présentés, qui permet de gérer plusieurs négociations simultanées entre 1 et n agents et qui est applicable à différentes applications telles la prise de rendez-vous, la vente aux enchères ou encore les places de marché.

Plateforme	cardinalité	généricité	rétraction	renégociation automatique	simultanéité
GeNCA	$n \to m$	oui	oui	oui	oui
GNP	$1 \to 1$	non	non	non	oui
SilkRoad	$1 \to 1$	non	?	?	oui
Magnet	$1 \to m$	oui	oui	non	oui
HP	$n \to m$	oui	non	non	oui
ADEPT	$1 \to 1$	non	?	oui	oui
Zeus	$1 \to 1$	non	non	non	oui
Kasbah	$1 \to 1$	non	non	non	oui
AuctionBot	$1 \to 1$	non	non	non	oui
Fishmarket	$1 \to m$	non	non	non	non
Traconet	$1 \to m$	non	?	?	oui

Plateforme	séparation de la communication	contre-propositions	réponse par défaut	paramétrage possible
GeNCA	oui	oui	au choix	oui
GNP	non	oui	refus implicite	oui
SilkRoad	non	oui	refus implicite	non
Magnet	non	non	refus implicite	non
HP	non	oui	non	oui
ADEPT	oui	oui	non	non
Zeus	non	non	non	non
Kasbah	non	oui	refus implicite	non
AuctionBot	non	non	refus implicite	non
Fishmarket	non	non	refus implicite	non
Traconet	?	non	refus implicite	non

TAB. 2.2 – *Tableau comparatif des plates-formes de négociation. Le ? signifie qu'aucune donnée ne nous permet d'affirmer ou d'infirmer la propriété énoncée.*

Chapitre 3

Proposition d'un modèle général de négociation

Nous avons présenté dans le chapitre précédent les différentes formes de négociation existantes ainsi que différentes plateformes de négociation réalisées parmi les plus connues. Nous présentons ici une réification de ces formes de négociation ainsi que notre modèle de négociation général, qui permet de traiter la plupart de ces négociations. Les apports de ce modèle sont la possibilité de négocier simultanément plusieurs contrats, de renégocier automatiquement les contrats qui doivent l'être et l'élimination des deadlocks. Il possède en outre les qualités de généricité, d'uniformisation des négociations et d'automatisation des envois de messages.

3.1 Réification de ces formes de négociation

Parmi les types de négociation présentés dans le chapitre précédent, on peut dégager un certain nombre de points communs qui permettent de les réifier. Le premier point commun évident à toutes ces formes de négociations est l'implication de ressources (ce qui va être négocié) et de personnes (celles qui vont négocier). Nous décrivons ensuite un ensemble minimal d'actes de langage nécessaires pour formaliser une négociation.

3.1.1 Les ressources

Les ressources sont les objets de la négociation. Pour le cas des enchères, ce seront des articles comme des tables, des chaises, etc., dans le cas de la prise de rendez-vous,

ce seront des créneaux horaires. Les ressources qui seront négociées peuvent être soit personnelles, soit communes. Lorsqu'elles sont communes, n'importe qui peut entamer une négociation les impliquant (cas des rendez-vous). Lorsqu'elles sont individuelles, seule la personne qui les possède peut engager une négociation qui les implique (cas des enchères). Des préférences sur les ressources peuvent être utiles aux personnes afin de définir quelles sont les ressources qu'elles désirent vivement obtenir et quelles sont celles qui lui sont indifférentes. Par exemple, dans le cas de la prise de rendez-vous, une personne peut préférer prendre ses rendez-vous entre 10 heures et midi ou 14 heures et 17 heures. Les priorités sont indiquées sur une échelle de 1 à 10, où 10 est la valeur la plus prioritaire. Afin de négocier l'obtention de ressources, nous allons les placer dans un contrat qui sera négocié. Ce contrat pourra également contenir d'autres attributs, comme un prix pour les enchères anglaises ou hollandaises, ou des arguments pour la négociation à base d'argumentation.

3.1.2 Les personnes

Dans chacun des modèles présentés, une personne formule une proposition à une ou plusieurs autres personnes. Nous appelons cette personne l'*initiateur* de la négociation et les personnes à qui la proposition est faite les *participants* à la négociation. Bien entendu, dans la vie de tous les jours, nous ne négocions pas de la même façon avec toutes les personnes. Par exemple, on accepte plus facilement les propositions de ses supérieurs que de ses subalternes. Afin de garder cette notion de préférence entre les personnes, on utilise une liste de priorité que l'utilisateur peut modifier à tout moment. Ainsi, chaque personne peut définir une priorité pour chaque autre personne et peut donc établir une stratégie de négociation basée sur ces priorités.

3.1.3 Un ensemble minimal d'actes de langage pour la négociation

Nous définissons ici un ensemble minimal d'actes de langages nécessaires à toutes les négociations décrites précédemment. L'ensemble de ces actes de langage avec leur séquencement forment le protocole de négociation qui permet de réaliser les négociations décrites précédemment. Chacune des négociations commence par la proposition d'un contrat, un premier acte de langage sera donc l'envoi de la `proposition` de l'initiateur aux participants. Puis chaque participant répond à l'initiateur, soit en acceptant ou en refusant la proposition, soit en raffinant la proposition ou en formulant une contre-proposition. Il faut donc au moins deux autres actes de langage constituants la réponse (`refus` ou `acceptation` plus ou moins forte) d'un participant à la proposition de l'initiateur. Dans le cas de l'acceptation, on peut inclure des paramètres permettant de donner une contre-proposition ou de raffiner la proposition, ce qui nous permet d'avoir exactement deux actes de langage. À ce stade, selon le nombre de tours dans la négociation, l'initiateur doit décider de la réussite ou de l'échec de la négociation, ou

alors de poursuivre la négociation. Trois nouveaux actes de langages doivent donc être mis en place : un pour faire savoir aux participants que la négociation a abouti sur un `succès`, un autre pour le cas de l'`échec` et un troisième pour la poursuite de la négociation. Pour poursuivre la négociation, l'initiateur peut demander aux participants de modifier la proposition pour qu'elle soit plus acceptable pour eux et synthétiser les résultats afin de formuler une nouvelle proposition. Deux actes de langages sont donc nécessaires : la `demande de modification` de l'initiateur et la `proposition de modification` des participants. Ces actes de langages sont nécessaires pour améliorer et accélérer le processus de négociation. En effet, si l'on ne disposait que de la proposition, de l'acceptation et du refus, l'initiateur ne pourrait continuer la négociation qu'en proposant un nouveau contrat aux participants. Ce nouveau contrat serait choisi par l'initiateur sans qu'il ne sache s'il a des chances de mieux convenir aux participants. Il peut alors s'écouler beaucoup de temps avant que le énième nouveau contrat proposé ne soit accepté. Tandis qu'avec le mécanisme de contre-propositions, l'initiateur reçoit des informations des participants qui l'aident à trouver un nouveau contrat qui sera plus satisfaisant pour les participants et ce plus rapidement.

Il peut aussi être utile de posséder un acte de langage pour se `rétracter` d'un contrat précédemment accepté. Il est en effet possible qu'un participant ne veuille plus ou ne puisse plus être en mesure d'honorer le contrat qu'il a pris. Pour certaines négociations comme les enchères, cette rétractation est impossible.

Nous considérons que l'ensemble *{propose, accepte, refuse, demande de modification, proposition de modification, confirmation, annulation, rétractation}* contient les actes de langage nécessaires pour la négociation et que cet ensemble est minimal. Nous étendons donc l'ensemble minimal proposé par Jennings dans [Beer et al., 1999], qui contenait les actes *{propose, counter-propose, accept, reject}*. Nous pensons en effet qu'il faut deux actes de langage supplémentaires pour clore la négociation, une pour le cas du succès de celle-ci, l'autre en cas d'échec.

Nous ajoutons également un acte pour demander des contre-propositions de la part des participants car nous croyons que l'initiateur a de meilleures chances de proposer un nouveau contrat qui sera accepté s'il recueille auparavant des contre-propositions de la part des participants et en effectue une synthèse.

Enfin, nous parons à l'éventualité de la rétractation d'un participant une fois la négociation terminée avec succès, ce qui conduit souvent à la renégociation du contrat.

Nous définissons nos propres actes de langage pour la négociation car les standards KQML et FIPA-ACL ne sont pas adaptés au modèle que nous souhaitons proposer. En première remarque, KQML et FIPA-ACL sont des langages de communication entre agents. Ils ont donc un but plus large que la négociation entre agents. Nous nous intéressons pour notre part uniquement à la négociation entre agents. Nous avons donc besoin de primitives de communication plus spécifiques que celles incluses dans FIPA-ACL et KQML. De plus, nous pouvons constater que les actes de langage que nous

avons définis ne font pas partie des actes de langage prévus par défaut dans KQML. Il faudrait donc étendre KQML pour y inclure nos primitives. Un autre obstacle à l'utilisation de KQML est que dans sa spécification, il est indiqué que la délivrance des messages doit être fiable et que si deux messages sont envoyés au même agent par un autre agent, alors ces messages doivent arriver dans le même ordre qu'ils ont été envoyés. Comme nous prévoyons de pouvoir utiliser différents moyens de communication, il n'est pas possible d'assurer ces propriétés. En effet, si l'on utilise la communication par envoi d'e-mails, on ne peut pas garantir l'arrivée des messages, ni être sûrs de la réception des e-mails dans l'ordre d'envoi de ceux-ci.

De plus, KQML et FIPA-ACL sont basés sur la communication entre des agents possédant des bases de croyances. Les messages échangés entre les agents concernent les propriétés contenues dans leur base de croyances. Par exemple, dans FIPA-ACL, la primitive *tell(a)* est utilisée par un agent pour indiquer au destinataire que l'expression *a* est vraie. Dans la spécification de FIPA-ACL, il y a des préconditions sur les croyances de l'agent qui déterminent la possibilité d'utiliser tel ou tel acte de langage. Notre modèle de négociation ne présuppose pas que les agents possèdent une base de croyances, ni que ce sont des agents BDI. Nous ne pouvons donc pas garantir la bonne utilisation des actes définis dans FIPA-ACL au sens où il ne serait pas possible de vérifier les préconditions requises pour l'utilisation de ces actes de langage. Certains actes de langage que nous avons définis ne font également pas partie des actes définis dans FIPA-ACL.

Ces différents points rendent difficile l'utilisation de KQML ou de FIPA-ACL dans notre modèle. C'est pourquoi nous avons défini nos propres actes de langages qui sont en fait des actes de négociation.

3.1.4 Le modèle général de négociation

Le besoin d'un modèle général de négociation provient de la richesse de types de négociation différents. Comme nous l'avons montré dans la section précédente, certaines notions communes aux différentes négociations peuvent être réifiées au sein d'un modèle général. Cependant, il n'existe pas de framework générique de négociation reconnu par tous les chercheurs du domaine. Notre objectif est donc de fournir ce framework, offrant ainsi à un concepteur d'applications de négociation un outil facilitant la spécification de son application.

Nos objectifs portent principalement sur quatre dimensions : la généricité, la portabilité, l'uniformisation et l'automatisation des envois de messages.

La portabilité est une caractéristique importante pour un modèle général, quel qu'il soit. En effet, un modèle qui ne pourrait pas s'appliquer dans différents contextes perdrait en exploitabilité.

L'uniformisation est nécessaire pour définir un cadre de travail commun pour tous les chercheurs du domaine. C'est un besoin très fort, pour que chacun puisse se comprendre et qu'il n'y ait pas de mauvaise interprétation, de malentendu. À notre connaissance, peu de travaux d'uniformisation existent. Nous pouvons cependant citer la *London Classification* décrite dans [Benyoucef, 2000], qui propose un système de classification des négociations selon quatre aspects : `personnes`, `biens`, `processus` et `critères d'évaluation`. L'aspect `personnes` comprend entre autres le nombre de personnes impliquées et l'anonymat, l'aspect `biens` contient le nombre d'items du bien sujet à la négociation, le nombre d'attributs dans la négociation (prix, quantité, qualité de service sont des attributs possibles). L'aspect `processus` caractérise le protocole utilisé pour la négociation, en termes de nombre de phases, de nombre d'étapes, de synchronisation, parmi d'autres critères. L'aspect `critères d'évaluation` comprend notamment la complexité computationnelle, la convergence, l'efficacité, l'équité. La *Montreal Taxonomy* [Ströbel and Weinhardt, 2003, Neumann et al., 2003] affine la *London Classification* qui ne prend pas en compte l'aspect `règles` de négociation.

La complexité en envois de messages est une notion importante qui démontre l'utilité de l'automatisation de la négociation. En effet, lorsqu'un nombre exponentiel de messages doit être envoyé, traité, mieux vaut utiliser un agent autonome que de s'en occuper soi-même afin de gagner du temps libre.

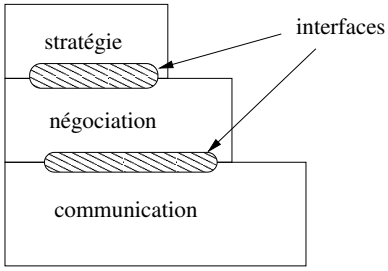


FIG. 3.1 – *Les trois niveaux de notre modèle général. Le niveau de négociation forme le cœur du modèle. Il possède une interface avec le niveau de communication qui se charge de faire communiquer les agents entre eux selon le mécanisme fourni par l'application. Il possède également une interface avec le niveau stratégique qui définit la façon dont l'agent négociera ses contrats, ce qui est également spécifique à l'application réalisée.*

Pour concevoir notre modèle et permettre une réelle généricité, nous avons choisi de nous appuyer sur une architecture à trois niveaux (Figure 3.1). Le niveau de négociation qui contient la gestion des structures de données et les actes de langage nécessaires aux agents pour faire évoluer leur connaissance ; le niveau de communication qui permet aux agents d'envoyer des messages de façon centralisée s'ils sont sur le même ordinateur, ou de façon distribuée s'ils sont sur des ordinateurs différents ; le niveau stratégique qui permet aux agents de raisonner sur le problème et d'inférer leurs décisions en fonction de la connaissance obtenue des autres agents négociateurs. L'intérêt d'une telle

décomposition est que chaque niveau peut être changé indépendamment des autres à la manière de briques logicielles. Il est par exemple possible d'utiliser *GeNCA* en round-robin avec des communications synchrones avec tous les agents sur le même ordinateur pour réaliser un jeu vidéo dans lequel les individus virtuels négocieront tour à tour, ou de l'utiliser de façon distribuée avec des communications asynchrones pour des places de marché électroniques avec les agents sur des machines différentes. Nous présentons dans le chapitre 5 différentes applications utilisant ces différents cas d'utilisation.

Nous décrivons dans les sections suivantes ces trois niveaux, ainsi que la complexité en nombre de messages induite par le protocole que nous proposons.

3.2 Le niveau communication

Ce niveau est responsable de la communication entre les agents et définit les primitives de communication que tous les agents doivent comprendre pour utiliser *GeNCA*.

Notre modèle utilise un mécanisme d'abonnement au système afin de localiser les participants et de collecter les ressources qui seront négociées. Un serveur de noms a été défini pour retenir les participants et leur adresse ainsi que les ressources. Chaque agent désirant participer au système de négociation doit s'inscrire auprès du serveur de noms en donnant son nom, son adresse, le type d'application de négociation qu'il veut rejoindre (par exemple, des enchères hollandaises) et ses propres ressources qu'il veut négocier. Après cette inscription, l'agent reçoit la liste des participants déjà présents dans le système pour ce type d'application ainsi que l'ensemble des ressources pouvant être négociées. Chaque participant déjà présent dans le système pour ce type d'application reçoit un message lui indiquant l'arrivée d'un nouvel abonné et les ressources qu'il a apportées. Chaque personne connaît donc toutes les autres personnes et toutes les ressources présentes pour le type d'application auquel il participe.

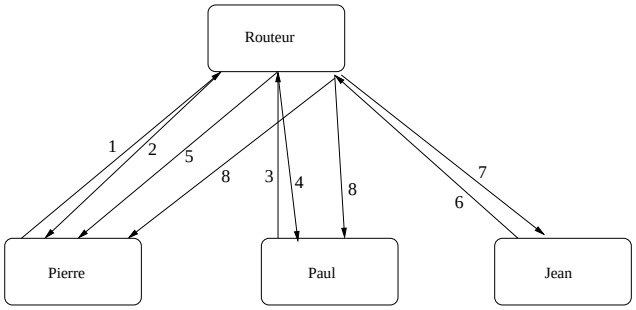


FIG. 3.2 – *Mécanisme d'abonnement. Pierre s'inscrit en premier, puis Paul et ensuite Jean. La figure montre les échanges de messages réalisés à cette occasion.*

La Figure 3.2 décrit le mécanisme d'abonnement de 3 personnes : Pierre, Paul et Jean, quel que soit le moyen de communication utilisé (envoi d'e-mail, utilisation au sein d'un SMA ou encore agents centralisés).

1. Pierre s'abonne auprès du routeur. Il envoie son nom, son adresse, ses ressources et le type d'application auquel il souhaite participer.
2. Le routeur envoie à Pierre la liste des ressources et des participants déjà enregistrés. Dans cet exemple, la liste des participants est vide car Pierre est le premier à s'abonner et la liste des ressources contient uniquement les ressources de Pierre.
3. Paul s'abonne à son tour. Il envoie son nom, son adresse, ses ressources et le type d'application auquel il souhaite participer.
4. Le routeur lui envoie la liste des ressources et des participants déjà enregistrés. A ce moment, la liste des participants contient uniquement Pierre et la liste des ressources l'union des ressources de Pierre et de Paul.
5. Le routeur signale à Pierre l'arrivée d'un nouveau participant : Paul et la liste des ressources que Paul amène dans le système.
6. Jean s'abonne à son tour. Il envoie son nom, son adresse, ses ressources et le type d'application auquel il souhaite participer.
7. Le routeur lui envoie la liste des ressources et des participants déjà enregistrés. Cette fois, la liste des participants contient Pierre et Paul et la liste des ressources l'union des ressources de Pierre, Paul et Jean.
8. Le routeur signale à Pierre et à Paul l'arrivée d'un nouveau participant : Jean et la liste des ressources que Jean amène dans le système.

Chaque agent peut se connecter et se déconnecter du système lorsqu'il le désire, en l'indiquant au serveur de noms. Lorsqu'un agent est déconnecté, le serveur de noms stocke tous les messages à son intention dans sa boîte aux lettres créée à son inscription. L'agent reçoit tous les messages de sa boîte aux lettres lors de sa reconnexion au système.

Chaque agent désirant envoyer un message à un ensemble de participants contacte le serveur de noms qui se charge d'envoyer le message. En effet, les participants connaissent seulement le nom des autres participants, pas leur adresse. Ils doivent donc contacter le serveur de noms pour envoyer leurs messages car il est le seul à connaître les adresses des participants. La Figure 3.3 représente le mécanisme d'envoi d'un message de Pierre à Paul et à Jean alors que Jean est déconnecté. Pierre envoie au routeur le message *msg* et la liste des destinataires, *Paul et Jean*. Le routeur consulte la liste des destinataires et envoie le message à Paul qui est connecté et place le message pour Jean dans sa boîte aux lettres comme il est déconnecté. Lorsque Jean se connecte à nouveau, le routeur lui envoie le message de Pierre et vide sa boîte aux lettres.

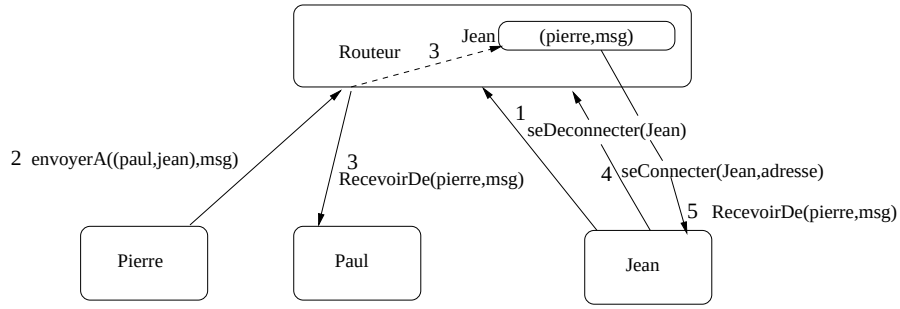


FIG. 3.3 – *Envoi d'un message vers plusieurs destinataires. Si l'un d'eux est déconnecté, le message est stocké dans sa boîte aux lettres et lui est transmis lors de sa reconnexion.*

La séparation de la couche de communication permet d'utiliser différents moyens de communication tels que ceux fournis au sein des plateformes multi-agents, ou encore l'envoi d'e-mails. Ceci permet à notre modèle d'être intégré dans des applications existantes simplement en définissant le mécanisme de communication utilisé. Ces différentes implémentations possibles seront présentées dans le prochain chapitre.

3.3 Le niveau négociation

Le niveau de négociation est le niveau principal de notre modèle, celui qui contient le coeur de notre modèle général : le protocole de négociation ainsi que la gestion des négociations que nous avons établie. C'est sur ce niveau que s'est concentré mon travail de thèse. Nous présentons dans cette section tout d'abord le protocole de négociation, puis la gestion des négociations que nous avons mise en place.

3.3.1 Le protocole général de négociation

L'objectif du protocole est de définir les messages que les agents pourront s'envoyer avec la dynamique opérationnelle associée. Le protocole de négociation que nous proposons est caractérisé par une suite de messages échangés entre un initiateur et des participants comme dans le cadre du *Contract Net Protocol*.

Les différentes phases du protocole

Le protocole se divise en trois phases : une phase de proposition, une phase de conversation et une phase de décision finale.

La phase de proposition Cette phase est la première phase de notre protocole, elle initie la négociation. Elle comprend la *proposition* du contrat par l'initiateur aux participants et la collecte des réponses de chacun des participants. Chaque participant peut soit *accepter*, soit *refuser* la proposition. Nous avons choisi de ne pas inclure de contre-proposition explicite dans cette phase car cette fonctionnalité n'est pas commune à toutes les formes de négociation étudiées. Les négociations permettant les contre-propositions utilisent la phase suivante du protocole. Si les contre-propositions sont autorisées dès la première proposition de l'initiateur, on peut utiliser la réponse *accepte* qui peut avoir un paramètre et dans ce cas, ce paramètre serait une contre-proposition.

La phase de conversation Cette phase de notre protocole est optionnelle, elle n'intervient que lorsque les participants n'ont pas été assez nombreux à accepter la proposition de contrat de l'initiateur et que les contre-propositions sont autorisées. L'initiateur indique alors aux participants cette possibilité. Une conversation entre l'initiateur et les participants se déroule durant laquelle des propositions de modifications sont échangées. Suite aux modifications proposées par les participants, l'initiateur leur propose un nouveau contrat et on se retrouve dans une nouvelle phase de proposition.

La phase de décision finale Cette phase de décision finale aboutit soit à la confirmation du contrat, soit à l'annulation du contrat. Cette décision est prise par l'initiateur selon les réponses des participants aux propositions qu'il leur a faites.

Les primitives de négociation

Pour mener à bien un processus de négociation entre agents, il est nécessaire de définir plusieurs primitives de négociation entre les agents. Il faut donc des primitives spécifiques à l'initiateur et des primitives spécifiques aux participants. Notre objectif ici n'est pas de faire communiquer un de nos agents avec n'importe quel autre agent issu d'une plateforme différente (ce qui nécessiterait une plateforme "FIPA compliant" ou plus simplement des agents communiquant via ACL), mais de faciliter la réalisation d'une application avec nos agents. Le séquencement de ces primitives est représenté à la Figure 3.4 dans le formalisme d'AgentUML 1.0 [Bauer et al., 2001, Huget et al., 2003].

Primitives de l'initiateur L'initiateur possède quatre primitives de négociation :

- *propose (contrat)* : c'est la première primitive que l'initiateur envoie aux participants pour leur proposer un contrat. Le contrat contient les différentes ressources à négocier.
- *demande modification (contrat)* : ce message indique aux participants que le contrat ne peut être conclu sous sa forme actuelle et qu'il faut le modifier. L'initiateur

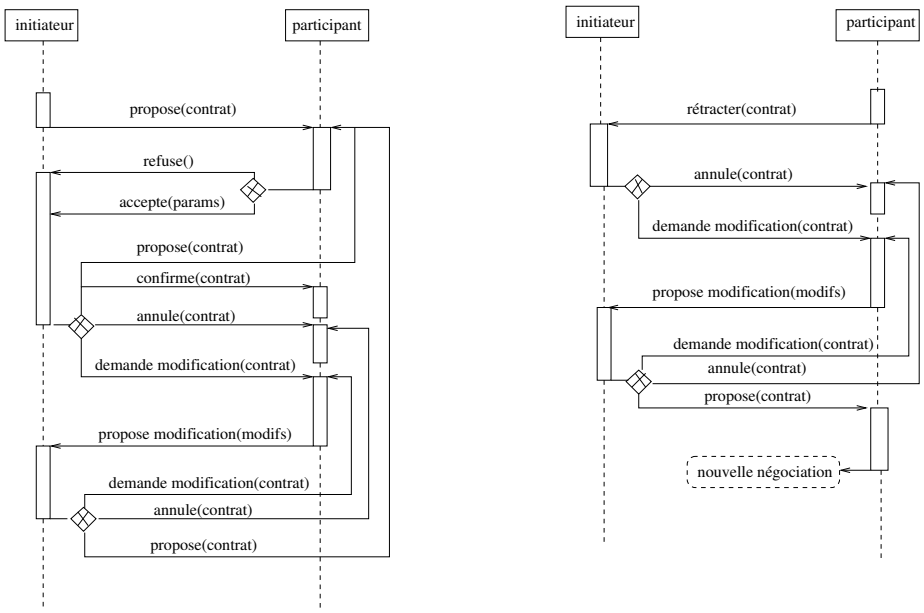


FIG. 3.4 – *Graphe d'interaction entre un initiateur et un participant : à gauche, le séquencement des messages pour la négociation d'un contrat entre un initiateur et des participants. Par souci de clarté, un seul participant est représenté ici. A droite, le séquencement des messages pour la renégociation d'un contrat pour lequel un participant s'est rétracté.*

demande aux participants de lui indiquer une ou plusieurs modifications possibles du contrat afin d'en proposer un nouveau convenant mieux à l'ensemble des participants. Ce peut également être un moyen de raffinement du contrat.
- *confirme (contrat)* : ce message indique aux participants que le contrat est confirmé. La négociation a été un succès.
- *annule (contrat)* : ce message indique aux participants que le contrat est annulé. La négociation a échoué.

Primitives du participant Les messages envoyés par les participants sont uniquement destinés à l'initiateur. Les autres participants n'ont pas connaissance de ces messages. De plus, les participants ne connaissent pas la liste des participants conviés à la négociation, ils ne peuvent donc pas former de coalition au cours de la négociation.
Le participant possède trois primitives de négociation :

- *accepte (paramètres)* : ce message répond à la proposition de contrat faite par l'initiateur. Le participant indique par ce message à l'initiateur qu'il accepte le contrat tel qu'il est. Il peut y avoir des paramètres dans le cas où le contrat est partiellement instancié. C'est le cas dans les enchères Vickrey où les participants doivent proposer un prix pour l'article mis en vente. Ce paramètre peut également être utilisé pour effectuer une contre-proposition lorsque l'application le permet.

– *refuse* : ce message répond à la proposition de contrat faite par l'initiateur. Le participant indique à l'initiateur qu'il refuse le contrat.
– *propose modification (liste modifications)* : ce message répond à une demande de modification de la part de l'initiateur. Le participant envoie à l'initiateur une liste de modifications possibles du contrat (une contre-proposition). Le nombre de modifications contenues dans la liste est un paramètre de la négociation. Cette liste peut être vide s'il n'existe pas de modifications possibles.

Une primitive de négociation est commune aux initiateurs et aux participants :
– *rétractation (contrat)* : le contrat avait été confirmé mais le participant ou l'initiateur ne peut (ou ne veut) plus l'honorer. Il décide donc de se rétracter. Suite à cette rétractation, l'initiateur peut renégocier automatiquement le contrat si le nombre d'accords restants est inférieur au nombre minimal d'accords nécessaires à la conclusion du contrat.

La dynamique opérationnelle

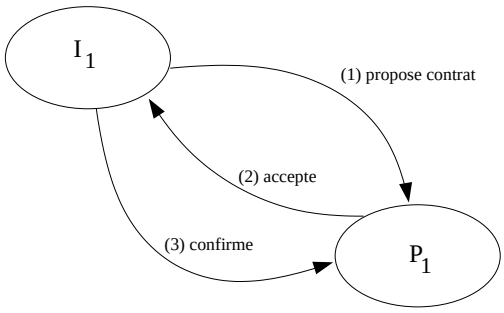


FIG. 3.5 – *Exemple simple de négociation : I_1 propose un contrat à P_1 qui l'accepte.*

Prenons l'exemple de la Figure 3.5 qui représente un exemple simple de négociation qui fait intervenir deux personnes : I_1 et P_1. I_1 joue le rôle de l'agent initiateur et P_1 celui de l'agent participant. I_1 crée le contrat et envoie à P_1 le message *propose contrat* (1). P_1 reçoit le contrat, l'étudie et envoie le message *accepte* à I_1 pour lui signaler qu'il accepte les termes du contrat (2). I_1 a donc reçu toutes les réponses (puisque l'ensemble des participants est constitué uniquement de P_1) et prend la décision de confirmer le contrat. Il envoie donc à P_1 le message *confirme* (3).

Examinons un exemple de négociation faisant intervenir 3 personnes : P_1, I_1 et I_2 (Figure 3.6). Pour cet exemple, P_1 considère que I_2 est plus prioritaire que I_1. Dans un premier temps, I_1 propose un contrat c_1 à P_1 qui accepte. I_1 joue le rôle de l'agent initiateur et P_1 celui de l'agent participant. I_1 crée le contrat et envoie à P_1 le message

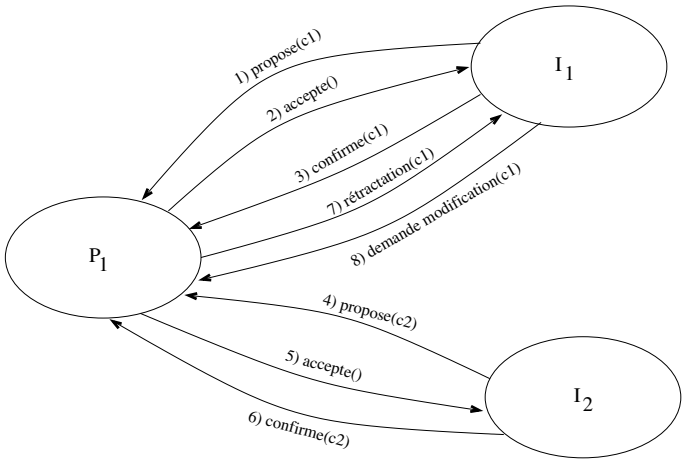


FIG. 3.6 – *Exemple de négociation avec renégociation : I_1 propose un contrat à P_1 qui l'accepte. Puis I_2 propose à P_1 un contrat entrant en conflit avec le précédent. P_1 trouve ce nouveau contrat plus avantageux et l'accepte. Comme I_2 confirme le contrat, P_1 se rétracte pour celui qu'il avait pris avec I_1. I_1 entame donc une renégociation du contrat avec P_1.*

propose(c_1). P_1 reçoit le contrat, l'étudie et envoie le message *accepte()* à I_1 pour lui signaler qu'il accepte les termes du contrat. I_1 confirme donc le contrat auprès de P_1 (message *confirme(c_1)*). Puis I_2 propose un contrat c_2 à P_1 pour les mêmes ressources. P_1, considérant I_2 plus prioritaire, envoie un message *accepte()* à I_2. Celui-ci confirme le contrat auprès de P_1 (message *confirme(c_2)*). P_1 prend alors le contrat avec I_2 et annule le précédent contrat pris avec I_1 (message *rétractation(c_1)*). Lorsque I_1 reçoit ce message, il décide de demander une modification pour ce contrat à P_1 (message *modifier contrat(c_1)*). La renégociation du contrat c_1 se fait donc automatiquement.

Cette fois-ci, considérons le cas où I_1 est plus prioritaire que I_2 pour P_1 (Figure 3.7). I_1 demande toujours en premier lieu un contrat à P_1 qui accepte et donc le contrat est confirmé. I_2 propose à son tour un contrat à P_1 pour les mêmes ressources. I_2 est moins prioritaire que I_1, P_1 refuse donc le contrat proposé par I_2 (message *refuse*). I_2 demande alors à P_1 de proposer une modification pour ce contrat (message *modifier contrat*). P_1 lui envoie à son tour une modification de contrat (message *contrat modifié*). I_2 examine alors la modification proposée par P_1 et lui propose un nouveau contrat (message *propose contrat*).

Afin de voir l'utilité d'un délai de réponse, prenons l'exemple décrit en Figure 3.8. I_1 est l'initiateur d'un contrat avec n participants : P_1 à P_n ($n > 5$). I_1 veut qu'au moins 50% des participants acceptent le contrat pour le confirmer. La réponse par défaut est d'accepter et le temps d'attente est fixé à 1 minute. P_1 et P_2 refusent le contrat. Après une minute, les autres participants n'ont pas répondu. I_1 considère donc la réponse par défaut pour ces $n - 2$ participants, c'est-à-dire qu'ils acceptent le contrat. Au total, il y a donc $n - 2$ acceptations et 2 refus, le contrat est donc confirmé.

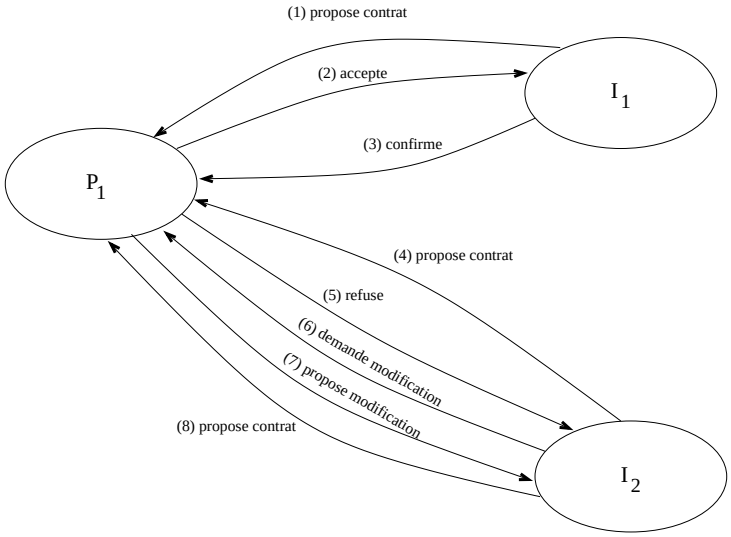


FIG. 3.7 – *Exemple de négociation avec conflit : I_1 propose un contrat à P_1 qui l'accepte. Puis I_2 propose à P_1 un contrat entrant en conflit avec le précédent. P_1 refuse ce contrat. I_2 demande alors à P_1 de lui faire une contre-proposition. Suite à celle-ci, I_2 propose un nouveau contrat à P_1.*

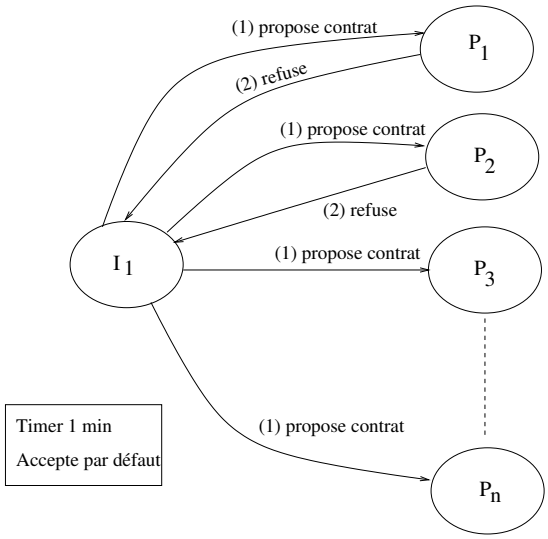


FIG. 3.8 – *Exemple de négociation avec timer : I_1 propose un contrat à n participants. Seuls deux d'entre eux répondent (ils refusent la proposition). Comme la réponse par défaut est l'acceptation de la proposition, I_1 va confirmer son contrat auprès des participants n'ayant pas répondu.*

La cardinalité de la négociation

La cardinalité de la négociation est une notion importante pour les SMA. Il s'agit de savoir combien d'agents négocient entre eux. Différents types de cardinalité de négociation existent [Guttman and Maes, 1998], depuis la négociation de 1 vers 1 jusqu'à n vers m. *Kasbah* [Chavez and Maes, 1996] est un exemple de négociation de 1 vers 1 : un acheteur négocie un article avec un vendeur à la fois. Cette forme de négociation n'est utile que lorsque deux personnes seulement sont impliquées dans la négociation. Mais lorsque la négociation implique plusieurs participants avec un initiateur, il s'agit de négociation de 1 vers n. C'est cette situation que l'on rencontre dans les systèmes de vente aux enchères comme *Fishmarket* ou *Magnet*. Notre protocole est basé sur l'échange de messages entre un initiateur et plusieurs participants, il permet donc de réaliser des négociations de 1 vers n. Comme nous le verrons dans la prochaine sous-section en page 94, nous autorisons la négociation simultanée de plusieurs contrats. C'est-à-dire que plusieurs initiateurs peuvent proposer simultanément un contrat à un ensemble de participants, il s'agit donc de négociation de n vers m agents, ou plus exactement n négociations simultanées de 1 vers m agents.

Le paramétrage du protocole

```
<!ELEMENT protocol (answer-delay,default-answer,minAgreements,
nbRounds,nb-modifications-by-round,retraction-allowed,
nbRenegotiations)>
<!ELEMENT answer-delay (#PCDATA)>
<!ELEMENT default-answer EMPTY>
<!ATTLIST default-answer value (accept | refuse) "refuse">
<!ELEMENT minAgreements (#PCDATA)>
<!ELEMENT nbRounds (#PCDATA)>
<!ELEMENT nb-modifications-by-round (#PCDATA)>
<!ELEMENT retraction-allowed EMPTY>
<!ATTLIST retraction-allowed value (true | false) "true">
<!ELEMENT nbRenegotiations (#PCDATA)>
```

FIG. 3.9 – *Fichier DTD pour la configuration du protocole de négociation. On y retrouve le délai de réponse des participants associé à une réponse par défaut, le nombre minimal d'accords nécessaires à la conclusion du contrat, le nombre de tours de parole des participants et le nombre de modifications qu'ils peuvent proposer par tour de parole. Il est également indiqué si la rétractation est autorisée et dans ce cas, le nombre maximal de renégociations qui seront effectuées.*

Nous avons vu différents types de négociation et une proposition d'un protocole de négociation général. Nous décrivons ici différents paramètres permettant, à partir du noyau de négociation, de formaliser les différents types de négociation. Nous avons choisi de configurer ces paramètres dans un fichier et d'utiliser le format XML pour ce

fichier, nous avons donc conçu un fichier DTD (Figure 3.9) afin de pouvoir valider le fichier de configuration.

Délai de réponse et réponse par défaut Lors de négociations distribuées comme c'est la cas lorsque ce sont des agents qui agissent pour le compte d'un utilisateur, il se peut qu'un participant ne réponde pas à la proposition de l'initiateur, soit parce qu'il est absent, soit parce qu'une panne est survenue, il faut alors que la négociation ne soit pas bloquée. Afin de permettre à la négociation de continuer, un mécanisme de temps d'attente des réponses est mis en place et lorsque ce temps est écoulé, l'initiateur considère une réponse par défaut pour le participant. Cette réponse par défaut sera bien souvent un refus de la proposition : en effet, pour les négociations commerciales, il est hors de question de forcer un participant à acheter le bien en question. Cette réponse par défaut qui est communiquée aux participants leur permet de ne pas répondre si leur réponse est celle définie par défaut, ainsi on limite le nombre de communications. Même si cela peut paraître étrange, une non-réponse qui vaut acceptation peut s'avérer utile, notamment lorsque la négociation se déroule dans le cadre de la prise de rendez-vous, par exemple. La réponse par défaut est définie à chaque proposition de contrat et est la même pour l'ensemble des participants pour ce contrat. Elle peut donc être différente pour deux propositions consécutives. Si l'initiateur désire recevoir les réponses à ses propositions sous 10 minutes et considère un refus par défaut, les paramètres seront les suivants : `<answer-delay>10</answer-delay>` et `<default-answer value="refuse"/>`.

Nombre d'accords nécessaire pour la confirmation du contrat Pour que l'initiateur puisse décider de confirmer ou d'annuler le contrat suite aux réponses fournies par les participants, un paramètre fixant le nombre minimal d'accords nécessaires pour confirmer le contrat est mis en place. Ce nombre peut prendre la forme d'un pourcentage. Par exemple, pour une enchère, il suffit qu'un seul participant accepte le contrat, tandis que pour d'autres applications, il peut être nécessaire que tout le monde accepte la proposition : `<minAgreements>100%</minAgreements>`.

Nombre de tours de parole des participants Afin de ne pas avoir une phase de conversation infinie, nous avons défini le nombre de tours de négociation, c'est-à-dire le nombre de fois où les participants pourront proposer une modification du contrat, faire une contre-proposition. Nous avons choisi de borner la durée de la négociation par un nombre de tours de parole couplé à un temps d'attente des réponses plutôt que par une durée maximale, comme c'est souvent le cas, car nous pensons que la négociation sera plus efficace ainsi. On peut effectivement supposer que le nombre de contre-propositions effectuées sera supérieur si les agents doivent répondre dans des délais plus brefs que lorsqu'ils ne connaissent qu'une date limite de fin de négociation

et dans ce cas répondent moins rapidement aux initiateurs. Mais cette utilisation provient plus spécifiquement du fait qu'une date limite de négociation pose de nombreux problèmes et plus principalement celui d'une référence de temps universelle ainsi que celui lié au décalage horaire qui peut exister entre les agents. En effet, la synchronisation des différents ordinateurs sur lesquels tournent les agents est un réel problème que nous n'avons pu résoudre.

Ce nombre de tours sera donc fixé à zéro si la négociation est du type *à prendre ou à laisser*, ou si ce sont des enchères à offres scellées au meilleur ou au second meilleur prix : `<nbRounds>0</nbRounds>`. A chaque tour de parole, les participants peuvent proposer un certain nombre de modifications à apporter au contrat. Ce nombre est fixé par le paramètre `<nb-modifications-by-round>0</nb-modifications-by-round>`.

Renégociation automatique Souvent, lors de négociations, certains contrats ne peuvent plus être honorés et doivent être renégociés. C'est par exemple le cas lors de la prise de rendez-vous. Un participant peut être amené à annuler un rendez-vous à cause d'un imprévu mais voudrait en obtenir un autre. Pour cette raison, nous proposons de pouvoir se rétracter d'un contrat pris et de renégocier automatiquement les contrats qui doivent l'être. Dès qu'un initiateur reçoit une rétractation pour un contrat, il peut choisir de l'annuler pour tous les participants et éventuellement d'entrer dans une nouvelle phase conversationnelle afin de trouver un nouvel accord. L'initiateur peut également ne rien faire si le nombre d'accords reste supérieur au nombre minimal d'accords nécessaires pour le succès de la négociation.

Nous utilisons pour cela les paramètres `<retraction-allowed value="true"/>` et `<nbRenegotiations>5</nbRenegotiations>` du fichier de configuration du protocole. Lorsque la renégociation est autorisée, l'initiateur d'un contrat ne pouvant plus être honoré peut automatiquement le renégocier tant que le nombre maximum de renégociations n'est pas atteint pour ce contrat.

Cette possibilité de renégocier automatiquement un contrat n'est pas fournie par les différentes plateformes que nous avons présentées dans l'état de l'art, comme *Magnet* ou *GNP*. Seules *GeNCA* et *ADEPT* offrent cette possibilité, *GeNCA* introduisant en plus la possibilité de borner le nombre de renégociations du contrat.

Tous ces paramètres nous permettent donc de décrire une négociation spécifique à partir du noyau commun. Enlever des paramètres rendrait le système moins puissant, moins général. Prenons l'exemple du nombre de tours dans le processus de négociation, si nous l'enlevions et ne faisions donc plus que de la négociation en une seule proposition (qui nous le rappelons, n'est pas de la négociation à nos yeux), nous ne pourrions plus implémenter les enchères anglaises ou hollandaises. L'idéal est d'avoir un système fournissant ce noyau et prenant en compte ces paramètres afin d'instancier différentes négociations. C'est ce que nous apportons.

L'évaluation de notre protocole

Notre protocole se voulant général, nous décrivons ici les types d'applications réalisables avec ce protocole. Cette liste n'est bien sûr pas exhaustive mais reprend les types de négociation évoqués au chapitre 2.

Comme nous l'avons mentionné auparavant, notre protocole s'inspire du *Contract Net* tout comme le FIPA-Contract-Net proposé par la *FIPA* [fip,]. La différence principale entre notre protocole et celui de la *FIPA* est que nous ajoutons une phase optionnelle de conversation. Ainsi, il est possible pour les participants de formuler des contre-propositions et ainsi converger vers une solution plus rapidement qu'avec le FIPA-Contract-Net où seul l'initiateur formule des propositions et ne peut donc pas s'appuyer sur les préférences des participants pour aboutir à une solution.

Notre protocole décrit les messages pouvant être échangés entre les agents et plus spécialement l'ordre de ces messages et les tours de parole des agents, mais il n'impose pas le contenu des messages (par exemple, il n'impose pas de prix). Il peut donc être utilisé par de nombreuses applications, ce qui n'est pas le cas de plusieurs autres protocoles dont celui utilisé dans *Zeus* [Nwana et al., 1999] qui est dédié aux places de marché.

Notre protocole convient pour la famille des systèmes de votes présentée en section 2.2.1, page 17. Si l'on veut réaliser un système de vote par la méthode de Borda, l'initiateur propose tour à tour chacune des ressources (représentant une alternative) et les participants lui répondent par un *accepte* contenant le nombre de points qu'ils attribuent à la ressource. L'initiateur conservera la ressource ayant obtenu la meilleure note afin de confirmer le contrat la concernant.

Notre protocole convient également à la famille des enchères présentée en section 2.2.2, page 23. Pour les *enchères à offres scellées*, l'initiateur propose un article et les participants répondent en donnant un prix en argument de la méthode *accepte* s'ils veulent enchérir pour l'article, sinon ils rejettent la proposition. Si aucun participant n'a proposé de prix satisfaisant pour l'initiateur, on entre dans la phase de conversation où les modifications proposées sont un nouveau prix. Ce processus se termine lorsqu'un prix satisfaisant a été proposé, ou lorsque personne ne surenchérit ou encore lorsque le nombre de tours de parole des participants défini par l'initiateur est atteint.

Pour les *enchères hollandaises*, l'initiateur propose un article à un prix très élevé et si personne n'accepte la proposition, l'initiateur propose à nouveau l'article en baissant le prix sans demander de modification aux participants. Le processus se termine lorsqu'un participant accepte la proposition, ou lorsque le prix atteint le prix minimum voulu par l'agent, ou encore lorsque le nombre de tours défini par l'initiateur est atteint.

La famille des négociations basées sur le *Contract Net Protocol* présentée en section 2.2.3, page 29, est en partie réalisable avec notre protocole. Il est par exemple possible d'utiliser notre protocole pour une négociation du type *à prendre ou à laisser* si la phase de conversation n'est pas utilisée. Notre protocole convient également pour les négociations multi-attributs lorsque ceux-ci sont négociés en même temps.

Notre protocole n'est néanmoins pas adapté aux négociations qui doivent être traitées en plusieurs étapes : par exemple, pour négocier l'achat d'une voiture, on peut négocier le modèle puis la couleur puis le prix, etc. Notre protocole ne permet pas de séquencer un contrat afin de le négocier point par point. Il permet cependant de traiter chaque étape du contrat. Nous pensons qu'il ne faudrait donc ajouter qu'un mécanisme permettant à l'initiateur de négocier chaque étape du contrat l'une après l'autre, c'est-à-dire lui permettre de créer un contrat par étape et le succès de la négociation d'une étape entraînerait le début de la négociation de l'étape suivante. Nous tentons actuellement d'effectuer cette modification.

Les négociations combinées ne peuvent pas non plus être implémentées avec ce protocole car elles nécessitent un lien entre plusieurs contrats. On ne peut pas créer deux contrats et dire que les deux doivent être pris ou aucun. Si l'on veut obtenir plusieurs ressources d'une même personne, il faut les mettre dans un même contrat, mais si l'on veut plusieurs ressources de personnes différentes, il faut créer un contrat par personne/ressource mais on ne peut pas spécifier que tous les contrats doivent être pris ou aucun. Pour pouvoir utiliser ce type de négociation, il faudrait ajouter un mécanisme de liaison entre les contrats, qui permettrait de savoir si les différentes négociations ont échoué ou sont en attente du succès des autres et ainsi pouvoir confirmer tous les contrats en même temps, ou terminer la négociation des contrats dès que la négociation de l'un d'eux est en échec. Cette modification fait partie de nos perspectives à moyen terme.

La famille des négociations par argumentation (présentée en section 2.2.4, page 33) n'est pas incluse dans *GeNCA*, bien que le protocole puisse convenir. En effet, les paramètres des différentes méthodes peuvent être des arguments. Nous pensons donc qu'en utilisant *GeNCA* avec des agents sachant argumenter et en faisant interagir le négociateur avec l'agent, il serait possible d'appliquer la négociation par argumentation.

De façon plus générale, notre protocole satisfait les différentes propriétés nécessaires à un framework générique de négociation décrites dans [Jennings et al., 2000], à savoir :

> "
> – *the minimum requirement of a negotiating agent is the ability to make and respond to proposals*
> – *proposals can be made either independently of other agents' proposals or based on the negotiation history*

– *the minimal requirement for the "other agents" (ie not the proposer) is that they are able to indicate dissatisfaction with proposals that they find unacceptable*
– *If agents can only accept or reject others' proposals, then negotiation can be very time consuming and inefficient.(etc.) To improve the efficiency of the negotiation process, the recipient needs to be able to provide more useful feedback on the proposals it receives. This feedback can take the form of a* critique *or a* counter-proposal*. From such feedback, the proposer should be in position to generate a proposal that is more likely to lead to an agreement.*

"

3.3.2 La gestion des négociations

La gestion des négociations que nous avons choisie permet d'effectuer plusieurs négociations simultanément lorsqu'aucun conflit n'apparaît sur les ressources impliquées. Différentes gestions des négociations entrant en conflit sont proposées : les négociations sont effectuées soit séquentiellement, soit simultanément.

Les micro-agents

Afin de pouvoir négocier simultanément plusieurs contrats, nous avons choisi de confier la négociation d'un contrat à un micro-agent[6]. Nous avons défini deux types de micro-agents : les buts et les engagements. Un micro-agent but est chargé de la négociation d'un contrat pour lequel l'agent tient le rôle d'initiateur. Un micro-agent engagement est chargé de la négociation d'un contrat pour lequel l'agent tient le rôle de participant. Chaque micro-agent est responsable de la négociation du contrat pour lequel il a été créé et a la charge de suivre le protocole de négociation qui a été défini pour l'application. Les micro-agents sont indépendants les uns des autres, ce qui leur permet de négocier leur contrat en même temps.

La représentation graphique des négociations

Afin de représenter les négociations, nous avons défini un modèle graphique (Figure 3.10). Une droite segmentée représente l'ensemble des ressources disponibles,

[6]Nous parlons de micro-agents car chaque micro-agent a sa propre forme d'autonomie. En effet, chaque micro-agent réagit aux messages de négociation qui lui arrive et envoie une réponse à l'initiateur s'il participe à la négociation (par exemple l'acceptation de la proposition) ou à l'ensemble des participants s'il initie la négociation (par exemple la confirmation du contrat). Le fonctionnement d'un micro-agent est détaillé dans le prochain chapitre en section 4.2.2

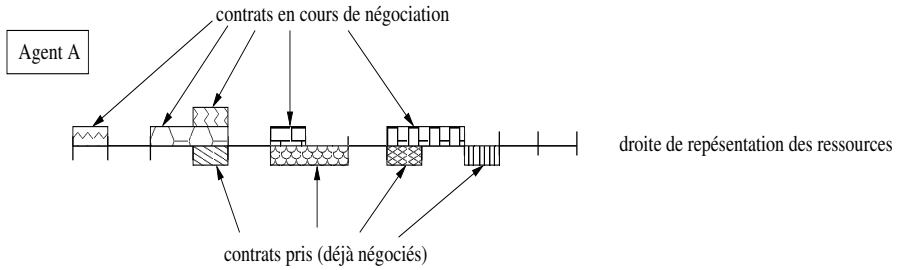


FIG. 3.10 – *Représentation des négociations. Une droite segmentée représente les ressources (un segment = une ressource). En dessous de la droite, on trouve les contrats pris par l'agent (ceux qui ont été confirmés) face aux ressources concernées. Au dessus de la droite, les contrats en cours de négociation sont empilés sur les ressources qu'ils nécessitent. Certains contrats portent sur une ressource, d'autres sur plusieurs ressources*

chaque ressource correspondant à un segment. La partie inférieure contient les contrats négociés sur ces ressources, c'est-à-dire les contrats pris par l'agent, les ressources correspondantes n'étant alors plus libres. La partie supérieure contient les contrats qui sont en cours de négociation sur les ressources correspondantes. Cette partie supérieure représente une matrice où les colonnes sont les ressources et où les contrats arrivent dans une liste de type premier arrivé, premier sorti. En fait, il s'agit d'une matrice dont le comportement est inspiré du jeu Tetris™ : les contrats qui arrivent s'empilent en cas de conflit sur les ressources. Les contrats en cours de négociation sont ceux situés à la surface (base de la colonne ainsi formée), les autres étant en attente. Lors de la fin d'une négociation, le contrat est retiré de la surface et celui qui le suivait est alors débloqué, c'est-à-dire que la négociation de ce contrat peut alors commencer.

Il peut y avoir des négociations en cours sur des ressources déjà prises car il est possible que l'initiateur du contrat en cours soit plus prioritaire pour l'agent que celui dont le contrat a été pris auparavant et donc l'agent peut prendre ce nouveau contrat et se dédire du précédent. La notion de priorité entre les participants est introduite dans la section 3.4.

Fonctionnement de la matrice Pour fixer les idées, prenons l'exemple de la Figure 3.11. Il y a quatre contrats : c_1, c_2, c_3 et c_4 et quatre ressources : r_1, r_2, r_3 et r_4. Le premier contrat c_1 qui arrive porte sur les ressources r_1 et r_3. Comme aucun contrat n'a encore été proposé, toutes les ressources sont libres et donc le contrat c_1 est placé à la surface de la matrice dans les colonnes correspondants aux ressources r_1 et r_3. Le processus de négociation du contrat c_1 est donc entamé. Puis, un second contrat c_2 est proposé pour la ressource r_2. Comme aucun contrat n'est en cours de négociation pour cette ressource, le contrat c_2 est placé à la surface de la matrice, dans la colonne correspondant à la ressource r_2. La négociation du contrat c_2 est entamée. Un troisième contrat c_3 est proposé pour les ressources r_2 et r_3. Malheureusement, ces deux res-

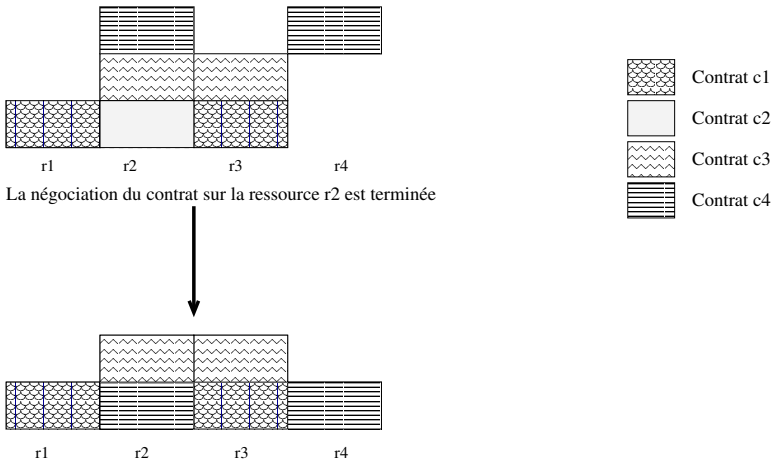


FIG. 3.11 – *Gestion des négociations entrant en conflit pour une ou plusieurs ressources. Les contrats c1 et c2 sont en cours de négociation pour les ressources {r1,r3} et {r2}. Le contrat c3 attend la libération des ressources r2 et r3. Le contrat c4 attend la libération de la ressource r2. Supposons que le contrat c2 soit annulé. La ressource r2 est donc libérée. Le contrat c3 ne peut toujours pas être négocié car la ressource r3 n'est pas libre. En revanche, le contrat c4 qui nécessite les ressources r2 et r4 peut maintenant être négocié.*

sources sont en cours de négociation. Le contrat c_3 est donc placé dans la matrice dans les colonnes correspondants aux ressources r_2 et r_3 au dessus des contrats c_1 et c_2. La négociation du contrat c_3 est mise en attente. Un quatrième contrat c_4 arrive ensuite. Ce contrat porte sur les ressources r_2 et r_4. Comme la ressource r_2 est en cours de négociation pour le contrat c_2, le contrat est placé au-dessus des autres. Comme c_3, le contrat c_4 est mis en attente. Supposons que la négociation du contrat c_2 soit terminée. Le contrat c_2 est donc retiré de la matrice, libérant ainsi la ressource r_2. Le contrat c_3 ne peut toujours pas descendre à la surface de la matrice car la ressource r_3 n'est pas libre. Donc, c'est le contrat c_4, qui nécessite les ressources r_2 et r_4, qui tombe à la surface de la matrice. Le processus de négociation du contrat c_4 commence alors.

C'est cette matrice qui permet aux micro-agents (but ou engagement) de savoir si le processus de négociation de leur contrat peut commencer ou non (c'est le cas lorsque le contrat se situe à la surface de la matrice). Si un contrat est déjà en cours de négociation sur les ressources nécessaires pour le contrat arrivant, ce dernier est alors mis en attente jusqu'à ce que les ressources soient libérées.

La gestion des négociations est par défaut basée sur cette matrice. Toutefois, l'utilisateur peut choisir de ne pas utiliser cette matrice et de négocier simultanément tous ses contrats. Dans ce cas, on permet de débuter une négociation sur une ressource même si une autre négociation sur la même ressource est déjà en cours.

Dans ces deux modes de gestion des négociations, on permet de commencer une négociation portant sur une ressource déjà prise par un contrat.

Exemple de la Figure 3.6 Nous présentons ici les représentations des négociations de P_1, I_1 et I_2 lors des différentes étapes des négociations de la Figure 3.6.

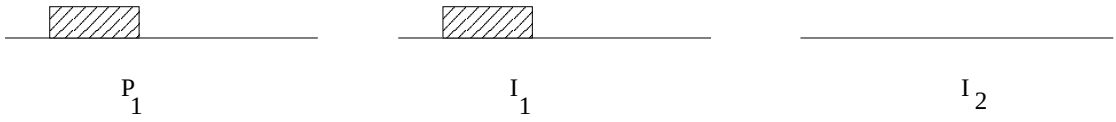


FIG. 3.12 – *Représentation des négociations des agents P_1, I_1 et I_2 de la Figure 3.6 lors de la proposition de contrat de I_1 à P_1*

Lorsque I_1 propose son contrat à P_1, les représentations des négociations des agents sont celles présentées en Figure 3.12. Les ressources nécessaires au contrat ne sont pas en cours de négociation, le contrat est donc placé en surface de la matrice. La négociation de ce contrat est entamée.

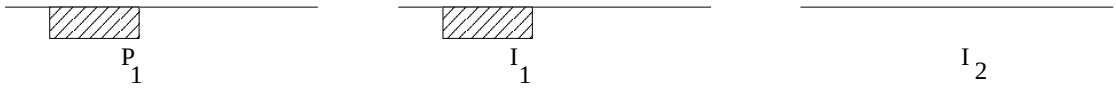


FIG. 3.13 – *Représentation des négociations des agents P_1, I_1 et I_2 de la Figure 3.6 lorsque I_1 a confirmé son contrat avec P_1*

Lorsque I_1 confirme son contrat auprès de P_1, le contrat passe dans la zone des contrats pris dans la représentation des négociations de I_1 et P_1 (Figure 3.13).

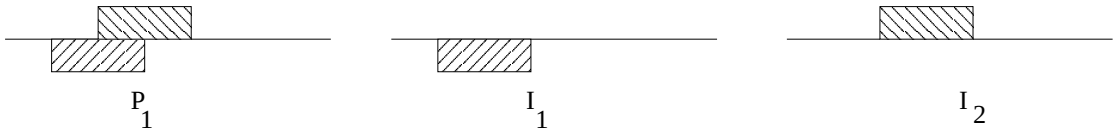


FIG. 3.14 – *Représentation des négociations des agents P_1, I_1 et I_2 de la Figure 3.6 lors de la proposition de contrat de I_2 à P_1*

Au moment où I_2 propose son contrat à P_1, ce dernier a déjà pris le contrat avec I_1. Les représentations des négociations sont alors celles présentées en Figure 3.14. P_1 et I_1 ont un contrat confirmé qui est placé sous la droite de représentation des ressources face aux ressources qui sont alors prises. P_1 et I_2 ont placé le contrat proposé par I_2 dans leur matrice respective. Les contrats proposés par I_1 et I_2 ont une ressource en commun, on observe donc le chevauchement entre le contrat pris avec I_1 et celui proposé par I_2 dans la représentation des négociations de P_1.

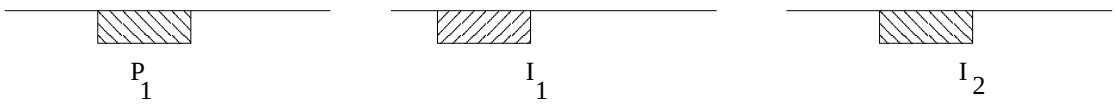


FIG. 3.15 – *Représentation des négociations des agents P_1, I_1 et I_2 de la Figure 3.6 lorsque I_2 a confirmé son contrat avec P_1*

Comme P_1 considère que I_2 est plus prioritaire que I_1, P_1 accepte le contrat de I_2. I_2 confirme alors son contrat auprès de P_1. P_1 enlève donc le contrat pris avec I_1 de la zone des contrats pris et place le contrat pris avec I_2. P_1 envoie une rétraction à I_1. Avant que I_1 ne reçoive la rétractation de P_1, les représentations des négociations de P_1, I_1 et I_2 sont représentées en Figure 3.15. P_1 et I_2 ont pris un contrat ensembles et I_1 pense toujours avoir un contrat avec P_1, c'est pourquoi le contrat est encore dans la zone des contrats pris.

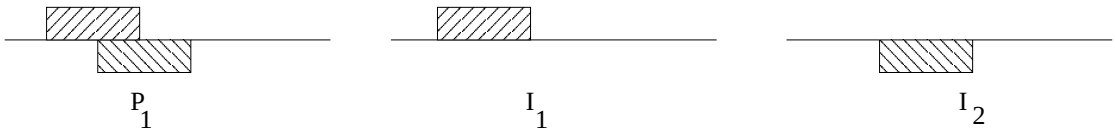


FIG. 3.16 – *Représentation des négociations des agents P_1, I_1 et I_2 de la Figure 3.6 lorsque I_1 demande une modification de son contrat à P_1*

Lorsque I_1 reçoit la rétractation de P_1, il lui demande une modification de ce contrat. Le contrat est alors à nouveau placé dans la zone des contrats en cours de négociation dans les représentation des négociations des agents P_1 et I_1 (Figure 3.16).

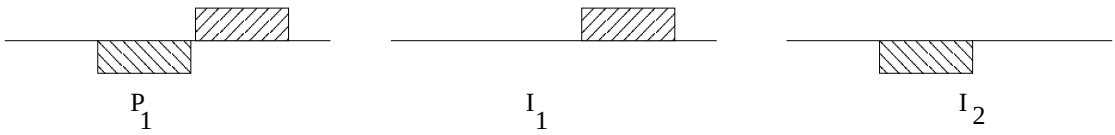


FIG. 3.17 – *Représentation des négociations des agents P_1, I_1 et I_2 de la Figure 3.6 lors de la nouvelle proposition de contrat de I_1 à P_1*

A la suite de la proposition de modification de P_1, I_1 propose un nouveau contrat sur d'autres ressources à P_1. L'ancien contrat est alors remplacé par le nouveau dans la représentation des négociations des agents P_1 et I_1 dans la zone des contrats en cours de négociation (Figure 3.17).

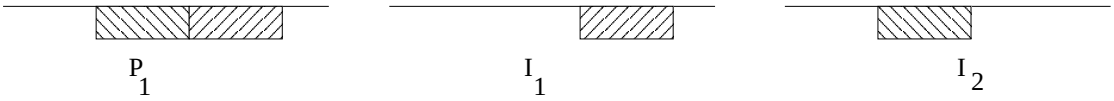


FIG. 3.18 – *Représentation des négociations des agents P_1, I_1 et I_2 de la Figure 3.6 lorsque I_1 a confirmé son nouveau contrat avec P_1*

P_1 accepte ce contrat et I_1 le confirme. Les représentations des négociations sont alors celles présentées en Figure 3.18 : P_1 a deux contrats dans la zone des contrats pris (celui qu'il a pris avec I_1 et celui qu'il a pris avec I_2), I_1 et I_2 possèdent chacun un contrat dans la zone des contrats pris (celui qu'ils ont respectivement pris avec P_1).

Les modes de gestion des négociations entrant en conflit

Lorsqu'une personne doit négocier de nombreux contrats, il peut être préférable d'en négocier plusieurs simultanément afin de gagner du temps. Cela ne pose pas de problèmes lorsque les ressources mises en jeu dans les contrats sont différentes, c'est pourquoi nous effectuons toujours les négociations de ces contrats simultanément. En revanche, il peut s'avérer souhaitable de négocier séquentiellement les contrats portant sur des ressources communes. Notre modèle propose deux façons de gérer les négociations de contrats portant sur des ressources communes : la gestion séquentielle et la gestion en parallèle. Le mode de gestion est un paramètre de l'application de négociation. Lorsque l'utilisateur choisit la gestion en parallèle, aucune restriction n'est faite sur la disponibilité des ressources, elles peuvent déjà être en cours de négociation pour un autre contrat. En revanche, lorsque l'utilisateur choisit la gestion séquentielle, le système utilise la matrice présentée auparavant afin de gérer les négociations.

La Figure 3.19 représente les différents cas qui peuvent se présenter pour la négociation de deux contrats ayant une ressource commune $r1$. Dans le cas de la gestion parallèle, on négocie tous les contrats simultanément, sans se préoccuper du conflit. En revanche dans le cas de la gestion séquentielle, on négocie tous les contrats les uns à la suite des autres. Le cas de la gestion semi-parallèle consiste à décomposer une négociation portant sur n ressources en n négociations portant sur une seule ressource. Chacune de ces négociations sera réalisée dès que la ressource sur laquelle elle porte n'est plus en cours de négociation. Si l'on reprend l'exemple de la Figure 3.19, cela consiste à négocier le premier contrat qui porte sur la ressource $r1$ et à négocier en même temps la ressource $r2$ du deuxième contrat. La négociation de la ressource $r1$ du deuxième contrat s'effectue lorsque la négociation du premier contrat est terminée.

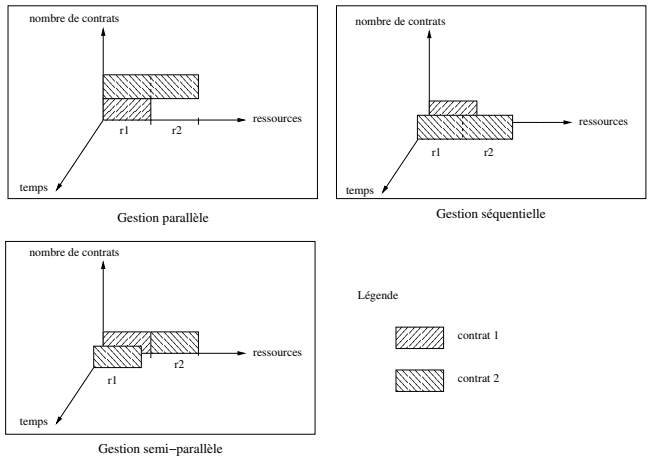


FIG. 3.19 – *Les différents cas de gestion des négociations portant sur des ressources communes. Gestion parallèle : tous les contrats sont négociés en parallèle. Aucune restriction n'est faite quant à la disponibilité des ressources. Gestion séquentielle : les contrats sont négociés séquentiellement. Gestion semi-parallèle : les contrats sont scindés en plusieurs parties (une par ressource). Chaque partie portant sur une ressource libre est négociée de suite. Les autres attendent la libération des ressources.*

Notre modèle ne propose pas de gestion semi-parallèle des négociations portant sur des ressources communes car ce type de gestion peut aboutir à des négociations inutiles sur plusieurs ressources d'un contrat alors que ce dernier n'a aucune chance d'être accepté.

Prenons un exemple où un contrat c_1 est proposé sur la ressource r_1 et un contrat c_2 est proposé sur les ressources r_1, r_2 et r_3. Supposons que le contrat c_1 est proposé avant le contrat c_2 et que la rétractation n'est pas autorisée.

La ressource r_1 est en cours de négociation pour le contrat c_1 et on effectue alors les négociations des ressources r_2 et r_3 du contrat c_2. Si le contrat c_1 est confirmé, alors la ressource r_1 n'est plus négociable et le contrat c_2 ne peut plus être accepté, donc les négociations sur les ressources r_2 et r_3 ont été inutiles.

C'est la raison pour laquelle notre modèle ne propose pas la décomposition d'un contrat sur n ressources en n contrats portant sur une seule ressource.

Notre modèle de gestion des négociations permet de négocier simultanément tous les contrats portant sur des ensembles de ressources disjoints et offre la possibilité de négocier soit séquentiellement, soit simultanément les autres contrats.

À part *Fishmarket*, toutes les plateformes étudiées fournissent un mécanisme de simultanéité. Cependant, cette simultanéité se fait sans contrôle de conflit pouvant apparaître entre les négociations, comme c'est le cas de *Zeus*. *GeNCA*, en revanche, permet

de vérifier qu'il n'y a pas de conflit entre deux négociations avant de les exécuter en parallèle.

La gestion des deadlocks

Les deadlocks constituent un problème important dans les applications de négociation. Ils peuvent causer d'importants dommages s'ils ne sont pas pris en compte et résolus. Dans notre modèle, un participant à une négociation peut être l'initiateur d'une autre négociation simultanée alors que les ressources sont partagées sans qu'il se produise un deadlock.

Les deadlocks peuvent survenir par exemple lorsqu'un agent ne répond pas à une proposition de contrat soit parce que l'agent n'est plus disponible (à cause d'une coupure du réseau par exemple), soit parce que cette proposition est mise en attente car elle entre en conflit avec une autre négociation et que la gestion séquentielle est appliquée. Nous avons choisi de résoudre ce problème en limitant l'attente des réponses des participants et en considérant une réponse par défaut. Ce délai d'attente des réponses ainsi que cette réponse par défaut font partie des paramètres du protocole de négociation (voir page 82).

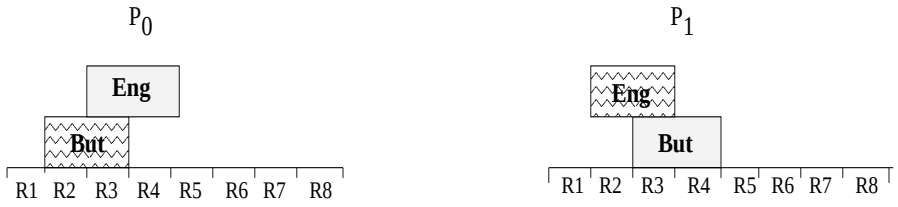


FIG. 3.20 – *Une situation de deadlock. Deux agents P_0 et P_1 créent un contrat sur des ressources communes et le propose à l'autre. Lorsque la gestion séquentielle des contrats conflictuels est utilisée, chacun attend la réponse de l'autre avant de donner la sienne. Le deadlock est évité grâce au mécanisme de délai d'attente des réponses. Le contrat ayant le plus petit délai de réponse sera annulé, ce qui déclenchera la négociation de l'autre contrat.*

Pour illustrer le cas de la non réponse d'un participant à une proposition de contrat qui entre en conflit avec une autre négociation alors que la gestion séquentielle est appliquée, prenons l'exemple de deux agents : P_0 et P_1, qui créent simultanément un contrat sur une ressource identique et le proposent à l'autre. Chaque initiateur place son contrat dans la matrice de gestion des négociations (Figure 3.20). Puis la proposition de contrat de l'autre initiateur arrive et le contrat est également placé dans la matrice. Comme la gestion séquentielle des contrats entrant en conflit est utilisée, chaque initiateur attend la réponse de l'autre avant de donner la sienne, puisque chacun effectue uniquement la négociation de son contrat. Cependant, dans notre système, le deadlock

ne se produit pas car chaque proposition de contrat est associée à un délai d'attente des réponses. La négociation associée au délai le plus court est alors abandonnée, ce qui entraîne le début de la négociation associée au délai le plus long.

3.4 Le niveau stratégique

Le succès d'une négociation dépend bien sûr de stratégies adaptées au problème traité. Pour être optimale, une stratégie nécessite une expertise du domaine de l'application de négociation. Notre modèle sépare donc le niveau stratégique afin de permettre à l'utilisateur d'écrire sa propre stratégie de négociation pour une application de négociation spécifique. En effet, la négociation de rendez-vous n'a pas les mêmes tenants et aboutissants que la négociation de produits de grande consommation telle qu'on la retrouve sur les places de marché ou encore que la négociation d'articles de seconde main aux enchères. Nous proposons ici deux stratégies de négociation qui prennent en compte les listes de priorité sur les ressources et les personnes et qu'il est possible d'utiliser dans toute négociation n'impliquant pas d'autres paramètres que les ressources (donc non-utilisables pour les enchères où il faut proposer un prix).

Nous avons vu que notre protocole distingue deux rôles : initiateur et participant. La stratégie de négociation n'est pas la même selon le rôle de l'agent, il y a donc deux sortes de stratégies. La stratégie de l'initiateur lui permet de décider de confirmer ou d'annuler le contrat et de synthétiser les différentes propositions de modifications des participants afin de proposer un nouveau contrat. L'autre stratégie est celle du participant qui l'utilise pour décider d'accepter ou de refuser la proposition de contrat et de trouver une modification pour le contrat lorsque l'initiateur en demande une.

Afin d'élaborer des stratégies, nous avons mis en place des listes de priorité et des outils permettant de prendre en compte les résultats des négociations passées.

3.4.1 Listes de priorité

Afin de pouvoir élaborer des stratégies, nous avons mis en place dans le modèle deux listes de priorités, une pour les ressources et une pour les personnes. La priorité est un entier compris entre 1 et 10, 10 étant la meilleure note. Ainsi chaque agent peut définir un ordre sur les ressources et les personnes. Chaque agent est libre de modifier les priorités qu'il a données à tout moment. Ces listes de priorité ont deux utilisations principales : pour un participant, elles lui permettent de choisir le contrat qu'il souhaite prendre en cas de conflits. Par exemple, au sein d'une équipe, le chef aura une plus grande priorité que les collègues. Si deux collègues ont pris un contrat sur une ressource et que le chef leur propose un contrat sur la même ressource, ils prendront le

contrat avec le chef et annuleront le leur. Ces listes de priorités permettent également aux initiateurs de pondérer les modifications proposées par les participants selon leur priorité et les priorités des ressources qu'ils proposent.

La place de l'agent dans la liste de priorité. Lorsque l'on donne des priorités aux personnes avec qui l'on va négocier, le problème de sa propre priorité se pose. En effet, peut-on convenir que l'on est plus prioritaire que tous les autres ? Nous ne pensons pas que cela puisse être une bonne solution. Imaginons que nous fassions partie d'une grande entreprise. Cette entreprise a une organisation hiérarchique et nous ne sommes pas à la tête de cette hiérarchie. Nous ne pouvons donc pas être plus prioritaire que nos supérieurs hiérarchiques. De même, il ne paraît pas cohérent de s'attribuer la plus petite priorité possible, puisque d'autres personnes peuvent être plus bas dans cette même hiérarchie. Il n'est donc pas possible de fixer la priorité de l'agent une fois pour toute avant de connaître les autres personnes avec qui il va négocier. La priorité de l'agent est donc elle aussi un paramètre dans la négociation.

3.4.2 Outils pour la création de stratégies

Deux types d'outils sont fournis : d'une part, des outils permettant de consulter les listes de priorité, d'autre part, des outils permettant de prendre en compte les négociations passées.

Les outils de gestion des listes de priorité mis à disposition permettent de :
- récupérer la priorité d'une personne,
- comparer deux personnes pour savoir laquelle est plus prioritaire,
- récupérer la liste des ressources dans l'ordre de priorité que l'utilisateur leur a donné,
- récupérer les ressources libres et dans l'ordre de priorité,
- récupérer les ressources libres ou prises par quelqu'un de moins important et dans l'ordre de priorité.

En ce qui concerne les outils permettant de prendre en compte les négociations passées, il est possible de :
- connaître la liste des contrats ou le dernier contrat proposé par une personne,
- connaître les ressources qui ont été proposées par une personne,
- connaître la ressource la plus (resp. moins) proposée par une personne,
- connaître la liste des personnes ayant proposé un ensemble de ressources,
- connaître le nombre de fois où une ressource a été proposée par une personne,
- savoir pour un couple personne/ressource les taux de succès et de rétractation pour les contrats proposés par l'agent sur cette ressource à cette personne.

Il est ainsi possible de définir des stratégies telles que :
- si les ressources proposées ne l'ont pas déjà été par ce même initiateur, alors j'accepte sa proposition.

– si l'initiateur s'est déjà rétracté pour un de mes contrats, alors je refuse sa proposition.
– si les ressources proposées sont libres ou prises par quelqu'un de moins important, alors j'accepte la proposition.
– etc.

Nous présentons maintenant la stratégie par défaut que nous proposons pour le rôle d'initiateur, puis celle pour le rôle de participant. Ces stratégies utilisent uniquement les listes de priorité et sont utilisables dans toutes les négociations concernant l'affectation ou le partage des ressources, comme par exemple la création d'emplois du temps. En effet, les contrats que nous définissons contiennent uniquement des ressources, il n'est donc pas possible de prendre en compte d'autres paramètres tels qu'un prix dans ces stratégies.

3.4.3 Comportement de l'initiateur

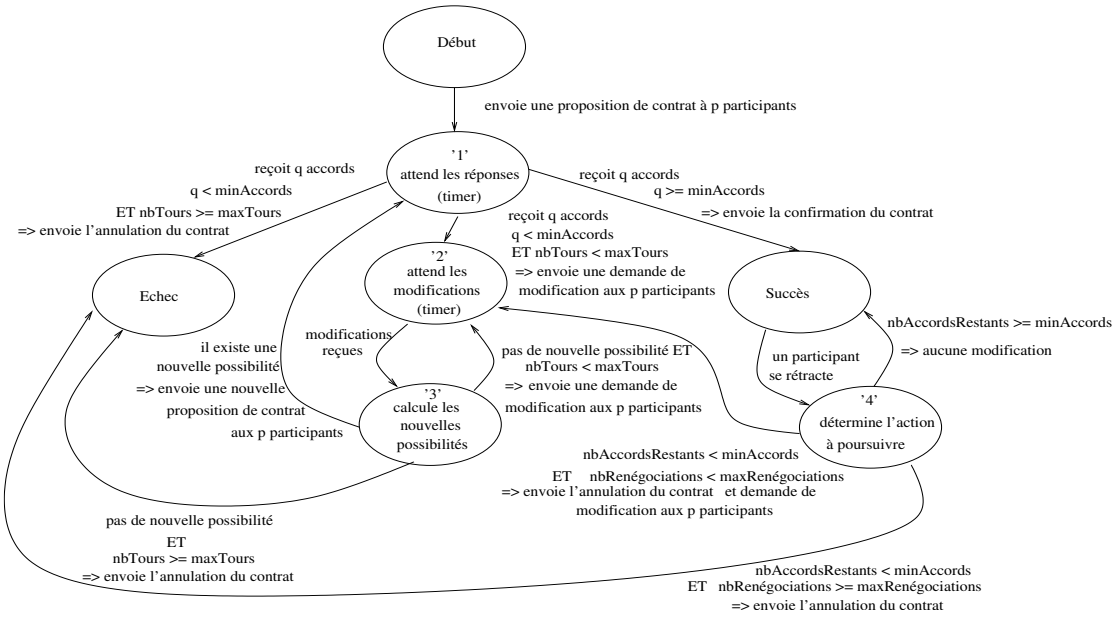


FIG. 3.21 – *Le graphe du comportement de l'initiateur. Cet automate à états décrit le comportement de l'initiateur face aux messages reçus de la part des participants et aux paramètres de négociation*

Le comportement de l'initiateur peut être formalisé par un graphe (figure 3.21). Tout commence par la proposition d'un contrat à un ensemble de `p` participants. L'initiateur passe dans l'`état '1'` où il attend les réponses des participants. Lorsque l'initiateur envoie la proposition de contrat, il définit un délai d'attente des réponses afin d'éviter

d'attendre éternellement les réponses des participants. Si un participant ne répond pas avant la limite définie, l'initiateur prend en compte une réponse par défaut qu'il a défini à la création du contrat. Chaque fois qu'une réponse est reçue, l'initiateur la prend en compte et met à jour ses informations le renseignant sur le nombre de réponses reçues et le nombre d'accords reçus, par exemple. Lorsque les réponses ont toutes été reçues ou que le délai d'attente a expiré, l'initiateur a trois possibilités qui s'offrent à lui, qui dépendent du nombre `q` d'accords qu'il a reçu :

1. Si `q` est supérieur ou égal au nombre minimal d'accords nécessaires à la prise du contrat, alors il confirme le contrat et passe à l'`état de succès`.
2. Si `q` est inférieur au nombre minimal d'accords nécessaires à la prise du contrat et si le nombre de tours de négociation déjà effectué est supérieur ou égal au nombre maximal de tours de négociation choisi par l'initiateur à la création du contrat, il annule le contrat et passe dans un `état d'échec`.
3. Si `q` est inférieur au nombre minimal d'accords nécessaires à la prise du contrat et si le nombre de tours de négociation déjà effectué est inférieur au nombre maximal de tours de négociation choisi par l'initiateur à la création du contrat, il demande aux `p` participants de lui proposer une modification pour le contrat et passe à l'`état '2'`.

Lorsque l'initiateur envoie sa demande de modification, il utilise le délai d'attente des réponses afin d'éviter d'attendre indéfiniment les modifications proposées par les participants. Lorsque l'initiateur a reçu toutes les modifications ou que le délai d'attente a expiré, l'initiateur passe dans l'`état '3'`. Lorsqu'il est dans l'`état '3'`, l'initiateur prend en compte toutes les modifications envoyées par les participants afin de trouver une nouvelle possibilité pour le contrat. Pour cela, il donne une note à chaque ressource selon la formule suivante :

$$\text{note}(r_i) = \text{priorité}(r_i, \text{init}) * \text{priorité}(\text{init},\text{init}) + \sum_{j=1}^{n} \text{priorité}(r_i, \text{part}_j) * \text{priorité}(\text{part}_j, \text{init})$$

où :

- *priorité(r_i,init)* est la priorité de la ressource r_i pour l'initiateur. Pour des raisons d'équité entre l'initiateur et les participants, on prend en compte le même nombre de ressources pour l'initiateur que pour les participants.
- *priorité(init,init)* est la priorité que l'initiateur s'est donnée.
- *priorité(r_i,$part_j$)* est la priorité relative de la ressource r_i pour le participant $part_j$. En effet, le participant ne communique pas les priorités qu'il donne à ses ressources. L'initiateur considère alors que la première ressource envoyée est la plus prioritaire chez le participant et lui attribue ainsi la priorité maximale. Puis il décrémente la priorité de 1 pour les ressources suivantes, sans descendre sous la valeur minimale. Si la ressource ne fait pas partie de la liste reçue, on lui attribue une priorité discriminante égale à 0.

– *priorité($part_j$,init)* est la priorité du participant $part_j$ pour l'initiateur.

Cette note est remise à jour à chaque réception de modification : les notes des nouvelles ressources proposées par participant sont incrémentées par le calcul effectué dans la deuxième partie de la formule. Les nouvelles ressources prises en compte pour l'initiateur voient leurs notes augmentées par le calcul effectué dans la première partie de la formule.

S'il existe une possibilité, l'initiateur envoie une nouvelle proposition de contrat aux `p` participants et passe à nouveau dans l'`état '1'`. S'il n'y a pas de nouvelle possibilité, alors l'initiateur a deux solutions. Si le nombre de tours de négociation est inférieur au nombre maximal de tours de négociations, alors il demande à nouveau aux participants de lui envoyer une modification pour le contrat et passe dans l'`état '2'`. Si le nombre maximal de tours de négociation est atteint, alors l'initiateur envoie une annulation pour le contrat et passe dans un `état d'échec`.

Lorsque l'initiateur est dans l'`état de succès`, il peut recevoir de la part d'un participant une rétractation. C'est-à-dire que le participant n'est plus en mesure d'honorer le contrat. L'initiateur passe alors dans l'`état '4'` où il doit décider que faire face à cette rétractation. Si le nombre minimal d'accords nécessaires à la réussite de la négociation est encore atteint, l'initiateur ne fait rien et passe à nouveau dans l'`état succès`. Si ce nombre n'est plus atteint (ou que la rétractation provient de l'initiateur), la suite dépend du nombre de renégociations déjà effectuées. Si ce nombre est supérieur au nombre maximal de renégociations défini à la création du contrat par l'initiateur, alors l'initiateur annule le contrat et passe dans un `état d'échec`. Sinon, l'initiateur annule le contrat et demande aux participants de proposer une modification pour ce contrat et passe dans l'`état '2'`.
La renégociation est donc automatique (pour peu qu'elle soit autorisée).

Évaluation de la formule utilisée pour noter les ressources

Prenons un exemple concret pour illustrer le fonctionnement de cette formule. Rappelons tout d'abord que les priorités des ressources et des participants sont comprises entre 1 et 10 et que 10 est la meilleure note possible. Le choix de la nouvelle ressource à proposer se fait donc en prenant la ressource ayant la plus grande note.

Dans cet exemple, cinq ressources sont présentes : h_1 à h_5 et l'initiateur propose un contrat à trois participants : P_1, P_2 et P_3. L'initiateur s'est donné la priorité maximale (10), il a donné une priorité égale à 10 à P_1, P_2 a une priorité de 5 et la priorité de P_3 vaut 1.

La figure 3.22 montre les priorités respectives des ressources pour chaque personne. On remarque que P_1 enverra les ressources dans l'ordre suivant : h_5, h_4, h_3, h_2, h_1. P_2,

ressource	priorités définies par			
	initiateur	P_1	P_2	P_3
h_1	8	4	1	10
h_2	10	5	7	8
h_3	9	6	10	2
h_4	3	7	9	5
h_5	2	8	2	1

FIG. 3.22 – *Priorités données aux ressources par l'initiateur et les participants*

quant à lui, enverra les ressources dans l'ordre : h_3, h_4, h_2, h_5, h_1. P_3 les enverra dans l'ordre : h_1, h_2, h_4, h_3, h_5.

Supposons que chaque participant envoie une seule ressource par demande de modification. P_1 envoie alors la ressource h_5, P_2 envoie h_3, P_3 envoie h_1 et on ne considère pour l'initiateur que la ressource h_2. Après la première demande de modification, on aura alors les notes suivantes :

- note(h_1) = $0 * 10 + 0 * 10 + 0 * 5 + 10 * 1 = 10$
- note(h_2) = $10 * 10 + 0 * 10 + 0 * 5 + 0 * 1 = 100$
- note(h_3) = $0 * 10 + 0 * 10 + 10 * 5 + 0 * 1 = 50$
- note(h_4) = $0 * 10 + 0 * 10 + 0 * 5 + 0 * 1 = 0$
- note(h_5) = $0 * 10 + 10 * 10 + 0 * 5 + 0 * 1 = 100$

Le classement des ressources qui en ressort est donc {h_2, h_5, h_3, h_1, h_4}. Ce résultat était prévisible, puisque l'initiateur et P_1 ont la priorité maximale, leurs choix se retrouvent donc en première position. Puis vient le choix de P_2 et celui de P_3. La ressource h_4 n'ayant pas été proposée, elle se retrouve en dernière position.

L'initiateur va donc proposer un nouveau contrat pour la ressource h_2. Supposons que ce contrat soit à nouveau rejeté par les participants (ce qui est cohérent puisqu'aucun d'entre eux n'a proposé cette ressource). L'initiateur va à nouveau demander une modification aux participants. P_1 et P_2 proposeront alors h_4, P_3 proposera h_2 et l'initiateur h_3. Les notes seront alors mises à jour de la façon suivante :

- note(h_1) = 10
- note(h_2) = $100 + 9 * 1 = 109$
- note(h_3) = $50 + 9 * 10 = 140$
- note(h_4) = $0 + 9 * 10 + 9 * 5 = 135$
- note(h_5) = 100

Le classement des ressources devient alors : {h_3, h_4, h_2, h_5, h_1}. Comme h_2 a déjà été proposée, elle n'entre plus en compte pour la sélection de la nouvelle ressource à proposer. Cela garantit que les propositions s'arrêteront au plus tard lorsque toutes les solutions possibles auront été proposées.

Cette fois-ci, l'initiateur va proposer un contrat pour la ressource h_3 aux participants. Cette proposition a toutes les chances d'aboutir, d'après les priorités qui ont été attribuées à h_3 par les participants. Seul P_3 est susceptible de refuser cette proposition, mais comme il a la plus basse priorité pour l'initiateur, cela ne posera pas de problème.

Cette formule permet de tenir compte de la priorité de chaque participant pour l'agent initiateur ainsi que de la priorité par participant de chaque ressource proposée. De plus, les ressources non proposées sont notées de façon discriminatoire afin qu'elles ne soient pas proposées dans le nouveau contrat.

Nous avons élaboré cette formule pour prendre en compte l'opinion de tous et l'importance donnée à cette opinion par l'initiateur de façon à aboutir plus rapidement à un accord. En effet, si l'on choisit de ne pas noter les ressources mais de choisir au hasard celle que l'on va proposer, on a alors 40% de chances de choisir h_1 ou h_5 dans l'exemple précédent, qui sont les ressources les moins bonnes candidates vis à vis des préférences des participants et de l'initiateur.

On peut également choisir de ne prendre en compte que la priorité de la ressource pour l'initiateur. On utilise alors la formule :

$$\text{note}(r_i) = \text{priorité}(r_i, \text{init})$$

où *priorité(r_i,init)* est la priorité de la ressource r_i pour l'initiateur. L'initiateur ne formule donc des propositions qu'à partir de ses préférences sur les ressources. Si l'on reprend l'exemple de la Figure 3.22, l'initiateur proposera tout d'abord la ressource h_2, puis h_3, puis h_1, puis h_4 et enfin h_5.

Or, si l'on regarde les préférences des participants, on remarque que la ressource h_2 n'est que l'avant dernière ressource préférée pour le participant P_1, la troisième pour P_2 et la deuxième pour P_3. La proposition de la ressource h_2 n'est donc pas un bon choix au regard de la satisfaction des participants.

Cette formule ne tient compte ni des préférences des participants, ni de l'importance des participants pour l'initiateur.

En revanche, la formule que nous utilisons tient compte de ces préférences et est donc mieux adaptée pour converger plus rapidement vers une solution acceptable par tous.

3.4.4 Comportement du participant

Le comportement du participant peut être formalisé par un graphe (figure 3.23). Tout commence lors de la réception d'une proposition de contrat. Le participant passe

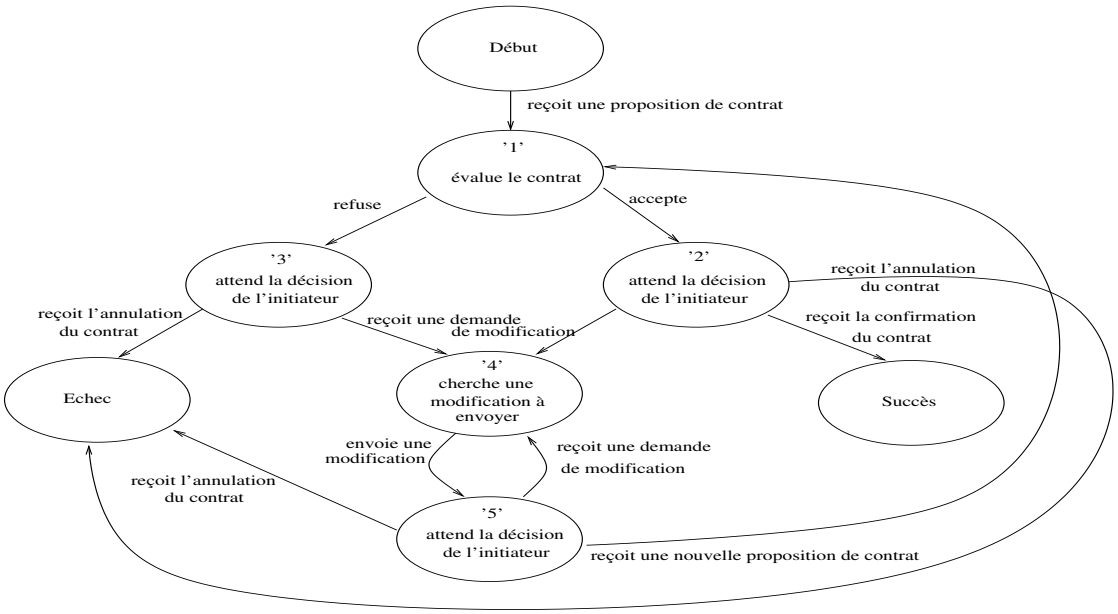


FIG. 3.23 – *Le graphe du comportement du participant. Cet automate à états décrit le comportement du participant face aux messages reçus de l'initiateur.*

dans l'`état '1'`. Cet état est celui où l'agent examine le contrat, l'évalue et choisit soit de l'accepter et dans ce cas il passe à l'`état '2'`, soit de le refuser et passe à l'`état '3'`. Lorsqu'il est dans l'`état '2'`, le participant attend la réponse de l'initiateur qui sera soit la confirmation du contrat, soit une demande de modification du contrat, soit l'annulation du contrat. Si le contrat est confirmé, le participant passe à un `état de succès`. Si l'initiateur demande une modification pour le contrat, il passe à l'`état '4'`. Si le contrat est annulé, ce peut être le cas lorsque l'initiateur souhaite que tous les participants acceptent le contrat et que l'un d'eux le refuse et que la négociation est du type sans contre-proposition, alors le participant passe à un `état d'échec`.
Lorsque le participant est dans l'`état '3'`, il attend la réponse de l'initiateur qui sera soit l'annulation du contrat, soit une demande de modification du contrat. Si le contrat est annulé, le participant passe à un `état d'échec`. Sinon, il passe à l'`état '4'`.
Lorsqu'il est dans l'`état '4'`, le participant recherche une modification possible du contrat et l'envoie à l'initiateur. Le participant passe alors dans l'`état '5'`. Dans cet `état '5'`, le participant attend la décision de l'initiateur, qui sera soit la proposition d'un nouveau contrat, soit la demande d'une autre modification, soit l'annulation du contrat. Si le contrat est annulé, le participant passe dans un `état d'échec`. Si l'initiateur demande une autre modification, le participant passe à l'`état '4'`. Si une nouvelle proposition de contrat est reçue, le participant passe dans l'`état '1'`.

3.5 Complexité du système

La complexité est une caractéristique importante pour la négociation. C'est l'une des principales raisons pour laquelle il est nécessaire d'utiliser des agents négociateurs. Examinons la complexité en nombre de messages induite par notre protocole. Nous supposons ici que ni le broadcast, ni le multicast UDP ne sont utilisés, c'est-à-dire qu'on envoie un message par destinataire [7].

Dans le pire des cas, le nombre de messages peut être en $O(m^n)$ si n est la profondeur du processus de cascade et m le nombre d'agents impliqués dans une négociation. On imagine facilement ce qui pourrait arriver à une secrétaire dans ce cas pour organiser une réunion avec 50 personnes !

Pour prouver ce résultat, regardons les différents cas possibles.

3.5.1 Ordre linéaire

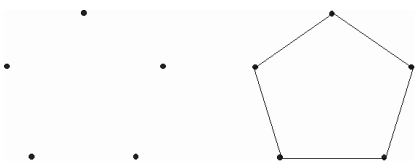


FIG. 3.24 – *Complexité linéaire*

Supposons que m personnes soient impliquées dans la négociation d'un contrat (l'initiateur et $m-1$ participants). La Figure 3.24 montre 5 personnes, avant et après que le contrat ait été pris (chaque point représente une personne).

Dans un premier temps, considérons que tous les participants acceptent la proposition. L'initiateur *propose* le contrat, chaque participant *accepte* et l'initiateur *confirme* le contrat : $3 * (m-1)$ messages sont échangés.

Dès qu'un participant n'est pas d'accord, l'initiateur *demande une modification* du contrat aux participants qui en envoient alors une à l'initiateur (message *propose modification*). $2 * (m-1)$ messages sont alors échangés. L'initiateur envoie une nouvelle proposition qui sera acceptée, ce qui ajoute $3*(m-1)$ messages. Au total, $7*(m-1)$ messages sont échangés, en tenant compte de la première proposition de l'initiateur et les

[7] Il s'agit ici d'une complexité théorique dont le calcul est basé sur les messages échangés dans notre protocole de négociation et non sur les messages qui transitent sur le réseau et dépendent de l'implémentation réalisée

réponses des participants dont au moins une est négative. L'initiateur envoie $4*(m-1)$ messages et en reçoit $3*(m-1)$. Chaque participant reçoit 4 messages et en envoie 3.

Prendre un contrat, avec ou sans demande de modification et sans renégociation d'autres contrats, a une complexité globale en O(m), linéaire pour l'initiateur et en O(1) pour les participants.

3.5.2 Ordre quadratique

Premier cas

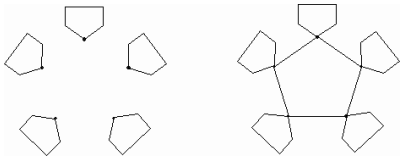


FIG. 3.25 – *Complexité quadratique - Premier cas*

Supposons maintenant que la prise d'un contrat remette en question des contrats pris précédemment avec d'autres personnes (Figure 3.25).

Pour simplifier, tous les contrats impliquent m personnes et le contrat négocié est plus prioritaire que tous les contrats pris précédemment.

Les participants acceptent le contrat, $3*(m-1)$ messages sont alors échangés. Mais lors de la confirmation du contrat, chaque participant devra modifier son contrat pris précédemment. Supposons que les participants sont les initiateurs des contrats qu'ils doivent modifier. Ils vont alors envoyer un message pour *annuler* le contrat, puis un message pour *demander une modification* du contrat, soit $2*(m-1)$ messages. Ils recevront $m-1$ modifications et proposeront un nouveau contrat que nous supposons accepté dès la première proposition. Le nombre de messages par renégociation est de $6*(m-1)$. Les participants du premier contrat, considérés comme les initiateurs des seconds, envoient $4*(m-1)$ messages et en reçoivent $2*(m-1)$. Lorsque toutes les renégociations sont indépendantes, il y a en tout $(m-1)$ renégociations et donc $6*(m-1)^2$ messages.

Le nombre total de messages échangés pour prendre le premier contrat est alors $6*(m-1)^2+3*(m-1)$. L'initiateur envoie $2*(m-1)$ messages et en reçoit $(m-1)$. Chaque participant reçoit $2+2*(m-1)$ messages et en envoient $1+4*(m-1)$.

Prendre un contrat impliquant une renégociation de contrat par participant a une complexité globale en O(m^2) et linéaire pour l'initiateur et les participants.

Second cas

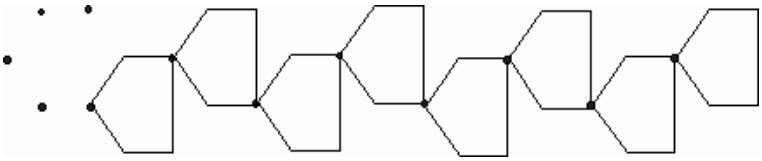


FIG. 3.26 – *Complexité quadratique - Second cas*

Supposons maintenant qu'un seul participant doive modifier un contrat pris précédemment avec $(m-1)$autres personnes (Figure 3.26). Durant la renégociation, cette personne doit également modifier un autre contrat et cela récursivement sur n personnes. La négociation principale nécessite $3*(m-1)$ messages, les autres $6*(m-1)$. Le nombre total de messages est donc $(3+6*n)(m-1)$.

Prendre un contrat nécessitant la renégociation d'un autre contrat par un participant et cela récursivement sur n personnes a une complexité globale en O($m*n$) et linéaire pour l'initiateur et les participants.

3.5.3 Ordre exponentiel

Pour prouver le résultat donné en début de section, prenons un exemple formel.

Supposons que tous les agents doivent remettre en question un contrat jusqu'à une profondeur n (Figure 3.27), que les contrats soient pris avec l'initiateur et deux participants et que tous ces déplacements soient indépendants.

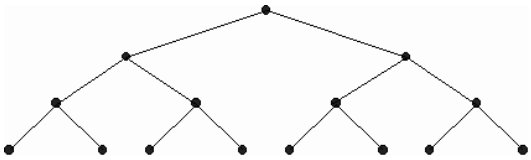


FIG. 3.27 – *Complexité de la négociation d'un contrat avec deux participants nécessitant des renégociations en cascade. Chaque point représente un agent. La complexité globale est exponentielle (en O(2^n)).*

Le nombre de messages échangés pour une modification de contrat qui sera acceptée immédiatement est égal à 6 : annulation du contrat, demande de modification, modification de la part des participants, proposition d'un contrat modifié, acceptation des participants puis confirmation de ce nouveau contrat. Le nombre d'agents à un niveau i est égal à 2^i et le nombre de messages échangés à ce niveau est égal à $6*2^i$.

La complexité globale est de O(2^n) et est linéaire pour l'initiateur et les participants.

Si nous supposons maintenant que les contrats ne sont plus indépendants mais que les agents situés à la profondeur n demandent à l'initiateur du contrat principal d'en déplacer un autre, le nombre de demandes de renégociations sera de 2^n pour l'initiateur.

La complexité globale est toujours en O(2^n) et reste linéaire pour les participants par contre elle devient en O(2^n) pour l'initiateur.

Ces résultats sur la complexité en envoi de messages induite par la négociation justifient l'usage d'un système automatique de négociation. Prenons un exemple d'une négociation impliquant 50 personnes. Si la négociation implique que chaque participant doive renégocier un contrat déjà pris, la complexité en envoi de messages de la négociation serait alors en $O(50^2)$ soit en $O(2500)$, ce qui veut dire que le nombre de messages envoyés serait de l'ordre de 2500. On imagine facilement que l'automatisation de ces envois de messages simplifierait la vie d'une secrétaire, qui ne passerait alors plus son temps à essayer de joindre chaque personne par téléphone et pourrait alors se consacrer à d'autres tâches.

3.6 Conclusion

Nous avons présenté dans ce chapitre la réification des différents types de négociation que nous avons étudiés, notre modèle générique de négociation et sa décomposition en trois niveaux ainsi que la complexité en envoi de messages de notre protocole de négociation.

Certains points communs entre les différentes négociations sont évidents, comme la présence de ressources et de personnes, d'autres sont plus difficiles à trouver. Nous avons cependant réussi à extraire un ensemble minimal d'actes de langage nécessaires pour la négociation que nous avons utilisés pour créer un protocole général de négociation. Ce protocole se spécialise grâce à de nombreux paramètres, tels que le nombre minimum d'accords nécessaires pour conclure le contrat, le nombre de tours de parole des participants et la possibilité de se rétracter et de renégocier les contrats.

Le modèle de négociation que nous proposons est basé sur une architecture à trois niveaux qui sont la communication, la négociation et la stratégie. Nous pensons en effet que la communication et la stratégie employée pour négocier sont spécifiques à une application et ne peuvent pas être fixées une fois pour toute. Il est important de pouvoir utiliser notre modèle au sein d'applications déjà existantes en utilisant par conséquent les mécanismes de communication entre agents qui leur sont propres. De même, chaque

application concerne un problème de négociation spécifique et il est nécessaire d'adapter sa stratégie de négociation à ce problème pour obtenir des résultats optimaux.

Seul le niveau de négociation est commun à toutes les applications, il comprend notre protocole général de négociation ainsi que le mécanisme de gestion des négociations que nous avons défini. Notre gestion des négociations réalise simultanément toutes les négociations ne portant pas sur des ressources communes et utilise une matrice dont le comportement est inspiré du jeu Tetris™ pour les négociations entrant en conflit. Cela permet de négocier séquentiellement ces négociations. Ce niveau est le plus important de notre modèle et constitue l'essentiel de nos travaux de recherche. Les points forts de ce niveau sont la possibilité de négocier simultanément plusieurs contrats, de renégocier automatiquement les contrats qui doivent être déplacés et l'élimination des deadlocks. Notre modèle possède en outre les qualités de généricité, d'uniformisation des négociations et d'automatisation des envois de messages.

Nous allons maintenant décrire l'implémentation de ces niveaux dans l'API *GeNCA* que nous avons réalisée afin de pouvoir tester notre modèle avec des applications de négociation de natures différentes et de pouvoir ainsi valider notre modèle.

Chapitre 4

GeNCA, une API Java implémentant notre modèle

Nous avons présenté dans le chapitre précédent le modèle général de négociation de contrats que nous proposons. Nous présentons dans ce chapitre l'implémentation de ce modèle que nous avons réalisée sous la forme d'une API Java. Elle reprend l'architecture en trois niveaux qui correspondent à trois paquetages différents. Nous détaillons l'implémentation de chacun de ces paquetages et plus particulièrement celui concernant la négociation, qui forme le cœur de notre modèle. Les diagrammes de classe des trois niveaux sont représentés en Annexe A. Nous présentons également dans ce chapitre le paquetage concernant l'interface utilisateur que nous avons élaborée afin qu'un utilisateur humain puisse interagir avec son agent durant les négociations. Enfin nous montrons comment utiliser ce paquetage afin de réaliser une application de négociation.

4.1 Implémentation du niveau communication

Comme nous l'avons indiqué dans le chapitre précédent, cette couche de communication permet d'abstraire le processus de communication entre les agents. Nous présentons ici le format des messages échangés par les agents, le serveur de noms utilisé pour l'abonnement des agents et enfin l'interface *Communicator* du paquetage `fr.lifl.genca.communication`, qui contient les différentes méthodes nécessaires pour interagir avec le serveur de noms, ainsi que quelques implémentations de cette interface.

4.1.1 Le format des messages

Nous avons défini un format général des messages échangés au sein de *GeNCA*. Chaque message possède un *émetteur*, une *primitive* et des *paramètres* éventuels. Ces caractéristiques communes à chaque message ont donné lieu à la création d'une interface `Message`. Il y a deux types de messages au sein de *GeNCA* : des messages de liaison avec le serveur de noms et des messages de négociation entre les agents. Ces messages diffèrent selon le type de primitives employées et les messages de négociation possèdent des caractéristiques supplémentaires. Nous présentons ici ces deux types de messages.

Le format des messages de liaison avec le serveur

Le serveur et les agents s'échangent des messages du type `ServerMessage`. Les primitives utilisées sont du type `ServerPrimitive`. Le serveur envoie les primitives :

- $PARTICIPANTS\_AND\_RESOURCES$ en réponse à l'inscription de l'agent, le serveur envoie la liste des personnes déjà inscrites pour le même type d'application ainsi que les ressources présentes pour ce type d'application ;
- $PARTICIPANT\_ARRIVAL$ pour indiquer l'arrivée d'un nouveau participant aux personnes déjà inscrites, le serveur envoie le nom de la nouvelle personne et la liste des ressources qu'elle apporte ; et
- $RECEIVE\_MESSAGE$ pour indiquer la réception d'un message de négociation de la part d'un autre agent, le serveur transfère un message du type `NegotiationMessage`.

L'agent envoie les primitives :

- $SUBSCRIBE$ pour s'inscrire auprès du serveur de noms, il fournit son nom, son adresse, le type d'application qu'il souhaite rejoindre et les ressources qu'il apporte ;
- $CONNECT$ pour se connecter, l'agent donne son nom et son adresse ;
- $DISCONNECT$ pour se déconnecter, l'agent donne uniquement son nom ; et
- $SEND\_MESSAGE$ pour envoyer un message de négociation à d'autres agents, l'agent donne le message à envoyer (de type `NegotiationMessage`) et la liste des agents destinataires.

Le format des messages de négociation

Les agents s'échangent des messages de type `NegotiationMessage`. Ce message comprend en plus de l'émetteur du message, de la primitive de négociation employée et des paramètres nécessaires pour le traitement de cette primitive, l'identifiant de la négociation ainsi que l'état de la négociation.

Les différentes primitives utilisées sont du type `NegotiationPrimitive`, ce sont celles décrites dans le protocole de négociation, à savoir :
- $PROPOSE$, pour proposer un contrat ;
- $ACCEPT$, pour accepter la proposition de contrat ;
- $REFUSE$, pour refuser la proposition de contrat ;
- $MODIFICATION\_REQUEST$, pour demander une modification du contrat ;
- $PROPOSE\_MODIFICATION$, pour proposer une modification du contrat ;
- $CONFIRM$, pour confirmer le contrat ;
- $ANNUL$, pour annuler le contrat ;
- $RETRACT$ pour se rétracter d'un contrat pris précédemment ; et
- $CANCEL$ pour annuler un contrat suite à une rétractation.

L'*identifiant* du contrat (qui est unique) sert à identifier la négociation concernée par le message et à le confier au micro-agent associé.

L'*état* de la négociation permet à un initiateur de vérifier si le message qu'il reçoit est encore valide. En effet, la négociation change d'état selon le dernier message envoyé par l'initiateur. À la première proposition de contrat, l'état sera "p1". Lorsque le délai de réception des réponses sera atteint et qu'il manque des réponses, l'état changera en "tooLate". Si une réponse arrive, elle indiquera l'état "p1" (l'état indiqué par le message qui implique cette réponse). L'initiateur compare cet état "p1" à l'état courant de la négociation "tooLate", il se rend donc compte que le message arrive trop tard et ne le prend pas en compte.

Pour fixer les idées, voici deux exemples de messages :
- si Jean veut envoyer une proposition de contrat, le message sera composé de $\{Jean, jean\#1, p1, PROPOSE, \{c\}\}$ où $jean\#1$ est l'identifiant du contrat, $p1$ l'état courant de la négociation et c le contrat proposé.
- Si un participant, Pierre par exemple, refuse la proposition de contrat de Jean, le message sera de la forme $\{Pierre, jean\#1, p1, REFUSE, \{\}\}$, où $jean\#1$ est l'identifiant du contrat refusé par Pierre et $p1$ l'état de la négociation chez Pierre.

4.1.2 Le serveur de noms

Le serveur de noms (`NamesServer`) permet de conserver la liste des participants selon le type d'application de négociation qu'ils rejoignent ainsi que les ressources impliquées dans les négociations selon le type d'application de négociation auquel elles appartiennent. Le serveur conserve également l'adresse de chaque participant et leur attribue une boîte aux lettres pour stocker leur courrier lorsqu'ils sont déconnectés.

Afin de conserver les ressources, le serveur de noms utilise un `ResourceWarehouse` qui permet d'ajouter de nouvelles ressources à celles déjà présentes pour un type d'application donné (en évitant les doublons) et qui permet

également de récupérer la liste des ressources enregistrées pour un type d'application de négociation donné.

La méthode `receiveFromAgent` est responsable de la réception d'un message de type `ServerMessage` de la part d'un agent. Quatre messages différents peuvent être envoyés au serveur, qui font appel aux quatre méthodes suivantes :

- `subscribe(name, address, application-type, resources)` : cette méthode permet à un agent négociateur de s'inscrire auprès du serveur. L'agent donne son nom logique (pseudo), son identifiant physique (adresse IP, mail, etc.), le type d'application de négociation dans laquelle il veut participer (enchères, prise de rendez-vous, etc.) et ses ressources personnelles qu'il va négocier,
- `connect(name, address)` : cette méthode permet à un agent déjà inscrit de se connecter au serveur. Elle est appelée après une déconnexion, c'est pourquoi il faut indiquer l'identifiant qui permettra au serveur de joindre l'agent,
- `disconnect(name)` : cette méthode permet à un agent connecté de se déconnecter du serveur, ce qui le rend indisponible pour négocier
- `sendMyMessage(to,msg)` : cette méthode est invoquée par un agent négociateur pour envoyer un message de type `NegotiationMessage` à un ensemble de participants. Si un destinataire n'est pas connecté, le message est placé dans une boîte aux lettres et sera envoyé au destinataire lors de sa prochaine connexion.

Pour faire fonctionner une application, le développeur doit définir un agent qui joue le rôle du serveur de noms.

4.1.3 L'interface Communicator

```
package fr.lifl.genca.communication;
public interface Communicator {

    /** for an agent to send a message to the names server
        @param msg the message to send
    **/
    void sendToServer(ServerMessage msg);

    /** to send a message to one participant (synchronous). Must call
        the method receiveMessage of the Negotiator of the agent.
        @param to the user/agent address
        @param msg the content of the mesage
    **/
    void sendToAgent(Object to,ServerMessage msg);
}
```

FIG. 4.1 – *L'interface Communicator*

Chaque agent a besoin d'un communicateur pour interagir avec le serveur de noms, qui a lui aussi besoin d'un communicateur lui permettant d'envoyer un message à un agent, connaissant son adresse. Ces communicateurs doivent implémenter l'interface *Communicator* (Figure 4.1). Cette interface comprend deux méthodes, l'une utilisée par les agents négociant pour interagir avec le serveur de noms et l'autre utilisée par le serveur de noms pour joindre un agent négociant.

Pour envoyer un message au serveur, l'agent utilise la méthode `sendToServer` de l'interface `Communicator`. Le message est obligatoirement du type `ServerMessage`.

Pour envoyer un message à un agent, le serveur utilise la méthode `sendToAgent` de l'interface `Communicator`. Le message envoyé est obligatoirement du type `ServerMessage`.

```
package fr.lifl.genca.magique;
public class MagiqueCommunicator extends MagiqueDefaultSkill
implements Communicator {

    private Agent myAgent;

    public MagiqueCommunicator(Agent name){
        super(name);
        myAgent=name;
    }

    /** for an agent to send a message to the server
        @param message the message to send
    */
    public void sendToServer(ServerMessage message) {
        perform("receiveFromAgent",message);
    }

    /** for the server to send a message to an agent (synchronous)
        @param to the user/agent address
        @param msg the content of the mesage
    **/
    public void sendToAgent(Object to,
                            fr.lifl.genca.communication.Message msg){
        askNow((String)to,"receiveMessage",msg);
    }
}
```

FIG. 4.2 – *L'implémentation de l'interface Communicator utilisant la plate-forme Magique*

Plusieurs implémentations de l'interface *Communicator* sont fournies avec *GeNCA*, parmi lesquelles les classes :

- `fr.lifl.genca.magique.MagiqueCommunicator` qui permet d'utiliser *GeNCA* au sein de la plate-forme Magique [Bensaid, 1999, mag,],
- `fr.lifl.genca.madkit.MadkitCommunicator` qui permet d'utiliser *GeNCA* au sein de la plate-forme Madkit [mad,],
- `fr.lifl.genca.centralised.CentralisedCommunicator` qui permet d'utiliser *GeNCA* indépendemment de toute plate-forme avec des agents threadés qui communiquent à tour de rôle (round robin),
- `fr.lifl.genca.mail.MailCommunicator` qui permet d'utiliser *GeNCA* avec des agents communiquant par envoi d'e-mails.

Pour les plateformes multi-agents Magique et Madkit, le communicateur utilise les primitives de communication fournies par celles-ci. Prenons l'exemple de la plate-forme Magique (Figure 4.2), les primitives de communication entre agents *perform* et *askNow* sont utilisées. La méthode *perform* est utilisée ici pour communiquer avec le serveur de noms, elle lui demande d'exécuter la méthode passée en paramètre sous la forme d'une chaîne de caractères avec l'argument fourni. La méthode *askNow* est employée par le serveur pour envoyer un message à un agent. Elle consiste en l'appel synchrone de la méthode *receiveMessage* définie dans l'agent négociant.

Pour la version des agents threadés parlant tour à tour, la communication se fait par simple invocation de méthode des agents qui sont alors des objets. Le communicateur contient une boîte aux lettres de courrier à envoyer et en délivre un par tour de parole. Le programme principal contient une liste d'agents et s'occupe de les faire parler chacun leur tour, c'est-à-dire leur demande à tour de rôle s'ils ont un message à envoyer.

Afin de communiquer par envoi d'e-mails, le communicateur est scindé en deux parties, l'une pour l'envoi de messages et l'autre pour la réception. Le communicateur `fr.lifl.genca.mail.MailCommunicator` s'occupe de l'envoi des messages et possède une instance de la classe `fr.lifl.genca.mail.PopMailWatcher` qui lit les messages dans la boîte aux lettres en entrée (inbox) de l'agent. Il est donc nécessaire de fournir les paramètres de configuration des e-mails tels que l'adresse e-mail de l'agent, celle du serveur de noms, le nom du serveur de mail utilisé pour envoyer les messages, celui du serveur pop où les messages seront récupérés ainsi que le nom d'utilisateur et le mot de passe associé et enfin le délai entre deux relèves des e-mails. La Figure 4.3 présente un exemple d'un tel fichier de configuration en XML.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE genca SYSTEM "mailProperties.dtd" >
<mailProperties>
  <SMTP-Properties>
    <SMTP>
      smtp.provider.fr
    </SMTP>
    <myAddress>
      myName@provider.fr
    </myAddress>
  </SMTP-Properties>
  <POP-Properties>
    <POP>
      pop.provider.fr
    </POP>
    <username>
      myName
    </username>
    <password>
      motDePasse
    </password>
    <popDelay>
      60
    </popDelay>
  </POP-Properties>
  <serverAddress>
     genca@provider.fr
  </serverAddress>
</mailProperties>
```

FIG. 4.3 – *Fichier de configuration des paramètres nécessaires à l'utilisation des e-mails.*

4.2 Implémentation du niveau négociation

Toute réalisation informatique nécessite une structure de données adaptée. Il faut ici pouvoir coder des négociations simultanées, l'avancement d'une négociation, les initiateurs et les participants et principalement les ressources et les contrats.

Pour implémenter notre protocole, nous avons défini deux types d'objets communs à toute forme de négociation : les objets de description de la négociation et les objets pour la dynamique de la négociation. Nous avons également défini des objets pour la mémorisation des négociations passées. Ces objets se trouvent dans le paquetage `fr.lifl.genca.negotiation`.

4.2.1 Les objets de description de la négociation

Les ressources Les ressources sont les objets que chaque agent essaie d'obtenir. Les ressources sont abstraites et peuvent aussi bien représenter des articles que des tranches horaires ou encore des cageots de fruits. N'importe quel objet peut être une ressource, incluant un contrat comme type de donnée, mais pas le contrat en train d'être négocié lui-même. Nous avons choisi de représenter les ressources par une chaîne de caractères, ce qui présente l'avantage de pouvoir détailler la ressource. Ainsi "table", "lundi 15 mars 2004, 8h-9h", ou encore "pommes de terre" peuvent être des ressources.

Les contrats Un *contrat* est un objet créé à l'initialisation de la négociation. Il comporte les *ressources* à négocier, l'*initiateur* de la négociation, le *délai* de réponse pour éviter les deadlocks et la *réponse par défaut* qui sera considérée par l'initiateur si le participant ne répond pas dans les temps. Par exemple, l'association {Pierre, "lundi 15 mars 2004, 8h-9h", 10, "accepte"} est un contrat proposé par Pierre pour la ressource "lundi 15 mars 2004, 8h-9h"dont le délai de réponse est 10 minutes et la réponse par défaut un accord.

Un contrat peut contenir une ou plusieurs ressources. Si un contrat porte sur plusieurs ressources, celles-ci sont liées par l'opérateur logique **ET**. Cela implique que l'obtention de toutes les ressources est nécessaire pour confirmer le contrat, les ressources forment un tout. Si toutes les ressources ne peuvent pas être obtenues, la négociation échouera. Au contraire, plusieurs contrats sur une seule ressource signifie que l'agent veut obtenir les ressources séparément, c'est-à-dire que si l'agent n'arrive pas à obtenir l'une des ressources, ce n'est pas grave, toutes les négociations (une par contrat) sont indépendantes, l'échec de l'une n'a pas d'importance sur la réussite ou l'échec d'une autre.

Le contrat ne contient pas l'ensemble des participants de sorte à préserver leur "vie privée" et donc un participant ne sait pas s'il y a d'autres participants à la négociation et n'a pas connaissance des messages qu'ils envoient à l'initiateur.

Le contrat est l'entité échangée entre l'initiateur et les participants, il est à la base de la négociation.

Les propriétés du contrat Les *propriétés* du contrat sont connues uniquement de l'initiateur, c'est lui qui les définit à la création du contrat. Elles sont identifiées par l'identifiant du contrat dont elles sont les propriétés et sont constituées de la liste des *participants* à la négociation, du *nombre minimal d'accords* pour confirmer le contrat, du *nombre de tours* dans la négociation et du *nombre de renégociations autorisées*. Cet objet mémorise également les participants qui ont accepté le contrat, le nombre d'accords reçus et le nombre de tours et le nombre de renégociations effectuées. Cela permet à l'initiateur de mémoriser les paramètres de la négociation du contrat qu'il a défini.

Les contrats pris par l'agent Cette structure de donnée permet de mémoriser les contrats qui ont été pris par l'agent sur les ressources. Elle représente la partie inférieure de la représentation des négociations que nous avons définie dans le chapitre précédent en page 87.

4.2.2 Les objets pour la dynamique de la négociation

Le négociateur C'est la classe principale de notre modèle. C'est le négociateur qui gère toutes les négociations de l'agent. Il est donc le seul à posséder tous les contrats proposés, toutes les propriétés des contrats qu'il a initié et les résultats des négociations.

C'est donc naturellement lui qui fournit des indications sur les négociations passées, informations utiles pour la création de stratégies.

C'est également lui qui gère les priorités des ressources et des participants. Il peut donc fournir les ressources dans l'ordre de priorité, les ressources libres et dans l'ordre de priorité, les ressources libres ou prises par quelqu'un de moins important et dans l'ordre de priorité. Il peut aussi indiquer si un ensemble de ressources est libre ou non et fournir une représentation textuelle des ressources.

De même, en comparant la priorité des participants, il indique si un contrat est plus prioritaire que celui ou ceux déjà pris pour les mêmes ressources.

Il reçoit les messages de la part du serveur et les traite, si c'est le message en retour de l'inscription de l'agent, il récupère les participants et les ressources présentes dans le système, si c'est une arrivée de participants, il l'ajoute à la liste des participants, si c'est un message de négociation, il le délègue au micro-agent concerné.

Les micro-agents Chaque agent négociateur est constitué de micro-agents appelés buts et engagements. Les buts sont créés chez les initiateurs de contrat tandis que les engagements sont créés chez les participants à une négociation. Les micro-agents sont des agents purement réactifs à l'intérieur de l'agent négociateur. Chaque micro-agent est responsable de la négociation du contrat pour lequel il a été créé. Il peut y avoir simultanément plusieurs buts, comme il peut y avoir plusieurs engagements au sein d'un agent négociateur. En fait, il y a autant de buts qu'il y a de contrats créés par l'agent et qui sont en cours de négociation et il y a autant d'engagements qu'il y a de négociations en cours où l'agent intervient en tant que participant. Chaque micro-agent interagit avec la stratégie qui lui a été attribuée. À chaque arrivée de message concernant la négociation de son contrat, le micro-agent informe la stratégie qui prend en compte le message. Celle-ci indique par la suite au micro-agent le message qu'elle souhaite envoyer en réponse. Prenons l'exemple d'une proposition de contrat arrivant à un micro-agent engagement, celui-ci envoie la proposition à la stratégie qui lui demande ensuite d'envoyer une acceptation ou un refus.

Le but est responsable de la mise à jour des propriétés du contrat, c'est-à-dire qu'il compte les accords et les refus, mémorise les personnes ayant accepté la proposition faite par l'agent et gère les délais de réponse et le nombre de tours de parole des participants ainsi que le nombre de renégociations effectuées.

La matrice de gestion des négociations Il s'agit de la matrice présentée dans le chapitre précédent (page 87) et qui correspond à la partie supérieure de la représentation des négociations de l'agent. Le principe de cette matrice dont les colonnes correspondent aux ressources est que les contrats qui arrivent s'empilent dans les colonnes qui correspondent aux ressources qu'il contient. Les contrats qui sont à la base de la matrice sont en cours de négociation tandis que les autres attendent la libération des ressources pour entamer leur processus de négociation.

Les objets de mémorisation des négociations

Les statistiques sur les négociations passées Nous avons défini une structure de donnée qui permet de mémoriser pour un couple (participant, ressource) le nombre de contrats que l'agent lui a proposé, le nombre de contrats pris avec ce participant ainsi que le nombre de rétractations de ce participant pour un contrat qu'il avait pris avec l'agent sur la ressource concernée. Cela nous permet de calculer un taux de succès et un taux de rétractation par couple (participant, ressource).

Les propositions reçues Cette structure de données mémorise les propositions de contrat reçues par l'agent. Elle permet de savoir quelles ressources ont été proposées

par une personne donnée ou alors quelles personnes ont fait des propositions pour une ressource donnée. Ainsi, il est possible de raisonner sur la demande des ressources et de savoir laquelle est la plus (resp. moins) demandée.

4.3 Implémentation du niveau stratégique

Comme nous l'avons vu dans le chapitre précédent, il y a deux sortes de stratégies : celle pour un initiateur et celle pour un participant. On retrouve donc deux interfaces dans l'API *GeNCA* : l'interface *fr.lifl.genca.strategy.InitiatorStrategy* (Figure 4.4) et l'interface *fr.lifl.genca.strategy.ParticipantStrategy* (Figure 4.5), qui contiennent chacune les méthodes correspondant aux décisions qui doivent être prises selon le rôle qu'elle représente.

Pour le côté initiateur, il y a deux types de méthodes : celles qui réagissent à un message provenant d'un participant et celles qui prennent les décisions après avoir reçu toutes les réponses des participants. La stratégie d'un initiateur doit donc réagir face à l'acceptation de la proposition faite à un participant, cette acceptation pouvant fournir des paramètres à l'initiateur. Par exemple, dans le cas d'enchères à offres scellées, il faut proposer un prix à l'initiateur si l'on accepte la proposition qui consiste en la mise en vente d'un article. En revanche, comme le but s'occupe de compter le nombre d'accords et de refus et mémorise les participants qui ont accepté la proposition, il n'y a pas de méthodes pour traiter le refus d'un participant. Lorsque toutes les réponses sont arrivées (ou lorsque le délai de réponse a expiré), le but demande à la stratégie ce qu'elle décide pour la suite de la négociation (méthode *decide()* ou *defaultDecision(...)* s'il manque des réponses), c'est-à-dire soit la confirmation du contrat, soit l'annulation du contrat, soit une demande de modification ou encore une nouvelle proposition de contrat. Si la stratégie décide de demander une modification aux participants, la méthode *initModification()* sera invoquée, ce qui permettra à la stratégie de préparer la réception des modifications (initialisation de variables, par exemple). Puis, à chaque proposition de modification reçue de la part d'un participant, la stratégie en sera informée et pourra alors traiter cette modification (méthode *judge(...)*). Une fois toutes les propositions de modification reçues (ou expiration du délai de réponse), le but invoque la méthode *decideModification()* (ou *defaultModification()* s'il manque des propositions). La stratégie décide alors la suite à donner à la négociation, qui est soit la proposition d'un nouveau contrat, soit une autre demande de modification, soit l'annulation du contrat.

Pour le côté participant, il n'y a que deux méthodes : celle qui répond à une proposition de contrat (*whatDoIanswer*) et celle qui propose une modification à l'initiateur (*proposeModification*).

```
package fr.lifl.genca.strategy;
public interface InitiatorStrategy {

    /**
    * Initialisation method that replaces a default constructor.
      @param n the negotiator of the agent
      @param g the goal responsible for the negotiation
      @param cp the properties of the contract negotiated
      @param nbRes the number of resources
      @param lg the length of the modification
    */
    void init(Negotiator n, Goal g, ContractAndProperties cp, int nbRes, int lg);

    /** method invoked at reception of an <i>accept</i> message
        @param from the participant who agreed
        @param params the parameters
    **/
    void receivedAccept(String from,Object[] params);

    /** method invoked when all answers are received **/
    void decide();

    /** method invoked when the timer has expired and all answers
           have not been received
        @param whoHasAnswered the list of participants who have answered
    **/
    void defaultDecision(Vector whoHasAnswered);

    /** method invoked before requesting modification at participants **/
    void initModification();

    /** method invoked when a modification has been proposed
        @param from the participant who proposed the modification
        @param params the modification
    **/
    void judge(String from,Object[] params);

    /** method invoked when all modification propositions
           have been received **/
    void decideModification();

    /** method invoked when the timer has expired and all
          modification propositions have not been received **/
    void defaultModification();
}
```

FIG. 4.4 – *Interface pour la stratégie côté initiateur*

```
package fr.lifl.genca.strategy;
public interface ParticipantStrategy {

    /** Initialisation method that replaces a default constructor.
        @param n the negotiator of the agent
        @param e the engagement responsible for the negotiation
        @param c the contract negotiated
        @param nbRes the number of resources
        @param lg the length of the modification
    */
    void init(Negotiator n, Engagement e, Contract c, int nbRes, int lg);

    /** method invoked at reception of a contract proposition
        @param c the proposed contract
    **/
    void whatDoIanswer(Contract c);

    /** method invoked at reception of modification request
        @return the proposed modification
    **/
    Object[] proposeModification();
}
```

FIG. 4.5 – *Interface pour la stratégie côté participant*

Pour la réponse à une proposition de contrat, la stratégie évalue ce contrat grâce aux outils fournis, puis décide soit de l'accepter et donc demande à l'engagement d'envoyer le message *accept*, soit de le refuser et donc demande à l'engagement d'envoyer le message *refuse*.

Pour la proposition d'une modification, la stratégie utilise à nouveau les outils fournis afin de choisir la modification qui la satisferait au mieux. Elle demande ensuite à l'engagement de l'envoyer à l'initiateur (message *proposeModification*).

Pour chaque application nécessitant une stratégie spécifique, le développeur doit implémenter ces deux interfaces. Des exemples d'implémentations seront développés dans la section Applications.

4.4 Interface graphique

Notre but est de fournir un système qui pourra être utilisé de façon automatique ou guidée par un utilisateur humain. Nous voulons en effet qu'un utilisateur puisse gérer ses négociations via notre modèle et dynamiquement laisser la main à son agent lorsqu'il doit s'absenter pour participer à une réunion par exemple et la reprendre à

son retour. Afin de pouvoir interagir avec son agent, une interface graphique est proposée à l'utilisateur. Nous avons conçu une interface utilisateur générique afin qu'elle s'adapte aux différentes applications qui seront réalisées. A travers cette interface, l'utilisateur peut créer des contrats, suivre l'avancement de ses négociations et répondre aux propositions de contrat s'il utilise *GeNCA* en mode manuel. Toutes les classes nécessaires à l'interface graphique se trouvent dans le paquetage `fr.lifl.genca.gui`. Nous décrivons ici chaque onglet de l'interface graphique.

4.4.1 L'onglet Paramètres

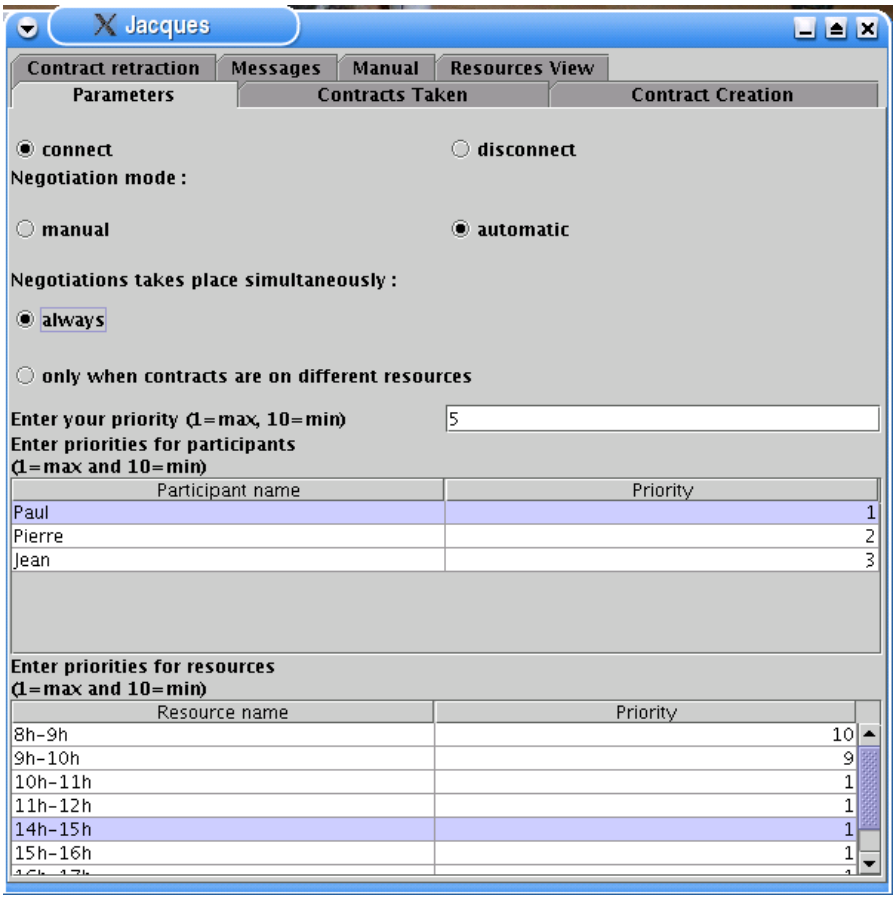


FIG. 4.6 – *Onglet Paramètres*

Cet onglet est celui sur lequel l'interface graphique se lance. C'est un onglet de configuration, il permet de se connecter/déconnecter du système (au départ, l'agent est connecté), de choisir le mode d'utilisation (manuel ou automatique), de choisir la manière d'effectuer les négociations (toutes simultanées ou séquentielles lorsque les

mêmes ressources interviennent dans plusieurs contrats). C'est aussi dans cet onglet que l'utilisateur définit sa priorité et les priorités des ressources et celles des participants. Ces priorités sont utilisées par les stratégies de négociation. Si l'utilisateur choisit le mode manuel d'utilisation, il devra alors répondre lui-même aux propositions de contrat qui lui seront faites, par le biais de l'onglet manuel (voir sous-section 4.4.4).

La Figure 4.6 montre cet onglet pour l'agent Jacques. Jacques est connecté au serveur, il utilise le mode automatique d'utilisation et négocie tous les contrats simultanément. Il s'est défini une priorité de 5 (milieu de l'intervalle de priorités).

Trois personnes sont présentes pour l'application : Paul, Pierre et Jean. Jacques leur a donné la priorité 1, 2 et 3 respectivement. Ils sont donc tous les trois plus prioritaires que Jacques.

Plusieurs ressources sont présentées. On voit que la ressource "8h-9h" a une priorité de 10, la ressource "9h-10h" a une priorité de 9 et les autres ont une priorité qui vaut 1. Ceci indique que Jacques préfère prendre des rendez-vous après 10h, car 10 est la moins bonne note possible pour les ressources.

4.4.2 L'onglet Création de contrat

Cet onglet permet de créer un contrat et de le proposer à un ensemble de participants. Il faut choisir les ressources que l'on va négocier et les participants avec lesquels le contrat va être négocié. Les différents paramètres de négociation sont prédéfinis avec les valeurs données par défaut dans le fichier de configuration du protocole. Ces paramètres peuvent être modifiés dans cet onglet pour la négociation de ce contrat. Parmi ces paramètres, on trouve le délai d'attente des réponses (en minutes), la réponse par défaut (celle qui sera considérée si les participants ne répondent pas), le nombre minimal d'accords nécessaires pour que le contrat soit pris (qui peut être un pourcentage), le nombre de tours de parole des participants et enfin le nombre maximal de renégociations qui peuvent être demandées. Une fois ces paramètres entrés, il n'y a plus qu'à créer le contrat en cliquant sur le bouton *create*. Celui-ci sera alors transmis aux participants et la négociation entamée.

La Figure 5.1 montre l'onglet de création de contrat de l'agent Jean. On y retrouve les valeurs par défaut des paramètres de négociation. La réponse par défaut est un refus, les participants doivent tous être d'accords pour que le contrat soit confirmé (100%), les participants ont 10 minutes pour répondre et 20 tours de parole leur sont accordés et le contrat peut être renégocié 3 fois.

Si l'utilisateur a oublié de choisir une (des) ressource(s) et/ou un (des) participant(s), une boîte de dialogue s'ouvre et l'indique à l'utilisateur pour qu'il les spécifie.

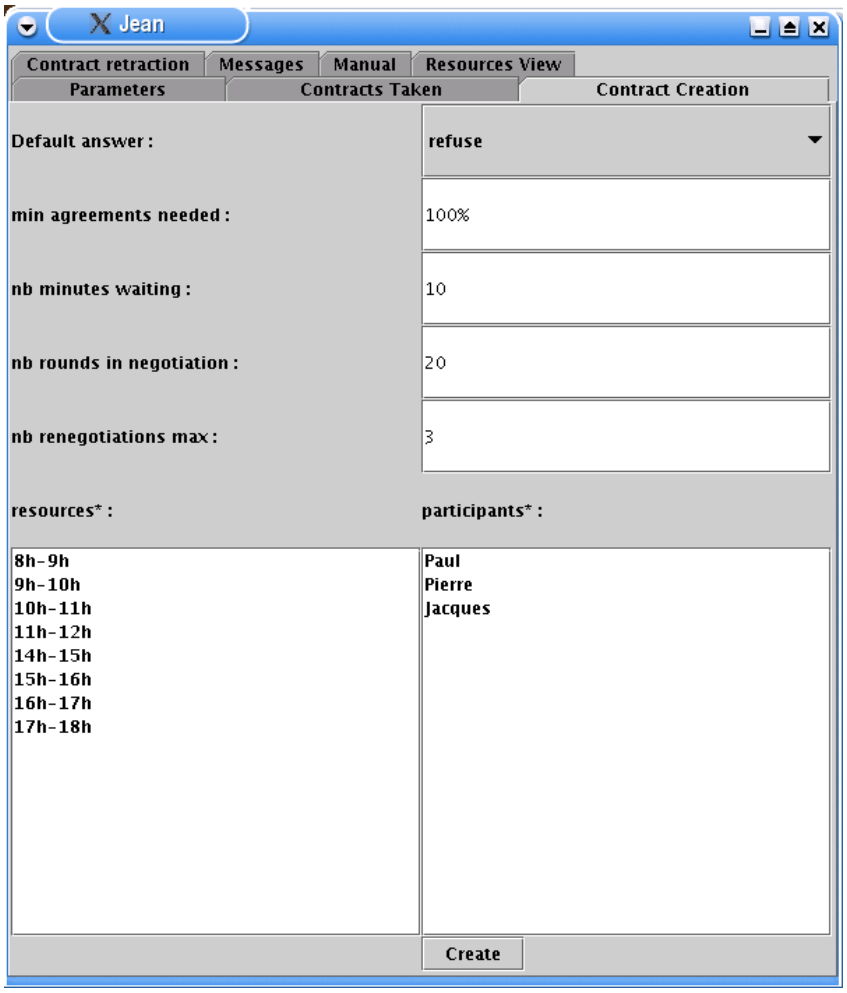


FIG. 4.7 – *Onglet Création de contrat*

4.4.3 L'onglet Messages

Grâce à cet onglet, l'utilisateur visualise les différents messages envoyés et reçus par l'agent. Lors d'une négociation où l'utilisateur est l'initiateur, il peut voir qui a répondu à sa proposition, quelles sont les modifications proposées, etc.. L'utilisateur voit également les propositions de contrat qui lui sont faites et ce que son agent a répondu. Il visualise également les changements de priorité qu'il a effectué et l'arrivée de nouveaux participants. Lorsque l'agent n'a pas de négociation en cours, un label spécial écrit en rouge s'affiche pour prévenir l'utilisateur qu'il peut quitter son agent sans que cela pose de problèmes.

La Figure 4.8 montre cet onglet pour l'agent Jacques. On voit que Jacques s'est inscrit le 3 mars 2004 à 17h30, que 2 personnes étaient inscrites avant lui et que Jean est arrivé après lui. Jacques a changé les priorités des trois personnes inscrites.

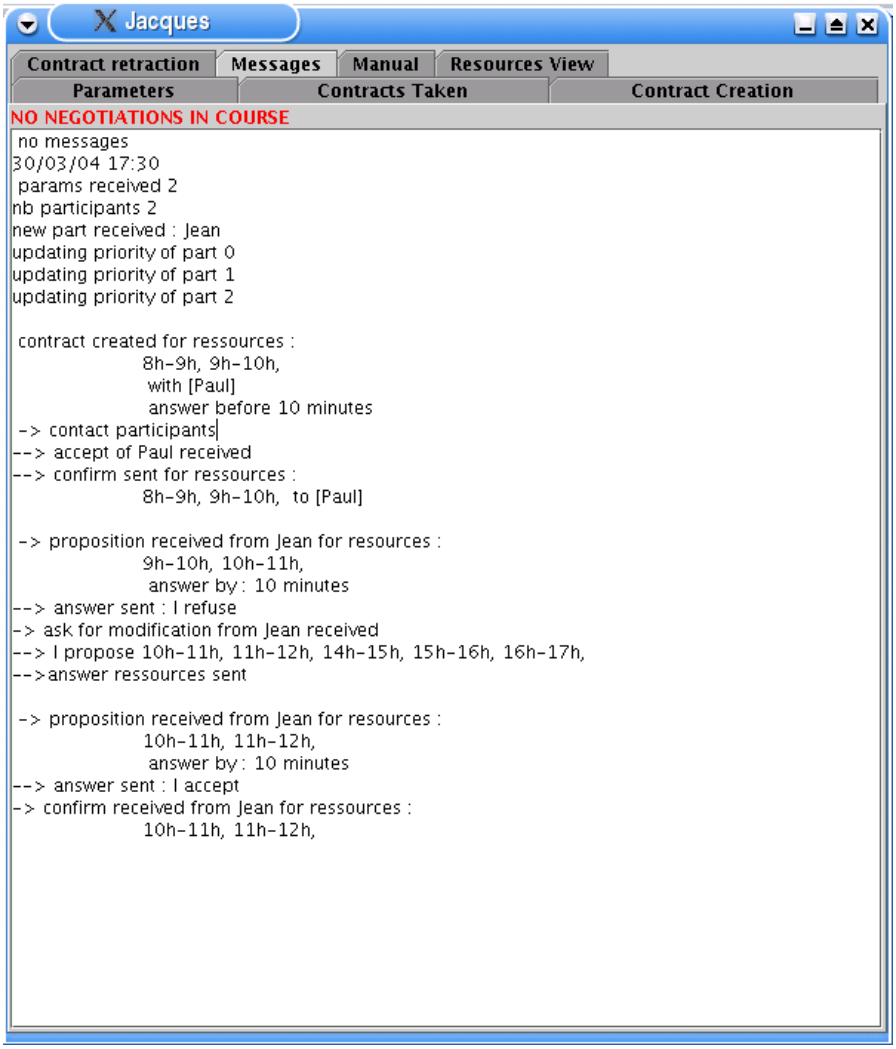


FIG. 4.8 – *Onglet Messages*

Jacques a créé un contrat avec Paul pour les ressources “8h-9h” et “9h-10h”. Paul a accepté la proposition et Jacques a confirmé ce rendez-vous.

Il a ensuite reçu une proposition de Jean pour les ressources “9h-10h” et “10h-11h”, qu’il a refusé. Jean a demandé une modification du contrat et Jacques lui a proposé les ressources “10h-11h”, “11h-12h”, “14h-15h”, “15h-16h” et “16h-17h”. Jean a alors proposé un nouveau contrat pour les ressources “10h-11h” et “11h-12h”. Jacques a accepté cette proposition et Jean a confirmé ce rendez-vous.

Au moment où la capture d’écran a été faite, Jacques n’avait plus de contrats en cours de négociation.

4.4.4 L'onglet Manuel

FIG. 4.9 – *Pop-up proposition reçue*

Cet onglet concerne le mode manuel d'utilisation. Lorsque ce mode est activé, c'est l'utilisateur qui répond aux propositions de contrat. Lorsqu'une proposition de contrat arrive, une boîte de dialogue s'ouvre et indique à l'utilisateur cette arrivée (Figure 4.9). L'interface se positionne automatiquement sur l'onglet manuel qui affiche la proposition de contrat et récupère la réponse de l'utilisateur.

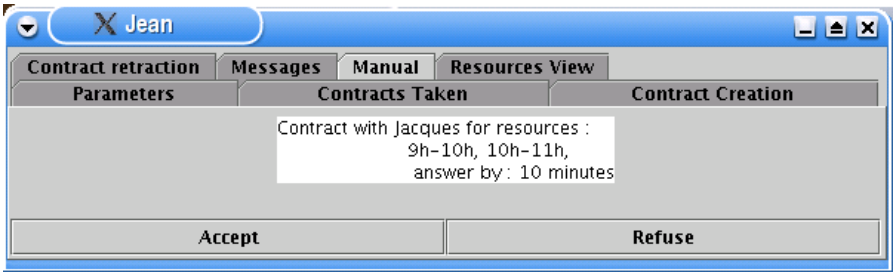


FIG. 4.10 – *Onglet Mode Manuel*

La Figure 4.10 montre l'onglet manuel de l'agent Jean. Une proposition de contrat lui est faite par Jacques pour les ressources "9h-10h" et "10h-11h", il a 10 minutes pour répondre. Si l'utilisateur accepte le contrat alors qu'un autre contrat a déjà été pris sur les ressources concernées, l'utilisateur est prévenu et il doit confirmer son choix (Figure 4.11).

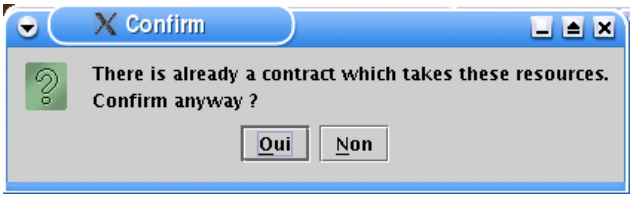


FIG. 4.11 – *Pop-up conflit de contrats*

4.4.5 L'onglet Contrat pris

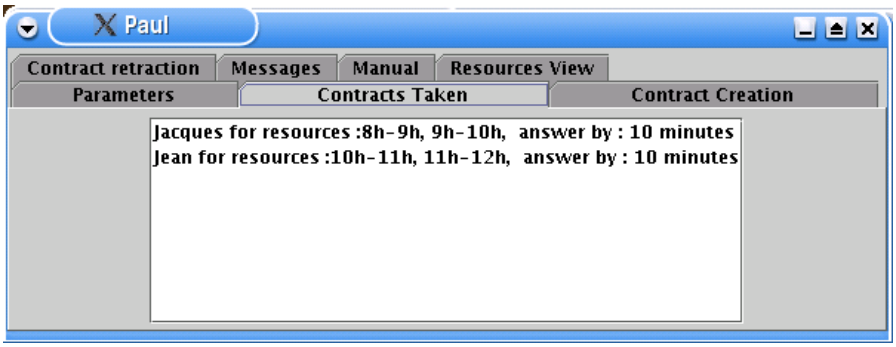


FIG. 4.12 – *Onglet Contrat pris*

Cet onglet permet à l'utilisateur de visualiser les contrats qu'il a pris avec l'initiateur et les ressources négociées.

La Figure 4.12 montre cet onglet pour l'agent Paul. Il a déjà pris deux contrats : l'un avec Jacques pour les ressources "8h-9h" et "9h-10h" ; l'autre avec Jean pour les ressources "10h-11h" et "11h-12h".

4.4.6 L'onglet Rétractation

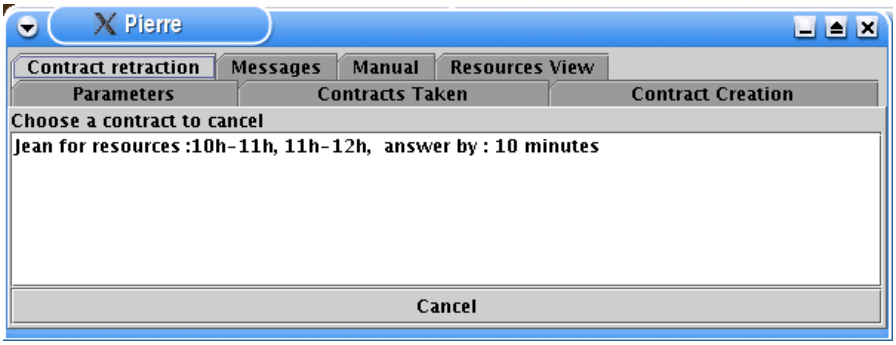


FIG. 4.13 – *Onglet Rétractation*

Cet onglet permet de se rétracter d'un contrat pris précédemment. Pour cela, il suffit de choisir dans la liste le contrat à annuler et de valider en cliquant sur le bouton *cancel*. Le contrat sera alors annulé auprès de l'initiateur l'ayant proposé.

La Figure 4.13 montre cet onglet pour l'agent Pierre. Il n'a qu'un seul contrat qu'il a pris avec Jean pour les ressources "10h-11h" et "11h-12h".

4.4.7 L'onglet Vue des ressources

Cet onglet affiche une représentation des ressources et des contrats négociés ou en cours de négociation sur celles-ci. Les ressources sont alignées en bas de l'onglet. Au dessous, on retrouve les contrats pris sur les ressources et au dessus, on empile les contrats en cours de négociation sur les ressources en question.

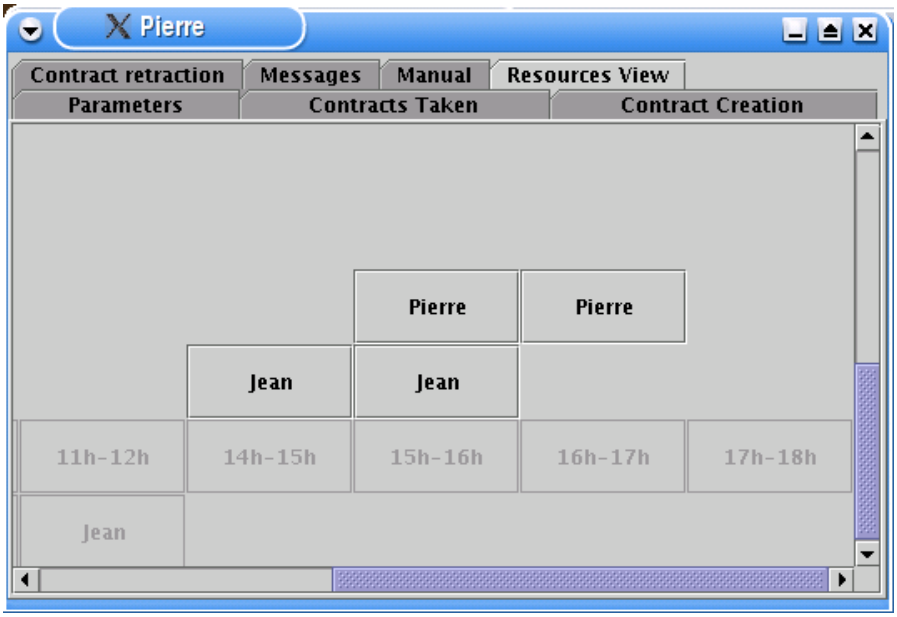


FIG. 4.14 – *Onglet Vue des ressources*

La Figure 4.14 montre cet onglet pour l'agent Pierre. On voit que Pierre a déjà pris un contrat avec Jean pour la ressource "11h-12h" et qu'il est en train de négocier un autre contrat proposé par Jean pour les ressources "14h-15h" et "15h-16h" et qu'il a créé un contrat sur les ressources "15h-16h" et "16h-17h".

En cliquant sur un contrat en cours de négociation, une fenêtre s'ouvre et indique l'état de la négociation à ce moment. La Figure 4.15 montre l'état de la négociation de Jean pour les ressources "14h-15h" et "15h-16h" avec Paul, Pierre et Jacques. Jean a reçu les réponses de Pierre et de Paul.

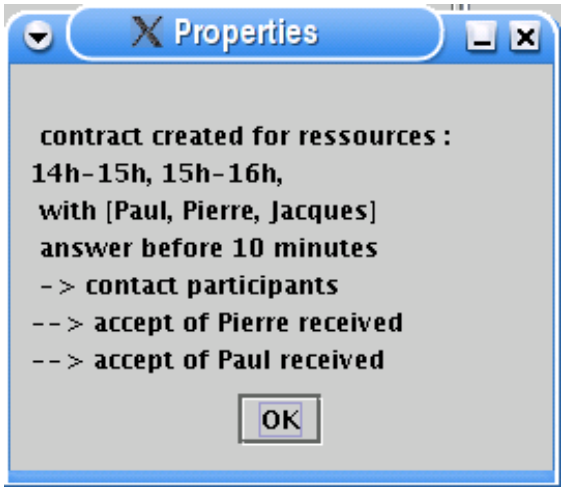


FIG. 4.15 – *État de la négociation*

4.5 Utilisation du paquetage pour une application

Le paquetage que nous fournissons implémente intégralement la couche de négociation et le serveur de noms de la couche de communication et donne des implémentations par défaut des interfaces des couches de communication et de stratégie.

Les implémentations de la couche de communication que nous fournissons permettent d'utiliser les plates-formes Magique et Madkit, ainsi que des agents centralisés parlant à tour de rôle et des agents communiquant par e-mail. Nous fournissons également l'agent système auprès duquel les agents des utilisateurs s'enregistrent et qui est responsable de l'envoi des messages. Pour ces quatre modes d'utilisation (Magique, Madkit, round robin et e-mail), une classe permettant de lancer l'application est fournie. Pour toute autre mode de communication, il est nécessaire d'implémenter l'interface *Communicator* et d'avoir un agent intégrant le serveur de noms implémenté dans le paquetage. Les stratégies implémentées sont celles décrites dans la section 3.4.

Le paquetage fournit également l'interface graphique présentée dans la section précédente. Dans notre paquetage, l'utilisateur humain a deux façons d'utiliser son agent. Manuellement, c'est alors un outil d'aide à la décision qui montre l'état de toutes les négociations en cours et, dans ce cas, c'est l'utilisateur humain qui répond à une proposition de contrat. Automatiquement, cette fois l'agent est caché et répond lui-même aux propositions sans intervention de l'utilisateur.

Afin de configurer l'application de négociation et plus particulièrement les agents, deux types de fichiers XML sont utilisés : l'un est commun à tous les agents, l'autre est propre à chaque agent et optionnel. Afin de valider ces fichiers de configuration,

```
<!ELEMENT genca (application-type,resources-list,communicator,
default-initiator-strategy,default-participant-strategy,protocol,management,
default-priority,window,application-parameters-list?)>
<!ELEMENT application-type (#PCDATA)>
<!ELEMENT resources-list (resource)*>
<!ELEMENT resource (#PCDATA)>
<!ELEMENT communicator (#PCDATA)>
<!ELEMENT default-initiator-strategy (#PCDATA)>
<!ELEMENT default-participant-strategy (#PCDATA)>
<!ELEMENT protocol (answer-delay,default-answer,minAgreements,
nbRounds,nb-modifications-by-round,retraction-allowed,
nbRenegotiations)>
<!ELEMENT answer-delay (#PCDATA)>
<!ELEMENT default-answer EMPTY>
<!ATTLIST default-answer value (accept | refuse) "refuse">
<!ELEMENT minAgreements (#PCDATA)>
<!ELEMENT nbRounds (#PCDATA)>
<!ELEMENT nb-modifications-by-round (#PCDATA)>
<!ELEMENT retraction-allowed EMPTY>
<!ATTLIST retraction-allowed value (true | false) "true">
<!ELEMENT nbRenegotiations (#PCDATA)>
<!ELEMENT management EMPTY>
<!ATTLIST management value (sequential | parallel) "sequential">
<!ELEMENT default-priority EMPTY>
<!ATTLIST default-priority value (1|2|3|4|5|6|7|8|9|10 ) "5">
<!ELEMENT window EMPTY>
<!ATTLIST window value (true | false) "true">
<!ELEMENT application-parameters-list (application-parameter)*>
<!ELEMENT application-parameter (name,parameter)*>
<!ELEMENT name (#PCDATA)>
<!ELEMENT parameter (class,value)>
<!ELEMENT class (#PCDATA)>
<!ELEMENT value (#PCDATA)>
```

FIG. 4.16 – *Fichier DTD pour la configuration d'une application utilisant GeNCA. On y indique entre autres le type de l'application, les ressources communes aux participants, le paramétrage du protocole, le moyen de communication utilisé par les agents ainsi que les valeurs par défaut pour le mode de gestion des négociations et les stratégies utilisées pour négocier*

deux DTDs ont été conçues : `genca.dtd` et `agent.dtd`. Le fichier DTD de configuration de l'application de négociation est présenté en Figure 4.16. Il indique le type de l'application (prise de rendez-vous, enchères hollandaises, etc.), la liste des ressources communes aux différents participants, les paramètres du protocole, le mode de communication entre agents, la gestion par défaut des négociations, les stratégies de négociation utilisées par défaut, la priorité par défaut de l'agent, l'utilisation de l'interface graphique et une liste éventuelle de paramètres nécessaires à l'application. Ce fichier est défini par le concepteur de l'application.

```
<!ELEMENT agent (name,address,resources-list?,initiator-strategy?,
participant-strategy?,protocol?,management?,default-priority?,
window?,application-parameters-list?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT resources-list (resource)*>
<!ELEMENT resource (#PCDATA)>
<!ELEMENT initiator-strategy (#PCDATA)>
<!ELEMENT participant-strategy (#PCDATA)>
<!ELEMENT protocol (answer-delay,default-answer,minAgreements,
nbRounds,nb-modifications-by-round)>
<!ELEMENT answer-delay (#PCDATA)>
<!ELEMENT default-answer EMPTY>
<!ATTLIST default-answer value (accept | refuse) "refuse">
<!ELEMENT minAgreements (#PCDATA)>
<!ELEMENT nbRounds (#PCDATA)>
<!ELEMENT nb-modifications-by-round (#PCDATA)>
<!ELEMENT management EMPTY>
<!ATTLIST management value (sequential | parallel) "sequential">
<!ELEMENT default-priority EMPTY>
<!ATTLIST default-priority value (1|2|3|4|5|6|7|8|9|10 ) "5">
<!ELEMENT window EMPTY>
<!ATTLIST window value (true | false) "true">
<!ELEMENT application-parameters-list (application-parameter)*>
<!ELEMENT application-parameter (name,parameter)*>
<!ELEMENT name (#PCDATA)>
<!ELEMENT parameter (class,value)>
<!ELEMENT class (#PCDATA)>
<!ELEMENT value (#PCDATA)>
```

FIG. 4.17 – *Fichier DTD pour la configuration d'un agent. Il permet de redéfinir les valeurs de certains paramètres de l'application, tels que les stratégies de négociation utilisées, l'utilisation de l'interface graphique, la priorité de l'agent. C'est également dans ce fichier que l'utilisateur indique ses ressources personnelles qu'il veut négocier.*

Le fichier personnel de configuration de l'agent, dont la DTD est présentée en Figure 4.17, permet à un utilisateur de redéfinir les valeurs par défaut de certains paramètres du fichier commun aux agents et d'indiquer les ressources personnelles de

l'agent qui seront négociées. Il peut ainsi définir sa stratégie de négociation et redéfinir le nombre de tours de parole des participants au cours de la négociation par exemple. Il lui est en revanche impossible de redéfinir la possibilité de se rétracter et de renégocier les contrats car cela constitue une caractéristique fixe de l'application.

Pour écrire une application avec ce paquetage, il suffit donc d'implémenter les interfaces de communication et de stratégies qui sont spécifiques à l'application (si les implémentations par défaut ne conviennent pas) et de définir le fichier commun de configuration en XML des agents et bien sûr écrire ses agents en leur incorporant le *négociateur* du paquetage. L'annexe B présente les classes qu'il a fallu écrire pour utiliser *GeNCA* au sein de la plateforme multi-agents Madkit. Lorsque l'application concerne la négociation de contrats portant uniquement sur les ressources, il n'y a rien d'autre à faire. Toute la gestion des négociations se fait automatiquement. En revanche, si le contrat nécessite d'autres paramètres (quantité, prix, etc.), il faut alors étendre la classe *Contrat* et modifier l'interface graphique de création de contrat afin d'y faire apparaître les nouveaux paramètres.

Si l'on compare le travail nécessaire pour réaliser une application de négociation avec *GeNCA* et avec *Zeus*, par exemple, on se rend compte que cela est plus facile avec *GeNCA* [Verrons, 2001]. Premièrement, le protocole de négociation doit simplement être paramétré dans un fichier pour *GeNCA*, tandis que pour *Zeus*, il faut définir les protocole sous la forme d'un automate et considérer le protocole pour l'initiateur et celui pour le participant. Chaque état et chaque arc de ces automates fait ensuite l'objet d'une classe. Le protocole est lui-aussi défini dans une classe qui décrit les états et les arcs de l'automate correspondant. Il faut ensuite modifier une classe pour qu'elle prenne en compte le nouveau protocole. Tout cela est très lourd à gérer et beaucoup de temps doit être consacré à la compréhension du fonctionnement interne de *Zeus*afin de réussir cette définition de protocole.

En ce qui concerne les ressources, elles sont définies dans un fichier pour *GeNCA* et via une interface graphique de création d'application pour *Zeus*. Il n'y a donc pas de différence fondamentale entre *GeNCA* et *Zeus* sur ce point.

Par contre, les stratégies de négociation sont séparées du niveau de négociation dans *GeNCA*, tandis que dans *Zeus*, elles font partie du protocole. En effet, la stratégie est codée par l'automate décrivant le protocole. Chaque état contient une méthode d'exécution et chaque arc représente une condition pour passer d'un état à l'autre. Donc, si l'on désire changer de stratégie, il faut changer l'automate (ou du moins son contenu) du protocole.

Zeus étant une plateforme multi-agents, il n'y a pas de travail à réaliser en ce qui concerne la communication entre les agents, il suffit d'utiliser les méthodes fournies. Ceci ne constitue cependant pas un point fort de *Zeus* par rapport à *GeNCA* puisque celle-ci s'utilise avec différents modes de communication et que deux utilisations de plateformes multi-agents sont fournies.

Il est par conséquent plus aisé de concevoir une application de négociation avec *GeNCA* qu'avec *Zeus*. Nous décrivons maintenant deux applications réalisées avec *GeNCA* afin de montrer sa facilité d'utilisation.

4.6 Conclusion

Nous avons présenté dans ce chapitre l'implémentation sous la forme d'une API Java de notre modèle de négociation, ce qui lui confère la portabilité que nous souhaitions. La structure en trois couches du modèle est respectée et donne lieu à trois paquetages : communication, négociation et stratégie. Un paquetage a été ajouté pour la réalisation d'une interface graphique qui permet à l'utilisateur de gérer ses négociations et d'utiliser *GeNCA* en mode automatique lorsqu'il est absent.

La couche de communication contient une interface qui permet de définir le mécanisme de communication entre les agents qui est spécifique à l'application réalisée. Un serveur de noms est utilisé pour centraliser les ressources qui seront négociées et mémoriser les participants inscrits pour différents types de négociation. Ce serveur est le seul à posséder les adresses (au sens large) des participants qui ne peuvent ainsi pas communiquer directement entre eux afin de former des coalitions. Cette couche définit également le format des messages échangés au sein de *GeNCA* qui peuvent être soit des messages de liaison avec le serveur de noms, soit de négociation entre les agents.

La couche de négociation, qui est la couche principale de notre implémentation, contient toutes les structures de données nécessaires à la gestion des négociations. Elle intègre ainsi des mécanismes permettant d'effectuer les négociations n'entrant pas en conflit sur des ressources simultanément et de séquencialiser les autres. Notre implémentation comprend aussi la possibilité de se rétracter et de renégocier des contrats et évite les deadlocks dans les conversations. Cette couche gérant l'ensemble des négociations d'un agent, c'est naturellement dans celle-ci que l'on trouve les différents outils pour la création de stratégies qui permettent de récupérer des informations sur les négociations passées et sur les priorités des ressources et des personnes.

La couche stratégique permet de séparer les mécanismes de raisonnement sur le comportement à utiliser lors d'une négociation du processus de négociation à proprement parler. Afin de pouvoir utiliser différentes stratégies de négociation, ces stratégies sont modélisées par des interfaces, l'une pour le rôle d'initiateur, l'autre pour le rôle de participant, qu'il suffit d'implémenter. Cela nous permet s'abstraire le raisonnement des agents ce qui renforce la généricité de notre modèle. Nous avons implémenté les deux stratégies de négociation décrites dans le chapitre précédent afin de pouvoir utiliser immédiatement notre système.

L'objectif de notre implémentation est non seulement de pouvoir être utilisée de façon automatique, c'est-à-dire avec des agents répondant aux différentes propositions

qui leur sont faites, mais également de façon manuelle, c'est-à-dire que c'est l'utilisateur humain qui prend les décisions. Pour ce faire, nous avons réalisé une interface utilisateur générique qui est adaptée aux différents types de négociation. Elle permet de configurer certaines propriétés de négociation comme la façon de gérer les négociations ou les priorités données aux ressources et aux personnes. L'utilisateur peut ainsi créer des contrats, répondre aux propositions qui lui sont faites, visualiser les messages envoyés et reçus par l'agent et surveiller les négociations en cours.

Notre paquetage est directement utilisable en écrivant le fichier de configuration du système incluant les ressources qui seront négociées et la spécialisation du protocole. Nous fournissons quatre modes de communication ainsi que les agents correspondant de façon à pouvoir utiliser le système de suite. Les stratégies que nous fournissons permettent de réaliser des négociations immédiatement pour peu qu'elles n'impliquent pas de contrats nécessitant l'ajout de paramètres. Nous avons également montré qu'il était plus simple de réaliser une application avec *GeNCA* qu'avec *Zeus*.

Plusieurs applications ont été réalisées avec *GeNCA*, elles font l'objet du prochain chapitre.

Chapitre 5

Applications

Dans ce chapitre, nous présentons quelques applications parmi celles que nous avons réalisées avec *GeNCA*. Afin d'illustrer la polyvalence de *GeNCA*, nous avons choisi trois applications qui utilisent des modes de communication totalement différents et qui concernent des domaines d'applications eux aussi très différents. La première utilise un mode de communication par envoi d'e-mails et des ressources communes à tous les participants, il s'agit d'un système de prise de rendez-vous. La seconde utilise la communication au sein d'un SMA et des ressources individuelles aux participants, il s'agit d'une application de ventes aux enchères. La troisième utilise un mode de communication synchrone où chaque agent parle à tour de rôle et des ressources communes à tous les participants, il s'agit d'un jeu de négociation appelé JNego. Ces applications peuvent être téléchargées à l'adresse suivante : `http://www.lifl.fr/SMAC/projects/genca`.

L'objectif de ce chapitre est de montrer, à travers ces applications, que notre modèle général de négociation est valide et l'apport de *GeNCA* pour l'implémentation de ces applications.

5.1 Un système de prise de rendez-vous

Pour commencer, nous décrivons ici une des applications nécessitant l'utilisation de la plupart de nos paramètres. Ce problème est une application de négociation pour la prise de rendez-vous. Chaque agent doit être capable de négocier des rendez-vous pour le compte de l'utilisateur. Chaque utilisateur possède un agenda avec des plages horaires libres ou non. Il possède de plus des préférences sur les heures et les personnes avec qui il peut prendre rendez-vous. Chaque utilisateur peut initier une demande de

rendez-vous avec 1 ou plusieurs participants sur 1 ou plusieurs plages horaires. On s'interdit bien sûr de voir les agendas de chaque participant en transparence.

Ce problème est complexe car il a besoin des préférences sur les personnes, par exemple mon chef d'équipe a une plus grande priorité que mon collègue, mais aussi des priorités sur les ressources (ici les plages horaires), par exemple, si je ne veux pas prendre de rendez-vous à l'heure du déjeuner ou avant 8 heures, je donnerai une priorité inférieure à ces plages horaires. De plus, la prise de rendez-vous est une application où il y a typiquement beaucoup de renégociations et de rétractations, parce qu'il est difficile de trouver des plages horaires convenant à tout le monde.

5.1.1 Spécification de l'application

Bien évidemment, la communication entre agents pourrait prendre différentes formes, mais la plus évidente pour cette application est la communication par e-mail. Comme de tels agents sont fournis dans le paquetage (voir sous-section 4.1.3), nous utilisons le *MailCommunicator* du paquetage et il nous faut simplement créer les fichiers XML de configuration des e-mails pour chaque agent et pour le serveur de noms. Le fichier de configuration de l'application que nous avons présenté en section 4.5 contiendra donc le paramètre :

```
<communicator>
fr.lifl.genca.mail.MailCommunicator
</communicator>
```

Il s'agit ici de négocier des rendez-vous, c'est-à-dire de fixer des tranches horaires qui seront partagées entre les participants au même rendez-vous. Les ressources qui seront négociées sont donc tout naturellement les tranches horaires.

Le contrat qui sera envoyé par l'initiateur de la négociation contiendra les plages horaires qu'il propose pour le rendez-vous. Comme aucun autre paramètre n'est nécessaire pour ce contrat, il n'y a pas à modifier la couche de négociation.

Reste donc à examiner la couche stratégique. Les stratégies proposées par défaut conviennent à l'application, excepté pour le cas où plusieurs ressources sont nécessaires pour un rendez-vous. En effet, les stratégies par défaut ne tiennent pas compte de la contiguïté des ressources et pour la prise d'un rendez-vous nécessitant deux tranches horaires, on pourrait se retrouver avec un rendez-vous avec la première tranche horaire le matin et la seconde l'après-midi. Il y a donc une légère modification à apporter à ces stratégies par défaut pour qu'elles prennent en compte la spécificité de l'application pour la prise de rendez-vous.

Nous décrivons cependant de façon détaillée le déroulement d'une négociation au sein de *GeNCA* du côté de l'initiateur, puis du côté du participant.

5.1.2 Comportement de l'initiateur

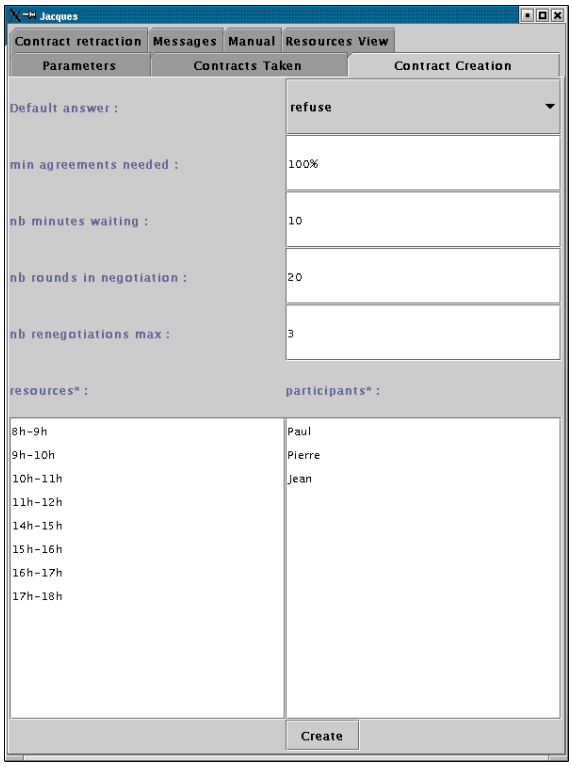


FIG. 5.1 – *Interface de création d'un contrat pour la prise de rendez-vous avec GeNCA*

L'agent humain, en tant qu'initiateur, crée son contrat via l'interface graphique où il indique les tranches horaires souhaitées, les participants conviés au rendez-vous, le nombre minimal d'accords nécessaires, le délai d'attente des réponses des participants, la réponse qu'il considère par défaut, le nombre de tours dans la négociation et le nombre de renégociations autorisées (Figure 5.1). On remarque que comme les personnes ne connaissent que leurs pseudonymes, la liste des participants ne contient donc pas les adresses e-mail des participants potentiels mais bien les pseudonymes fournis par le serveur.

Une fois cette intervention humaine réalisée, tout se fait automatiquement au sein de *GeNCA*. Il faut en effet qu'un utilisateur lance une première proposition pour initier la négociation.

L'interface graphique est nécessaire pour la création de contrat qui est toujours réalisée par un agent humain. Si l'utilisateur souhaite seulement participer aux négociations et ne jamais en initier, il peut alors se passer de l'interface et régler le paramètre `window` de son fichier de configuration personnel à `false`. Il devra alors regarder dans

le fichier de log de l'agent si il veut savoir l'état des négociations auxquelles il participe et s'il a pris des contrats.

GeNCA crée les objets *contract* et *contractProperties* ainsi qu'un micro-agent *but* auquel il donne en argument ces deux objets. La première chose que le but réalise est d'envoyer la proposition de contrat à l'ensemble des participants. Simultanément, il lance un timer (associé à une réponse par défaut, ici un refus) dans le but d'éviter d'attendre indéfiniment la réponse des participants.

Lorsqu'un accord est reçu, le nombre d'accords reçus est incrémenté. Le but l'indique à la stratégie via la méthode *receiveAccept(from,params)* qui permet à la stratégie de savoir qui a répondu favorablement et les paramètres qu'il a fournis. Dans le cas de la prise de rendez-vous, aucun paramètre n'est nécessaire et la stratégie ne fait rien dans ce cas. Une fois toutes les réponses reçues, le but consulte la stratégie (via la méthode *decide*) qui décide de *confirmer* le rendez-vous si le nombre minimal d'accords nécessaires est atteint, de *demander une modification* si ce nombre n'est pas atteint et si le nombre maximum de tours ne l'est pas non plus, ou de l'*annuler* dans les autres cas.

Ici, une modification est la proposition de nouvelles plages horaires qui conviennent mieux au participant.

Lorsqu'une *modification de contrat* est reçue, le but l'indique à la stratégie via la méthode *judge(from,params)*. La stratégie mémorise alors la proposition faite par ce participant et calcule une note (voir page 98) pour les plages horaires proposées selon la priorité du participant et selon la priorité de ces ressources pour le participant.

Lorsque toutes les modifications sont arrivées (ou lorsque le délai de réception a expiré), le but l'indique à la stratégie via la méthode *decideModification()*. Celle-ci choisit les nouvelles tranches horaires qui conviennent le mieux et *propose* le nouveau contrat si une possibilité est trouvée. Dans le cas contraire, quand aucune autre tranche n'est possible, elle peut soit demander une nouvelle modification (si le nombre de tours maximum n'est pas atteint), soit annuler le contrat.

La stratégie mémorise les propositions que l'initiateur a faites afin de ne pas proposer plusieurs fois le même contrat.

Lorsqu'un initiateur reçoit une *rétractation*, il regarde si il reste assez de participants pour le rendez-vous et tente de le déplacer si tel n'est pas le cas.

5.1.3 Comportement du participant

Lorsqu'un participant reçoit une *proposition* de rendez-vous, il crée un micro-agent engagement auquel il passe en paramètre le contrat. L'engagement consulte alors

la stratégie (via la méthode *whatDoIanswer(contrat)*) qui commence par vérifier si la tranche horaire est encore libre. Si tel est le cas, elle *accepte* le rendez-vous. Sinon, elle regarde si l'initiateur du rendez-vous est plus prioritaire que celui du rendez-vous déjà pris. Si oui, elle *accepte* le rendez-vous, sinon elle le *refuse*.

Lors de la réception d'une *confirmation* de rendez-vous, le participant l'ajoute à son agenda et envoie, si nécessaire, un message de *rétractation* à l'initiateur du(des) rendez-vous moins prioritaire(s) nécessitant ces tranches horaires.

Lors de la réception d'une *demande de modification*, le but la demande à la stratégie, via la méthode *proposeModification()*. Celle-ci consulte l'agenda du participant et envoie un nombre prédéfini de tranches horaires libres et par ordre de priorité à l'initiateur du rendez-vous. Elle mémorise sa proposition afin de ne pas refaire la même postérieurement.

5.1.4 Fichier de configuration de l'application

La figure 5.2 représente le contenu du fichier de configuration de l'application. On y retrouve la liste des ressources communes qui seront négociées, les stratégies à utiliser par défaut, les paramètres du protocole, la priorité par défaut définie pour l'agent, le mode de gestion des négociations conflictuelles et la nécessité de l'interface graphique.

Comme l'application consiste à négocier des rendez-vous et que logiquement, une personne ne peut pas être à deux rendez-vous différents en même temps, il nous paraît logique de négocier séquentiellement les rendez-vous qui entrent en conflit pour une (ou plusieurs) tranche horaire. D'où la valeur `sequential` du paramètre `management`.

Pour cette application, il est également possible de décider qu'un participant ne répondant pas à la proposition de rendez-vous l'accepte (`<default-answer value="accept"/>`). Ainsi, tous les participants qui auraient accepté la proposition ne sont pas obligés de répondre, puisque s'ils ne répondent pas, l'initiateur choisit de considérer qu'ils acceptent. Ceci permet de faire chuter le nombre de messages échangés dans une application où l'on peut espérer avoir de nombreux accords pour une proposition.

La Figure 5.3 montre le fichier de configuration personnel de l'agent *Agent1*. Cet agent souhaite utiliser ses propres stratégies de négociation : *rdv.MyRdvInitiatorStrategy* et *rdv.MyRdvParticipantStrategy*. Il se donne la priorité maximale (10) et ne souhaite pas avoir d'interface graphique (*false*).

```
<?xml version="1.0"?>
<!DOCTYPE genca SYSTEM "genca.dtd" >
<genca>
  <application-name>rdv</application-name>
  <resources-list>
    <resource>8h-9h</resource>
    <resource>9h-10h</resource>
    <resource>10h-11h</resource>
    <resource>11h-12h</resource>
    <resource>14h-15h</resource>
    <resource>15h-16h</resource>
    <resource>16h-17h</resource>
    <resource>17h-18h</resource>
  </resources-list>
  <communicator>
    fr.lifl.genca.mail.MailCommunicator
  </communicator>
  <default-initiator-strategy>
    rdv.RdvInitiatorStrategy
  </default-initiator-strategy>
  <default-participant-strategy>
    rdv.RdvParticipantStrategy
  </default-participant-strategy>
  <protocol>
    <answer-delay>10</answer-delay>
    <default-answer value="accept"/>
    <minAgreements>100%</minAgreements>
    <nbRounds>20</nbRounds>
    <nb-modifications-by-round>5</nb-modifications-by-round>
    <retraction-allowed value="true"/>
    <nbRenegotiations>3</nbRenegotiations>
  </protocol>
  <default-priority value="5"/>
  <management value="sequential"/>
  <window value="true"/>
</genca>
```

FIG. 5.2 – *Fichier XML de configuration de l'application pour la prise de rendez-vous*

```
<?xml version="1.0"?>
<!DOCTYPE agent SYSTEM "agent.dtd" >
<agent>
  <name>Agent1</name>
  <initiator-strategy>
    rdv.MyRdvInitiatorStrategy
  </initiator-strategy>
  <participant-strategy>
    rdv.MyRdvParticipantStrategy
  </participant-strategy>
  <default-priority value="10"/>
  <window value="false"/>
</agent>
```

FIG. 5.3 – *Exemple de fichier de configuration d'un agent souhaitant modifier les valeurs des paramètres stratégies à utiliser, priorité personnelle et interface graphique visible.*

5.1.5 Autres applications de prise de rendez-vous

D'autres chercheurs ont étudié la prise de rendez-vous et proposé un système pour la réaliser. Ce sont des applications dédiées à la prise de rendez-vous, tandis que nous utilisons un modèle général de négociation que nous appliquons à la prise de rendez-vous. A priori, traiter un problème de prise de rendez-vous avec un système de négociation n'est ni instinctif, ni commun, mais nous montrons ici que cela fonctionne parfaitement.

Sen et son équipe [Sen and Durfee, 1990, Sen and Durfee, 1994a, Sen and Durfee, 1994b, Sen et al., 1997, Sen, 1997] ont développé un planificateur de rendez-vous qui prend en compte les préférences de l'utilisateur. Différentes dimensions sont considérées, et pour chacune d'elles est définie la préférence de l'utilisateur. Parmi les différentes dimensions, on trouve le moment de la journée, le jour de la semaine, le statut des autres invités et le sujet de la réunion. Puis chaque dimension est pondérée de façon à respecter les choix que l'utilisateur ferait. Par exemple, si le patron de l'utilisateur propose une réunion à un moment de la journée qui ne satisfait pas l'utilisateur, il accepterait quand même la proposition. Un mécanisme de vote est ensuite utilisé pour arriver à un consensus. Le nombre de voix est réparti entre chaque dimension selon leur poids.

Le protocole de négociation utilisé permet aux participants de formuler des contre-propositions à l'initiateur de la négociation, sous la forme de nouveaux horaires leur convenant mieux. Différentes stratégies sont proposées, qui reposent entre autres sur le biais de recherche et les stratégies d'annonce, de réponses et de *commitement*.

Leur application permet, comme la nôtre, à un utilisateur d'être l'initiateur d'une proposition de rendez-vous et participant à d'autres simultanément. Elle prend également en compte plus de paramètres pour les préférences des utilisateurs que la nôtre,

qui prend surtout en compte les préférences sur les horaires et les personnes. Comme dans notre application, les agendas des utilisateurs sont privés.

Jennings et al. [Jennings and Jackson, 1995] ont proposé une conception et une implémentation pour la prise de rendez-vous prenant en compte les préférences de l'utilisateur sur les horaires et les personnes. Les agendas des utilisateurs sont privés. La proposition est constituée de l'ensemble des invités au rendez-vous, du sujet de la réunion, de la durée de la réunion puis de contraintes sur le temps (vendredi après-midi, par exemple) et sur les participants devant être présents pour confirmer la réunion (tous, un sous-ensemble des participants, etc.). Les participants proposent alors un nombre fixé de date et heure possibles pour eux, avec leur estimation. L'initiateur cherche alors une solution parmi les réponses envoyées. S'il en trouve une, il propose alors cette solution aux participants qui doivent encore la confirmer, sinon, le processus de propositions de date/heure est itéré. Si aucune solution n'est trouvée, le rendez-vous est renégocié en changeant les contraintes et/ou les participants. De même, si un rendez-vous doit être déplacé car un autre rendez-vous plus important a été pris à la même heure, celui-ci est renégocié automatiquement.

Cette application a donc de nombreux points communs avec la nôtre, comme la prise en compte des préférences de l'utilisateur, la renégociation automatique et la confidentialité des agendas des utilisateurs. Une différence est que nous ne dévoilons pas la liste des invités au rendez-vous.

Ito et Shintani [Ito and Shintani, 1997b, Ito and Shintani, 1997a] utilisent une négociation par persuasion et un processus analytique hiérarchique (AHP) afin d'organiser un ensemble de rendez-vous. Chaque agent donne un poids au rendez-vous et un coût aux intervalles de temps. Pendant la négociation du placement des rendez-vous, si l'intervalle de temps proposé pour un rendez-vous est rejeté, sa négociation est suspendue. Lorsque tous les rendez-vous ont été traités, il reste une liste de négociations suspendues. C'est là qu'intervient la persuasion. Si l'on prend l'exemple d'un agent a_1 qui avait fait une proposition p_1 à un agent a_2 et que l'agent a_2 l'avait refusée et vice-versa, l'agent a_1 peut indiquer à l'agent a_2 que s'il accepte sa proposition p_1, alors a_1 acceptera la proposition p_2 de l'agent a_2. Le conflit sera ainsi résolu.

Des logiciels pour prendre des rendez-vous se trouvent également dans le commerce, tels que Microsoft's Schedule+ 1.0, ON Technology's Meeting Maker 1.5, Now Software's Now Up-to-Date 2.0 et WordPerfect Office 3.0's Calendar module, qui sont basés sur le partage d'agendas. Ils ne réalisent donc pas du tout de la négociation mais plutôt résolvent un problème de satisfaction de contraintes.

Notre application permet aux agents de négocier des rendez-vous pour le compte de leur utilisateur. Contrairement à d'autres systèmes que l'on trouve dans le commerce, les agendas des utilisateurs sont privés et le problème ne consiste pas simplement à trouver un créneau horaire libre et commun à tous les participants, connaissant leurs

agendas, mais à négocier l'heure du rendez-vous en tenant compte des préférences des utilisateurs sur les horaires et les personnes. De plus, ce système renégocie automatiquement un rendez-vous qui doit être déplacé suite aux désistements des participants.

La convergence du système est liée aux paramètres de la négociation. En effet, notre stratégie permet de noter les ressources afin de proposer les plus appréciées des participants avant les autres en cas de non acceptation d'une proposition. Notre stratégie consiste à proposer toutes les ressources possibles, les unes après les autres, par ordre de préférence. Toutes les propositions possibles sont envisagées. Par conséquent, si une solution existe, la stratégie la trouve toujours. Le seul risque de ne pas trouver la solution est celui provenant du paramètre du nombre de tours dans la négociation. En effet, si l'on suppose que 100 propositions sont possibles, mais que le nombre de tours est limité à 50 et que la solution est la $60ème$ proposition, alors celle-ci ne sera pas trouvée.

De même, une fois toutes les propositions possibles effectuées, la négociation s'arrête. Une même proposition n'est jamais envoyée deux fois. Ainsi, on est sûr que le système s'arrête toujours.

5.2 Un système de ventes aux enchères

Les systèmes de ventes aux enchères connaissent une recrudescence depuis l'explosion de l'utilisation d'Internet. On peut citer des exemples de sites de ventes aux enchères tels que *eBay*, *Yahoo !Enchères*, qui connaissent une forte utilisation. Nous nous proposons ici de réaliser une application de vente aux enchères à offres scellées, dans laquelle chaque agent doit être capable de négocier pour l'utilisateur qu'il représente. Pour cela, chaque utilisateur définit une somme d'argent qui est à sa disposition pour faire ses achats (son crédit) et une stratégie d'enchère (linéaire, quadratique, etc.).

5.2.1 Spécification de l'application

L'enchère se déroule de la façon suivante : un vendeur propose un article pour lequel il décide d'obtenir un prix minimal qu'il garde secret (prix de réserve). Ensuite, les acheteurs lui font savoir s'ils sont intéressés ou non par cet article et, s'ils le sont, lui proposent un prix pour celui-ci. L'acheteur retient le prix le plus élevé proposé et si ce prix est supérieur au prix de réserve, l'acheteur ayant proposé ce prix gagne l'enchère. Si le meilleur prix proposé n'atteint pas le prix de réserve, le vendeur propose à nouveau son article aux acheteurs intéressés afin qu'ils enchérissent. Ceci est répété jusqu'à ce qu'un acheteur gagne l'enchère ou que personne n'enchérisse pour atteindre le prix de réserve.

Pour cette application, la rétractation n'est pas autorisée : une fois l'article vendu, il l'est définitivement. Le paramètre `retraction-allowed` du fichier de configuration de l'application sera donc à faux. Comme il s'agit ici d'une application de vente, la réponse par défaut qui sera considérée est un refus. On ne peut évidemment pas obliger un participant à acheter un article. Un seul accord est nécessaire à la confirmation du contrat car le vendeur ne peut vendre son article qu'à une seule personne.

D'après cette spécification, on remarque qu'un paramètre supplémentaire est nécessaire pour l'initiateur : le prix de réserve. Comme c'est un paramètre que seul l'initiateur connaît, il fait partie des propriétés du contrat. Il font donc étendre la classe *ContractProperties* pour qu'elle mémorise ce prix de réserve. Il faut également ajouter la saisie de ce paramètre dans l'onglet de création de contrat de l'interface graphique.

Cette application implique que l'agent possède une certaine somme d'argent et qu'il a envie d'acheter certains articles. Il nous faut donc mettre en place des méthodes permettant de consulter le crédit d'un agent et de savoir s'il désire acquérir l'objet mis en vente. Cela peut se faire en étendant la classe *Negotiator*. Le montant que l'agent possède au départ fera partie des paramètres spécifiques à l'application. Nous allons donc utiliser le paramètre optionnel du fichier de configuration de l'agent `application-parameters-list` qui contiendra un seul paramètre : la somme d'argent dont dispose l'agent. On trouvera donc dans le fichier de configuration d'un agent possédant 50 euros les lignes :

```
<application-parameters-list>
  <application-parameter>
    <name>credit</name>
    <parameter>
      <class>java.lang.Float</class>
      <value>50</value>
    </parameter>
  </application-parameter>
</application-parameters-list>
```

Il nous faut donc légèrement modifier (en fait, étendre) la couche de négociation de notre paquetage.

Nous allons utiliser une plateforme multi-agents pour implémenter cette application. Deux implémentations pour des SMA sont fournies avec le paquetage, l'une pour l'utilisation de Madkit, l'autre pour Magique. Nous allons ici utiliser la plateforme Magique. Le communicateur Magique est présent dans le paquetage (cf. sous-section 4.1.3), aucun travail de réalisation n'est donc nécessaire en ce qui concerne la couche de communication.

Les stratégies définies par défaut ne sont pas utilisables pour cette application car elle met en jeu des prix. Ils font donc écrire de nouvelles stratégies de négociation. Nous présentons ces stratégies dans les sous-sections suivantes.

5.2.2 Comportement de l'initiateur

L'initiateur commence par créer un contrat via l'interface graphique comme pour l'application de prise de rendez-vous. Dans le cas des enchères, un paramètre supplémentaire doit être indiqué : le prix de réserve. Le comportement du but étant le même quelle que soit l'application, nous décrivons ici uniquement la stratégie de l'initiateur.

Lorsqu'un accord est reçu, la stratégie met à jour le prix le plus élevé proposé jusqu'alors. Si le nouveau prix proposé le dépasse, ce prix devient le prix maximal proposé et l'acheteur l'ayant proposé le vainqueur courant de l'enchère. Une fois toutes les réponses reçues, l'initiateur décide de *confirmer* l'enchère auprès du participant ayant proposé le prix maximal si ce prix dépasse le prix de réserve du vendeur et donc d'*annuler* l'enchère auprès des autres participants. Si le prix de réserve n'est pas atteint ni le nombre de tours, l'initiateur *demande une modification* aux participants intéressés. Dans les autres cas, il *annule* l'enchère.

Lorsqu'une *modification* est reçue, l'initiateur procède exactement comme pour la réception d'un accord, car une modification consiste à proposer un nouveau prix pour l'article.

5.2.3 Comportement du participant

Nous décrivons ici uniquement la stratégie du participant car le comportement d'un engagement est le même quelle que soit l'application.

Lorsqu'un participant reçoit une *proposition* d'enchère, il regarde si l'article proposé l'intéresse ou non. Si c'est le cas, il *accepte* le contrat et propose un prix. Dans la stratégie que nous avons élaborée, le prix proposé est un pourcentage aléatoire du crédit de l'agent. La stratégie mémorise ce prix de façon à en proposer un qui soit supérieur par la suite si cela est nécessaire. Si l'article n'intéresse pas le participant, il *refuse*.

Lorsqu'une *confirmation* d'enchère est reçue, le participant ajoute l'article dans son sac et paie virtuellement le vendeur.

Lorsqu'une *demande de modification* est reçue, le participant vérifie la somme d'argent qu'il lui reste et propose un nouveau prix supérieur à celui qu'il avait proposé le tour précédent s'il a assez d'argent ou un prix égal à zéro s'il ne veut plus participer à l'enchère.

Pour calculer le prix à proposer dans le cas d'une demande de modification, la stratégie suit le comportement spécifié par l'utilisateur. La stratégie utilisée dans cet exemple augmente le prix précédemment proposé d'un pourcentage aléatoire de la somme restant à disposition de l'agent. D'autres méthodes d'enchérissement peuvent être utilisées, telles que les méthodes linéaire, quadratique ou exponentielle qui font varier le prix proposé au moyen d'une fonction linéaire (par exemple, $f(p) = 2 * p$), quadratique ($f(p) = p^2$) ou exponentielle ($f(p) = e^p$).

5.2.4 Fichiers de configuration de l'application

La Figure 5.4 représente le fichier de configuration de l'application de vente aux enchères. Aucune ressource n'est commune à tous les participants puisque ceux-ci vendent des articles qui leur sont propres. Comme nous l'avons indiqué précédemment, la rétractation n'est pas autorisée et seule une acceptation est nécessaire pour la vente de l'article. La réponse par défaut d'un participant est bien entendu un refus. Les agents utilisent le communicateur Magique et ne possèdent pas d'argent par défaut.

Un agent qui souhaite vendre des articles doit posséder son propre fichier de configuration où il indique ses articles. Tous les agents doivent également spécifier dans leur fichier de configuration la somme d'argent dont il dispose. La Figure 5.5 représente le fichier de configuration d'un agent nommé Paul qui désire vendre une table, un réfrigérateur et un livre de cuisine. Il possède 50 euros.

La Figure 5.6 montre les interfaces de 4 agents négociant des enchères avec notre API.

L'écran de l'agent en haut à gauche montre la fenêtre de visualisation des messages reçus et envoyés par l'agent. Elle permet de voir les différentes propositions reçues et l'avancement de la négociation (réponse envoyée, confirmation, annulation, demande de modification, etc.). L'écran de l'agent en haut à droite est l'interface de création d'un nouveau contrat. L'écran de l'agent en bas à gauche montre les contrats déjà pris par l'agent et l'écran de l'agent en bas à droite montre la proposition d'un contrat pour le mode manuel.

De nombreuses applications d'enchères existent, parmi lesquelles *Kasbah* [Chavez and Maes, 1996], *AuctionBot* [Wurman et al., 1998] et *Fishmarket* [Noriega, 1998], mais dans la plupart d'entre elles, une tierce personne regroupe des offres de vente et des offres d'achat et les apparie.

Les avantages de notre application sont nombreuses, nous allons donc citer ici les plus importantes. Premièrement, cette application aide l'utilisateur à enchérir et enchérit à sa place quand il n'est pas là, selon la stratégie qu'il a défini. Deuxièmement, cette

```
<?xml version="1.0"?>
<!DOCTYPE genca SYSTEM "genca.dtd" >
<genca>
  <application-name>auction</application-name>
  <resources-list>
  </resources-list>
  <communicator>
    fr.lifl.genca.magique.MagiqueCommunicator
  </communicator>
  <default-initiator-strategy>
    auction.AuctionInitiatorStrategy
  </default-initiator-strategy>
  <default-participant-strategy>
    auction.AuctionParticipantStrategy
  </default-participant-strategy>
  <protocol>
    <answer-delay>10</answer-delay>
    <default-answer value="refuse"/>
    <minAgreements>1</minAgreements>
    <nbRounds>20</nbRounds>
    <nb-modifications-by-round>1</nb-modifications-by-round>
    <retraction-allowed value="false"/>
    <nbRenegotiations>0</nbRenegotiations>
  </protocol>
  <default-priority value="5"/>
  <management value="sequential"/>
  <window value="true"/>
  <application-parameters-list>
    <application-parameter>
      <name>credit</name>
      <parameter>
        <class>java.lang.Float</class>
        <value>0</value>
      </parameter>
    </application-parameter>
  </application-parameters-list>
</genca>
```

FIG. 5.4 – *Fichier XML de configuration de l'application pour la vente aux enchères*

```
<?xml version="1.0"?>
<!DOCTYPE agent SYSTEM "agent.dtd" >
<agent>
  <name>Paul</name>
  <resources-list>
    <resource>table</resource>
    <resource>fridge</resource>
    <resource>cook book</resource>
  </resources-list>
  <application-parameters-list>
    <application-parameter>
      <name>credit</name>
      <parameter>
        <class>java.lang.Float</class>
        <value>50</value>
      </parameter>
    </application-parameter>
  </application-parameters-list>
</agent>
```

FIG. 5.5 – *Exemple de fichier de configuration d'un agent souhaitant vendre une table, un réfrigérateur et un livre de cuisine. Cet agent possède 50 euros.*

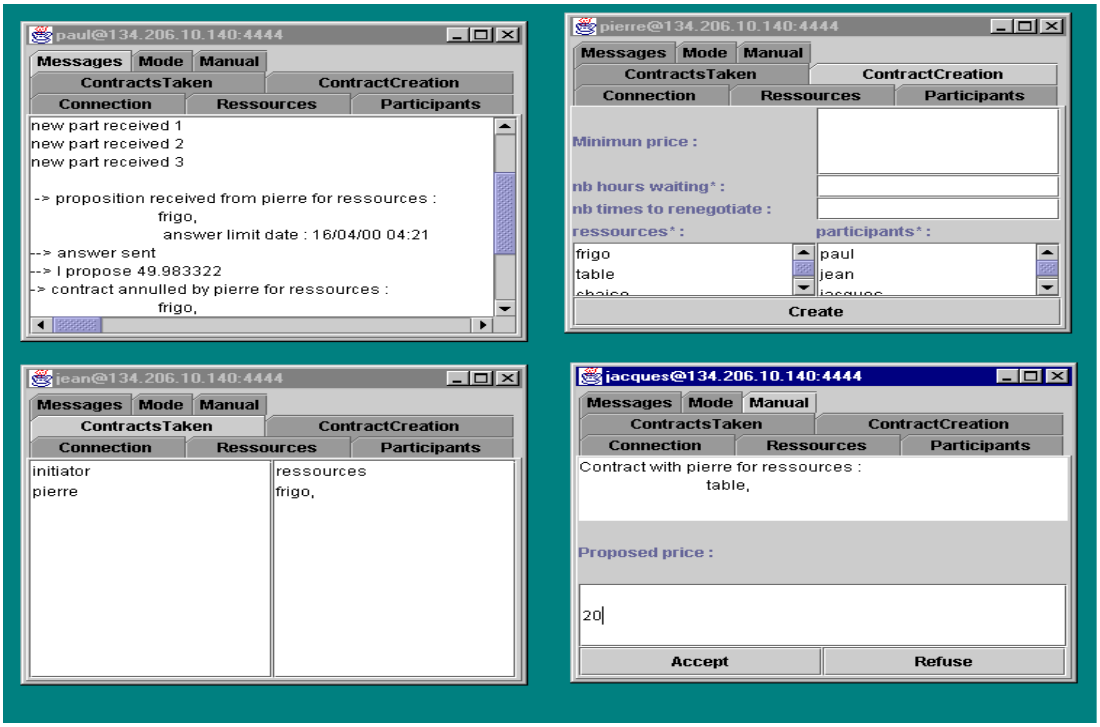


FIG. 5.6 – *Quatre agents participant à l'application de vente aux enchères*

application peut facilement être étendue à un autre type d'enchères, comme les enchères anglaises, hollandaises, Vickrey, etc. Et troisièmement, cette application est portable, en effet, les agents peuvent être placés sur un réseau hétérogène, sur des PDAs, etc.

5.3 JNego : un jeu de négociation de ressources

JNego est un jeu de négociation que nous avons inventé en nous inspirant du jeu Civilization™ (Avalon Hill Game Company). Ce jeu permet d'illustrer de nombreux concepts tant de négociation que de théorie des jeux, ce qui en fait sa richesse : ressources, intervenants, stratégies, bases de croyances, négociation, raisonnement sont au programme. Il a pour but de mettre en compétition des joueurs qui doivent maximiser la valeur des ressources qu'ils possèdent. Ce jeu fonctionne de façon synchrone, chaque joueur parle à tour de rôle. Nous présentons ce jeu afin de montrer que notre couche de négociation est adaptée à de nombreux domaines totalement différents. Nous avons vu qu'en adaptant uniquement la couche de communication, on peut utiliser *GeNCA* avec des agents communiquant par e-mail ou au sein d'un SMA. Cette application montre qu'on peut également l'utiliser dans un mode de négociation séquentiel, c'est-à-dire où les agents parlent à tour de rôle, pour effectuer de la négociation dans les systèmes synchrones comme des jeux de plateaux ou les jeux video.

5.3.1 Règles du jeu

Il y a 6 ressources différentes qui valent dans l'ordre et par convention de 1 à 6 points et qui sont présentes dans le jeu en quantité limitées :

1. le blé
2. le bois
3. la pierre
4. le bronze
5. l'argent
6. l'or

Chaque joueur possède N ressources. On note *valeur(r_i)* la valeur de la ième ressource et *nbCartes(r_i)* le nombre de cartes représentant la ième ressource que le joueur possède. La valeur de la main d'un joueur, notée *valeur(main)*, est calculée par la formule suivante :

$$\text{valeur(main)} = \sum_{j=1}^{6} \text{valeur}(r_i) * \text{nbCartes}(r_i)^2 \tag{5.1}$$

Cette formule [8] montre bien que plus un groupe de ressource est important, plus il rapporte. En effet, l'accroissement est quadratique par rapport au nombre de ressources identiques possédées. Prenons l'exemple d'une main composée de 3 bois et d'une main composée de 1 bois, 1 blé et 1 pierre. La première main vaut $2 * 3^2 = 18$ points alors que la seconde vaut $2 * 1^2 + 1 * 1^2 + 3 * 1^2 = 6$ points.

Pour parvenir à obtenir une main de valeur maximale, les joueurs s'échangent des cartes. La seule obligation est d'en échanger 2 contre 2. Cela oblige parfois les joueurs à rompre un groupe de ressources qu'il avait formé. Supposons qu'un joueur possède 5 pierres et 1 bois et qu'il veut échanger son bois, alors il devra aussi céder 1 pierre.

Les échanges peuvent être nominatifs ou faits par broadcast. Trois types d'échanges sont possibles :

1. l' échange de deux ressources nommées contre deux autres ressources nommées, par exemple 2 bois contre 2 pierre,
2. l'échange de deux ressources nommées contre deux autres ressources non nommées, par exemple 2 bois contre n'importe quoi,
3. l'échange de deux ressources non nommées contre deux autres ressources nommées, par exemple n'importe quoi contre 2 bois.

Exemple

Joueur	J_1	J_2	J_3
Ressources initiales	1 blé 1 bois 1 pierre 3 argent	1 bois 4 pierre 1 or	2 blé 3 bois 1 bronze
valeur	51 points	56 points	26 points

FIG. 5.7 – État initial du jeu.

Prenons l'exemple de 3 joueurs, J_1, J_2 et J_3. Leur jeu initial est donné en Figure 5.7. Supposons que le premier joueur décide de se débarrasser de sa pierre et de son blé. Le joueur J_2 lui propose alors 1 bois et 1 or, ce qui rapporterait 8 points supplémentaires à J_1. Le joueur J_3 propose 2 blé en échange, ce qui n'apporterait aucun point supplémentaire à J_1.

J_1 décide donc de réaliser l'échange avec J_2, la configuration du jeu étant alors celle de la Figure 5.8.

[8] Cette formule fait partie des règles du jeu, elle permet d'établir un ordre sur les joueurs en fonction des ressources qu'ils possèdent. Elle ne se rapporte pas à la formule que nous avons présentée dans notre stratégie de négociation en 3.4.3 qui elle sert à déterminer les ressources à proposer dans un contrat.

Joueur	J_1	J_2	J_3
Ressources initiales	2 bois 3 argent 1 or	1 blé 5 pierre	2 blé 3 bois 1 bronze
valeur	59 points	76 points	26 points

FIG. 5.8 – État du jeu après le premier échange.

Joueur	J_1	J_2	J_3
Ressources initiales	2 blé 3 argent 1 or	1 blé 5 pierre	5 bois 1 bronze
valeur	55 points	76 points	54 points

FIG. 5.9 – État du jeu après le second échange.

Maintenant, le joueur J_3 souhaite obtenir du bois. J_2 n'en possède pas donc refuse l'échange. Par contre, J_1 en possède 2 qu'il propose à J_3 et lui demande du blé en échange. J_3 accepte l'échange, ce qui lui rapporte 28 points supplémentaires (Figure 5.9).

5.3.2 Analyse du problème

Après une première lecture du problème, il apparaît clairement que le contrat devra être étendu pour permettre de spécifier la ressource voulue en échange de celle proposée (celle déjà présente dans le contrat de base). Précisons que la valeur d'une de ces 2 ressources peut être inconnue (cas 2 et 3 des types d'échange). Il faut donc écrire une nouvelle classe *JNegoContract* qui étend la classe *Contract*.

Il faudra également étendre la classe *Negotiator*. En effet, il faudra ajouter des méthodes d'évaluation et de mise à jour de la main du joueur lorsque celui-ci a effectué un échange. Cette classe devra également retenir les ressources que le joueur possède (sa main). De plus, il faudra lui ajouter les fonctions spécifiques pour le jeu, c'est-à-dire pouvoir informer ses buts ou engagements sur les cartes qu'il possède, les cartes qui l'intéresse et celles qu'il veut échanger. Les préférences sur les ressources peuvent être retenues par la liste de priorité sur les ressources déjà définie dans la classe de base.

Deux classes de la couche de négociation doivent donc être étendues pour cette application.

En ce qui concerne la communication, aucun travail ne sera nécessaire car notre paquetage fournit un *CentralisedCommunicator* qui permet de faire communiquer les agents de façon synchrone et de les faire parler à tour de rôle (cf. section 4.1.3).

Des stratégies doivent être mises en place pour ce jeu car elles doivent être en mesure de proposer un échange de ressource et d'évaluer la main du joueur, ce qui n'est pas prévu dans les stratégies par défaut. En effet, la stratégie d'initiateur que nous proposons (qui est décrite en section 3.4) confirme le contrat auprès de tous les participants ayant accepté la proposition de contrat. Or, dans ce jeu, le contrat consiste en un échange de ressources entre 2 joueurs (l'initiateur et un participant). Le contrat ne doit donc être confirmé qu'auprès d'un seul participant. De plus, plusieurs types de propositions existent (toutes les ressources sont nommées ou seulement deux d'entre elles) ce qui nécessite le traitement de chacun des cas. L'initiateur doit évaluer individuellement chacun des échanges proposés par les participants en cas de ressources non nommées. Suite à cette évaluation, il choisira celle qui lui convient le mieux. Si aucun échange ne lui convient, il peut alors demander une modification aux participants. Là encore, une évaluation individuelle des offres sera effectuée. Cette évaluation ne peut se faire avec la formule proposée dans la stratégie par défaut car premièrement, cette formule ne tient pas compte de l'évaluation de la main du joueur et deuxièmement, cette formule s'utilise afin de proposer un nouveau contrat pour une ressource préférée par tous et d'avoir de bonnes chances que ce contrat soit accepté par la majorité des participants, c'est une formule d'évaluation collective. Or ici, il s'agit pour l'initiateur de proposer une ressource qui ne l'intéresse pas mais qui soit la plus préférée possible par les participants pour le cas d'une ressource dont il se sépare, et le contraire s'il s'agit d'une ressource qu'il veut obtenir, c'est-à-dire une ressource qui maximise son gain mais aussi celui du participant avec qui l'échange sera fait. C'est pourquoi la stratégie par défaut ne peut être utilisée. Nous utiliserons donc une formule basée sur la valeur de la main du joueur afin d'évaluer les échanges proposés. Cette formule (5.2) sera présentée en section 5.3.3.

Nous présentons des stratégies simples pour ce jeu dans les sous-sections suivantes.

5.3.3 Le comportement de l'initiateur

La stratégie que nous présentons ici se décompose en trois cas correspondants aux trois types de contrats définis dans la présentation du problème.

Échange de 2 ressources nommées contre 2 autres ressources nommées C'est le cas le plus simple, si un participant a accepté le contrat, il le remporte ; si personne ne l'a accepté, il est annulé.

Si plusieurs participants l'ont accepté, l'un deux sera désigné au hasard (ou le premier ayant répondu) comme vainqueur du contrat.

Échange de 2 ressources proposées contre n'importe quoi Dans ce cas, il s'agit de céder les cartes proposées et d'en obtenir deux plus intéressantes en échange.

Si personne n'est intéressé par ces ressources, le contrat est annulé. Si plusieurs joueurs ont proposé deux ressources en échange, l'initiateur confirme le contrat auprès du participant ayant proposé l'échange e_i tel que :

$$\begin{cases} \text{gain}(e_i) > \text{gain}(e_j) \forall j \neq i \\ \text{gain}(e_i) > 0 \end{cases} \tag{5.2}$$

avec

$\text{gain}(e_k) = \text{valeur}(\text{main après l'échange } e_k) - \text{valeur}(\text{main avant l'échange } e_k)$

L'initiateur annule alors le contrat auprès des autres participant. Si aucune offre ne satisfait ces conditions, l'initiateur demande alors aux participants d'en proposer d'autres (demande de modification). La stratégie mémorise les propositions effectuées afin de pouvoir comparer les offres successives des participants.

Le principe est le même lorsque toutes les modifications ont été reçues.

Échange de 2 ressources désirées contre n'importe quoi Il s'agit ici d'obtenir les cartes désirées et d'en fournir deux autres en échange.

Si personne ne veut céder ces ressources, le contrat est annulé.

Si plusieurs joueurs veulent bien les échanger, l'initiateur utilise la formule 5.2 pour désigner l'offre qui sera retenue.

Si l'initiateur ne veut céder aucune des cartes demandées ou s'il n'a aucune de ces cartes, alors il demande aux participants de modifier leur demande. La stratégie mémorise les propositions qui ont été faites afin de pouvoir comparer les offres successives des participants.

Ce principe est conservé lors de la réception des modifications.

5.3.4 Le comportement du participant

Ici encore, nous allons décrire une stratégie simple qui se décompose elle aussi selon les trois types d'échanges pouvant être réalisés.

Échange de 2 ressources nommées contre 2 autres ressources nommées Si le participant ne possède pas les ressources désirées par l'initiateur ou si celles qu'il propose en échange ne l'intéressent pas (si elles n'augmentent pas la valeur de sa main), il refuse le contrat. Sinon, il l'accepte.

Échange de 2 ressources proposées contre n'importe quoi Si les ressources proposées intéressent le participant (il en a déjà), il accepte le contrat et propose en échange les ressources qu'il possède qui ne l'intéressent pas. Ce peut être par exemple les ressources qu'il possède en unique exemplaire et/ou celles qui valent le moins de points. Si les ressources proposées ne l'intéressent pas, il refuse le contrat.

En cas de demande de modification, le participant propose d'autres ressources qui ne l'intéressent pas ou rien s'il décide de ne se séparer d'aucune autre carte.

La stratégie mémorise les propositions afin de ne pas envoyer deux fois la même à l'initiateur.

Échange de 2 ressources désirées contre n'importe quoi Si le participant a les ressources désirées et qu'elles ne l'intéressent pas, alors il accepte le contrat et demande en échange deux cartes qui l'intéressent (celles qui augmenteraient la valeur de sa main). S'il ne possède pas les ressources désirées par l'initiateur, il refuse le contrat.

Si l'initiateur lui demande de modifier sa requête, alors il demande les ressources qui l'intéressent le plus à ce moment et qui sont différentes de celles qu'il avait proposées auparavant. Si aucune autre ressource ne l'intéresse, alors il ne propose aucune autre ressource en échange.

5.3.5 Fichier de configuration de l'application

La Figure 5.10 représente le fichier de configuration de l'application. On y retrouve le type de l'application : *jnego*, les ressources qui seront négociées, le communicateur utilisé par les agents ainsi que le nom des stratégies à utiliser par défaut. Vient ensuite le paramétrage du protocole. L'échange est effectué avec un autre joueur, un seul accord est par conséquent nécessaire à la confirmation du contrat. La réponse par défaut est un refus car on ne peut obliger un joueur à effectuer un échange, d'autant plus qu'il peut ne pas avoir les ressources à échanger. Les joueurs peuvent faire une seule proposition par tour lorsque l'échange concerne des ressources non nommées et l'initiateur peut leur demander une autre proposition jusqu'à 5 fois. La rétractation n'est pas autorisée, une fois un échange effectué, on ne peut plus changer d'avis.

Cet exemple montre bien que seules suffisent à être implémentées les actions spécifiques au problème posé, le protocole étant implémenté une fois pour toutes. Cette application n'a en effet nécessité que l'extension de deux classes du niveau de négociation et la création de stratégies de négociation adaptées au jeu.

5.4 Conclusion

Nous avons présenté dans ce chapitre trois applications utilisant des modes de communication totalement différents. La première utilise la communication par e-mails, la seconde au sein d'un SMA et la troisième une communication synchrone avec des agents parlant à tour de rôle. Deux de ces applications impliquent des ressources communes, la dernière des ressources individuelles.

La première application que nous avons présentée est un système de prise de rendez-vous. Cette application utilise la communication entre agents par envoi d'e-mails, implique des ressources communes à tous les agents (des tranches horaires) et

```
<?xml version="1.0"?>
<!DOCTYPE genca SYSTEM "genca.dtd" >
<genca>
  <application-name>jnego</application-name>
  <resources-list>
    <resource>ble</resource>
    <resource>bois</resource>
    <resource>pierre</resource>
    <resource>bronze</resource>
    <resource>argent</resource>
    <resource>or</resource>
  </resources-list>
  <communicator>
    fr.lifl.genca.centralised.CentralisedCommunicator
  </communicator>
  <default-initiator-strategy>
    jnego.JNegoInitiatorStrategy
  </default-initiator-strategy>
  <default-participant-strategy>
    jnego.JNegoParticipantStrategy
  </default-participant-strategy>
  <protocol>
    <answer-delay>10</answer-delay>
    <default-answer value="refuse"/>
    <minAgreements>1</minAgreements>
    <nbRounds>5</nbRounds>
    <nb-modifications-by-round>1</nb-modifications-by-round>
    <retraction-allowed value="false"/>
    <nbRenegotiations>0</nbRenegotiations>
  </protocol>
  <default-priority value="5"/>
  <management value="sequential"/>
  <window value="true"/>
</genca>
```

FIG. 5.10 – *Fichier XML de configuration de l'application pour le jeu JNego*

autorise les rétractations et la renégociation automatique. Pour cette application, les priorités sur les ressources et les personnes sont nécessaires pour gérer les conflits et ainsi choisir le contrat le plus prioritaire selon la personne qui le propose et les ressources qu'il implique. Plusieurs systèmes de prise de rendez-vous existent dans le commerce mais bien souvent, ils consistent à collecter les agendas des utilisateurs et à résoudre un problème de satisfaction de contraintes, à savoir trouver un créneau horaire libre pour tous les participants. *GeNCA* , quant à lui, permet de négocier des rendez-vous. Les utilisateurs gardent leur agenda privé et s'entendent sur un créneau horaire leur convenant tous.

La deuxième application que nous avons réalisée est un système de vente aux enchères. Cette application est réalisée au sein d'un SMA, implique des ressources individuelles et n'autorise pas la rétraction ni de ce fait la renégociation. Cette application intègre une nouvelle dimension dans les contrats : un prix. Il a donc fallu écrire des stratégies adaptées à cette application. Contrairement à de nombreux systèmes de vente qui collectent des offres d'achat et des offres de vente puis les apparient, *GeNCA* permet à un utilisateur de gérer ses enchères en tant que vendeur ou acheteur.

La troisième application est un jeu de négociation appelé JNego. Il utilise la communication synchrone où les agents parlent chacun leur tour et implique des ressources communes. Chaque agent a pour but de négocier l'échange de ressources afin de maximiser la valeur de sa main. Cette application fait intervenir des échanges de ressources. Il a donc fallu écrire de nouvelles stratégies prenant en compte la valeur de la main du joueur et le gain que l'échange proposé lui rapporterait.

Ces trois exemples montrent que notre protocole peut être appliqué à différents types d'applications de négociation comme les enchères ou la prise de rendez-vous. Cela permet de valider notre protocole et notre modèle général de négociation.

Chapitre 6

Conclusion et Perspectives

Notre recherche se place dans le domaine des protocoles de négociation. Notre objectif était de fournir un protocole général de négociation permettant de réaliser différentes applications de négociation, comme des enchères ou un système de prise de rendez-vous. Nous avons proposé un modèle général de négociation, et son implémentation, ce qui permet de l'utiliser plus facilement pour créer une application de négociation. Différents travaux ont été réalisés dans le thème de la négociation automatisée, et plusieurs systèmes de négociation ont vu le jour. Cependant, aucun ne permet de traiter l'ensemble des différentes formes de négociation les plus utilisées et la plupart sont dédiés à un type d'application, comme le commerce électronique, par exemple.

La négociation est de plus en plus utilisée dans les systèmes multi-agents pour résoudre les conflits. Elle peut prendre diverses formes, de la plus simple négociation *à prendre ou à laisser* à la plus complexe impliquant des contre-propositions. Nous avons montré dans cette thèse l'intérêt d'un modèle général de négociation.

De l'étude de ces différentes formes de négociation, nous avons dégagé plusieurs points communs qui existent entre elles, comme la présence des participants et de ressources (objets de la négociation), ainsi que différentes propriétés possédant des valeurs variables selon la négociation effectuée, comme le nombre de propositions qui sont faites et la possibilité de formuler des contre-propositions. Ces points communs et ces propriétés nous ont fourni une base pour l'élaboration de notre modèle général. En étudiant le déroulement de ces différentes négociations, nous avons également repéré un ensemble minimal d'actes de langages nécessaires pour réaliser l'ensemble de ces négociations. Cet ensemble minimal, contenant entre autres la proposition d'une offre, son acceptation ou son refus, nous a permis de concevoir notre protocole général de négociation, paramétrable afin d'obtenir un protocole spécifique pour une négociation donnée. Les paramètres correspondent aux différentes propriétés des négociations.

Notre proposition de modèle général de négociation a plusieurs objectifs, dont la généricité, la portabilité, l'uniformisation et l'automatisation des envois de messages. Nous fournissons en effet un modèle de négociation général, permettant de réaliser plusieurs formes de négociation différentes, sans demander de lourde charge de travail à un utilisateur.

Notre modèle, appelé *GeNCA*, sépare les niveaux de communication, de négociation et de stratégie qui sont indépendants les uns des autres au sein d'une application de négociation. Le niveau de communication sert à indiquer le moyen de communication utilisé par les agents. Ce peut être de la communication au sein d'un système multi-agents, aussi bien que par envoi d'e-mails ou encore par appel de procédure. Ce niveau est interchangeable pour permettre à une application spécifique de choisir son mode de communication.

Le niveau de négociation contient notre protocole général de négociation et l'ensemble des structures nécessaires afin de gérer les différentes négociations d'un agent. Ce niveau forme le cœur de notre modèle, qui a pour but de faciliter la modélisation et l'implémentation d'une application de négociation.

Le niveau stratégique permet de séparer le processus décisionnel de l'agent du protocole de négociation pour que la stratégie de négociation utilisée soit adaptée au problème traité. Il est en effet évident que la stratégie de négociation utilisée dépend de l'application, et que la négociation d'une tonne de pommes de terre ne se fait pas de la même façon que la négociation d'un rendez-vous chez le médecin.

Les principales qualités de notre modèle, outre les objectifs que nous avons cités précédemment, sont la possibilité de se rétracter et de renégocier un contrat qui avait été conclu, la possibilité de négocier simultanément tous les contrats ou seulement ceux qui concernent des ressources différentes, l'élimination des deadlocks pouvant survenir lors des négociations, notamment en utilisant un mécanisme de délai d'attente des réponse des participants.

Nous avons implémenté notre modèle sous la forme d'une API Java également appelée *GeNCA*, qui nous a permis de réaliser différentes applications de négociation, avec divers moyens de communication. Nous avons par exemple conçu une application permettant à des agents communicant par e-mail de négocier le choix d'un restaurant pour une sortie commune. Un autre exemple s'exécutant sur la plateforme Magique est celui de la vente aux enchères hollandaises d'articles possédés par les agents. Ces réalisations sont disponibles sur le site `http://www.lifl.fr/SMAC/projects/genca`.

Notre modèle s'adapte donc à différentes sortes de négociations, commerciales ou non, ce qui était l'un des objectifs que nous nous étions fixés. Il convient par exemple pour les familles des systèmes de vote et d'enchères et en partie pour la famille des négociations basées sur le *Contract Net Protocol*. En revanche, il ne convient pas pour l'instant pour la famille des négociations par argumentation. L'implémentation de ce

modèle est portable grâce à la technologie Java, et nous avons réalisé des applications s'exécutant sur différentes plates-formes multi-agents comme Magique et Madkit, et différents environnements tels que Windows™ et Linux.

Perspectives

La première perspective à moyen terme consiste à enrichir notre modèle afin de pouvoir représenter de nouvelles formes de négociations et en particulier, les négociations combinées, les négociations multi-niveaux et la négociation par argumentation.

En ce qui concerne les négociations combinées, nous allons reprendre l'exemple de la composition d'un voyage pour lequel il faut acquérir les nuits d'hôtel et les vols aller/retour pour la destination voulue. Nous allons donc permettre aux agents de participer dans différents types de négociation en même temps et de gérer les résultats de ces négociations afin d'obtenir tous les éléments composant le voyage, ou aucun.

La négociation multi-niveaux nécessitera de mettre en place un chaînage de plusieurs contrats qui devront être négociés les uns à la suite des autres, en permettant de revenir en arrière quand un contrat ne peut être négocié avec succès. Quant à la négociation par argumentation, elle nécessitera l'élaboration d'agents BDI et l'utilisation de formules logiques et d'un système expert pour vérifier les assertions des agents.

Nous avons présenté nos perspectives pour les mois à venir, mais nous envisageons d'autres perspectives à plus long terme, que nous présentons maintenant.

Représentation du protocole Nous avons également l'intention d'extraire le protocole de négociation de notre modèle afin de pouvoir en changer facilement, et de proposer une bibliothèque de protocoles utilisables avec *GeNCA*. Pour cela, il est nécessaire de définir les protocoles dans un formalisme qui soit clair pour l'utilisateur et facilement intégrable du point de vue génie logiciel. Une étude des différents formalismes de représentation des protocoles sera donc nécessaire. Parmi ces formalismes, les plus courants sont les automates, les réseaux de Pétri (colorés), et le formalisme AUML. Ceux-ci ont été étudiés par Huget dans [Huget, 2001], qui a proposé une ingénierie des protocoles d'interaction pour les systèmes multi-agents. D'autres formalismes existent, comme le langage Q [Ishida, 2002], Little-JIL [Wise et al., 2000], le langage COOL [Barbuceanu and Fox, 1995] et AgenTalk [Kuwabara et al., 1995]. Nous souhaitons utiliser un formalisme qui permette une utilisation simple et claire pour l'utilisateur et qui peut être échangé facilement entre les agents et qui permette également de générer automatiquement le code nécessaire à son utilisation. Nous allons donc étudier plus particulièrement les travaux de Secq [Secq, 2003] et de Huget [Huget, 2002b, Huget, 2002a] sur les protocoles d'interactions. Nous aimerions four-

nir à l'utilisateur un outil lui permettant de définir un protocole de négociation et de l'utiliser au sein de *GeNCA*.

Élaboration de stratégies Du point de vue du niveau stratégique, nous envisageons d'étudier de manière plus approfondie les stratégies de négociation. Le but de cette thèse étant de fournir un modèle général de négociation entre agents, nous n'avions pas approfondi le niveau stratégique de notre modèle car, comme nous l'avons indiqué dans cette thèse, la stratégie de négociation est totalement dépendante de l'application de négociation. En effet, lorsque l'on négocie des rendez-vous, la stratégie adoptée est totalement différente de celle utilisée au cours de ventes aux enchères. De façon plus générale, il est impossible d'utiliser la même stratégie de négociation pour des applications impliquant des contrats n'ayant pas les mêmes attributs si l'on veut une stratégie optimale pour l'application donnée. Une stratégie qui consiste à toujours accepter les propositions est utilisable quelle que soit l'application, mais elle ne donne pas des résultats optimaux pour toutes les applications.

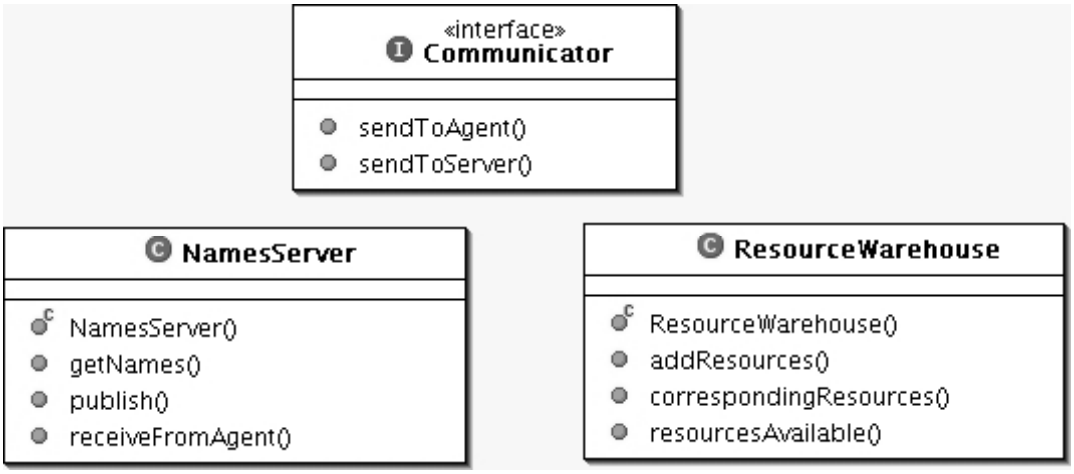
Notre but est de proposer différentes stratégies adaptées aux problèmes les plus courants, comme les ventes aux enchères. C'est un domaine de recherche important, qui nécessite une étude détaillée du contexte de la négociation, de ses règles et de son déroulement. Cette étude est à réaliser pour chaque application de négociation pour laquelle on veut fournir des stratégies de négociation.

Réalisation d'une application de grande envergure Pour valider ce type de travail, il est nécessaire de s'appuyer sur un contrat industriel permettant la mise en œuvre de notre AGL. C'est pourquoi nous aimerions réaliser une application de négociation de grande envergure, telle qu'une centrale d'achats, qui modélise les négociations entre les producteurs et la grande distribution, en passant par des grossistes ou des coopératives. Une telle application nous permettrait d'asseoir notre modèle et de montrer son utilité et sa facilité d'adaptation.

Annexe A

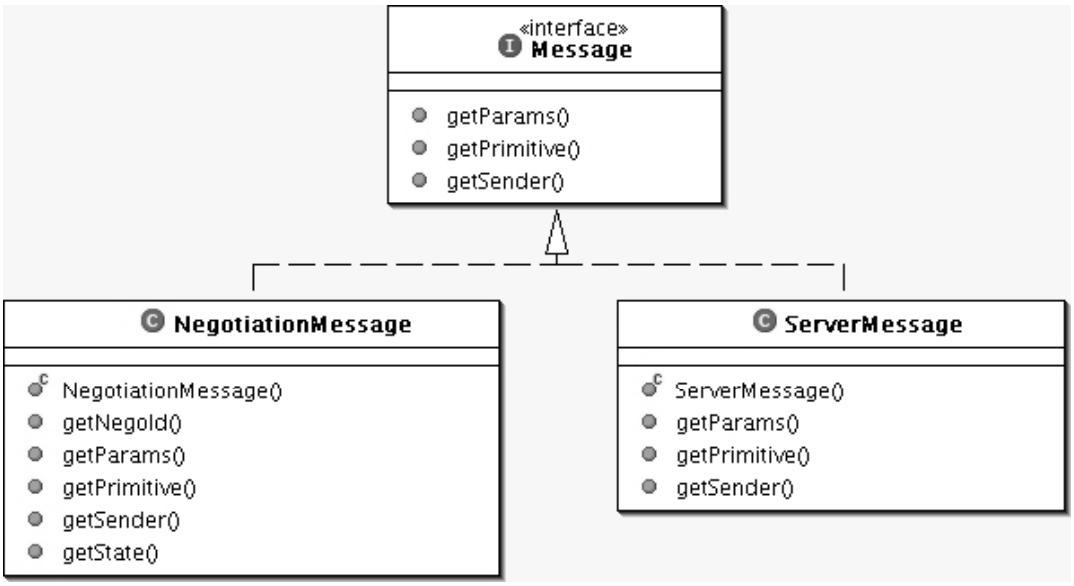
Diagrammes de classes

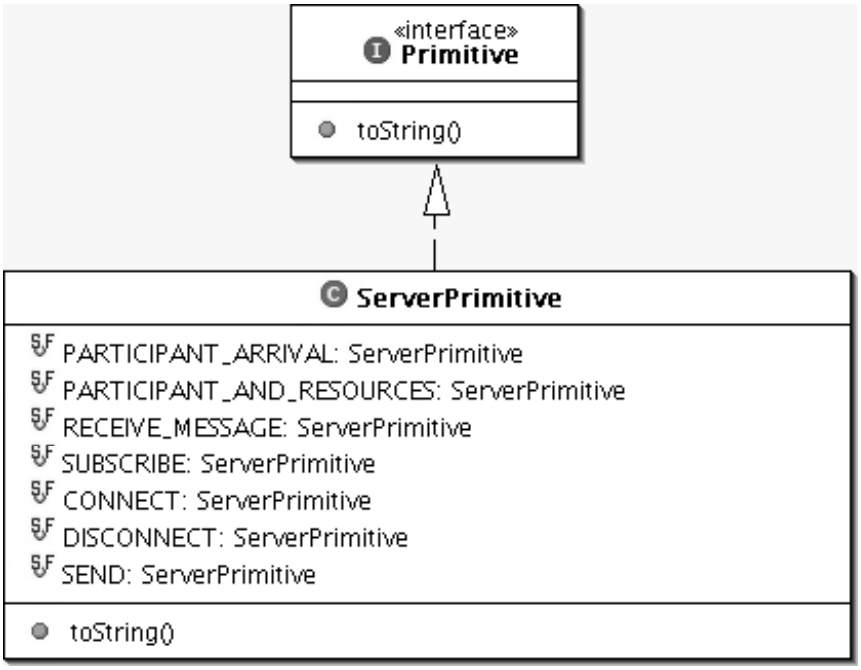
A.1 Paquetage `fr.lifl.genca.communication`



Ce paquetage correspond au niveau de communication de notre modèle. Afin de communiquer, les agents doivent posséder un *Communicator*, qui permet d'envoyer un message au serveur pour un agent négociant, ou à un agent pour le serveur. Ce paquetage comprend une description des messages qui seront utilisés (interface *Message*) et les deux types de messages qui seront échangés, à savoir des messages de liaison avec le serveur (*ServerMessage*) et des messages de négociation entre les agents négociants (*NegotiationMessage*). Chaque message contient l'expéditeur du message, une *primitive* qui indique le propos du message, ainsi que des paramètres. Un message de liaison avec le serveur possède sept primitives différentes (*ServerPrimitive*). Quatre d'entre elles sont

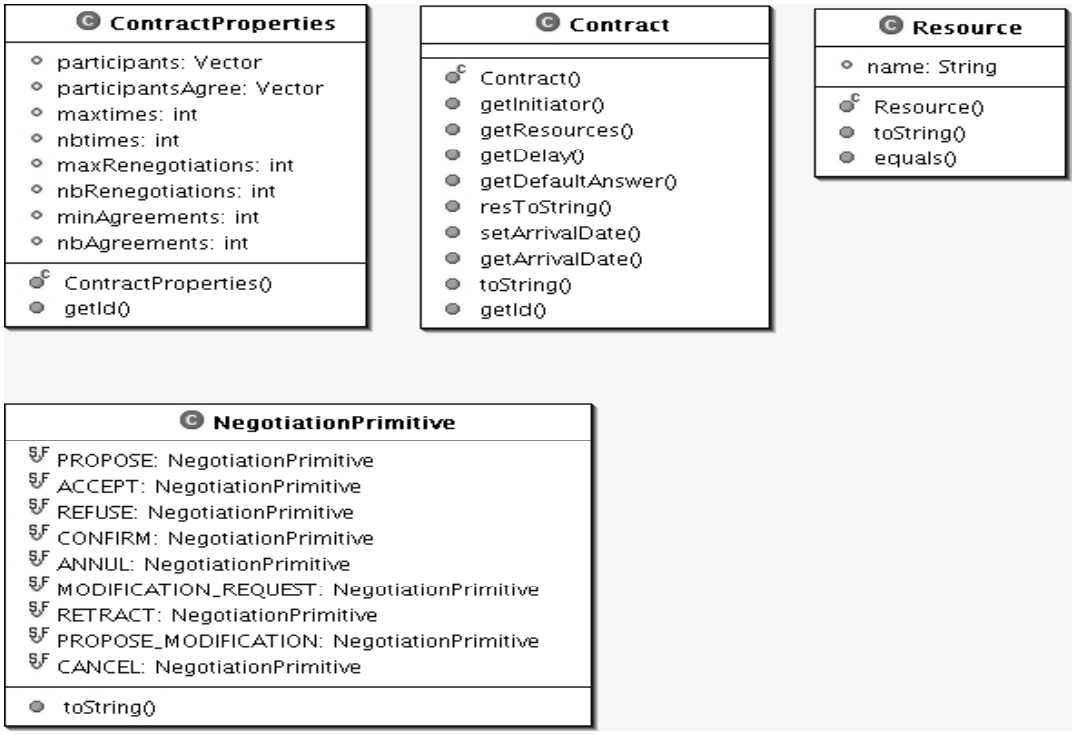
utilisées par les agents négociants pour joindre le serveur (SUBSCRIBE, CONNECT, DISCONNECT, SEND), les trois autres sont utilisées par le serveur pour joindre un agent négociant (PARTICIPANT_ARRIVAL, PARTICIPANT_AND_RESOURCES, RECEIVE_MESSAGE).





Le serveur possède un serveur de noms (*NamesServer*) qui lui permet de mémoriser les participants aux négociations selon le type d'application auquel ils s'abonnent, ainsi que les ressources mises en jeu lors des négociations. Ces ressources sont stockées dans un *ResourceWarehouse* qui permet de récupérer les ressources d'un type d'application donné, ou toutes les ressources.

A.2 Paquetage `fr.lifl.genca.negotiation`



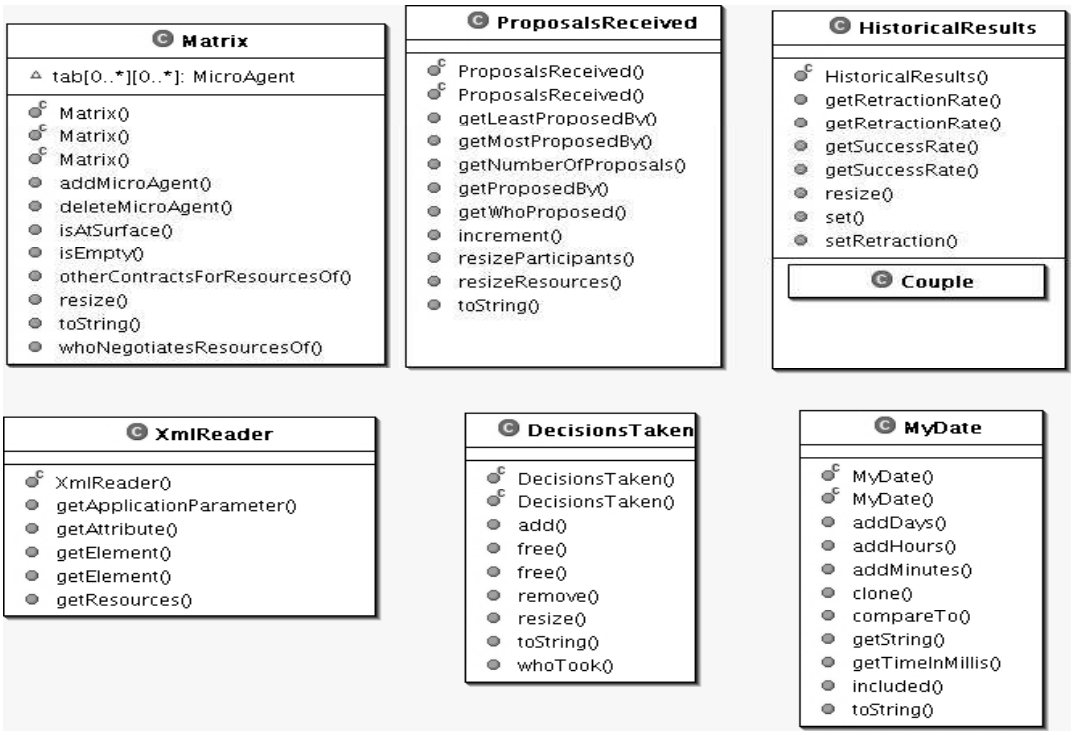
Ce paquetage correspond au niveau de négociation de notre modèle. Il contient les primitives de négociation que nous avons définies dans notre modèle et qui sont utilisées par notre protocole (*NegotiationPrimitive*). Ce paquetage comprend également les objets de description de la négociation, comme les ressources (*Resource*), les contrats (*Contract*) et les propriétés associées à un contrat (*ContractProperties*).

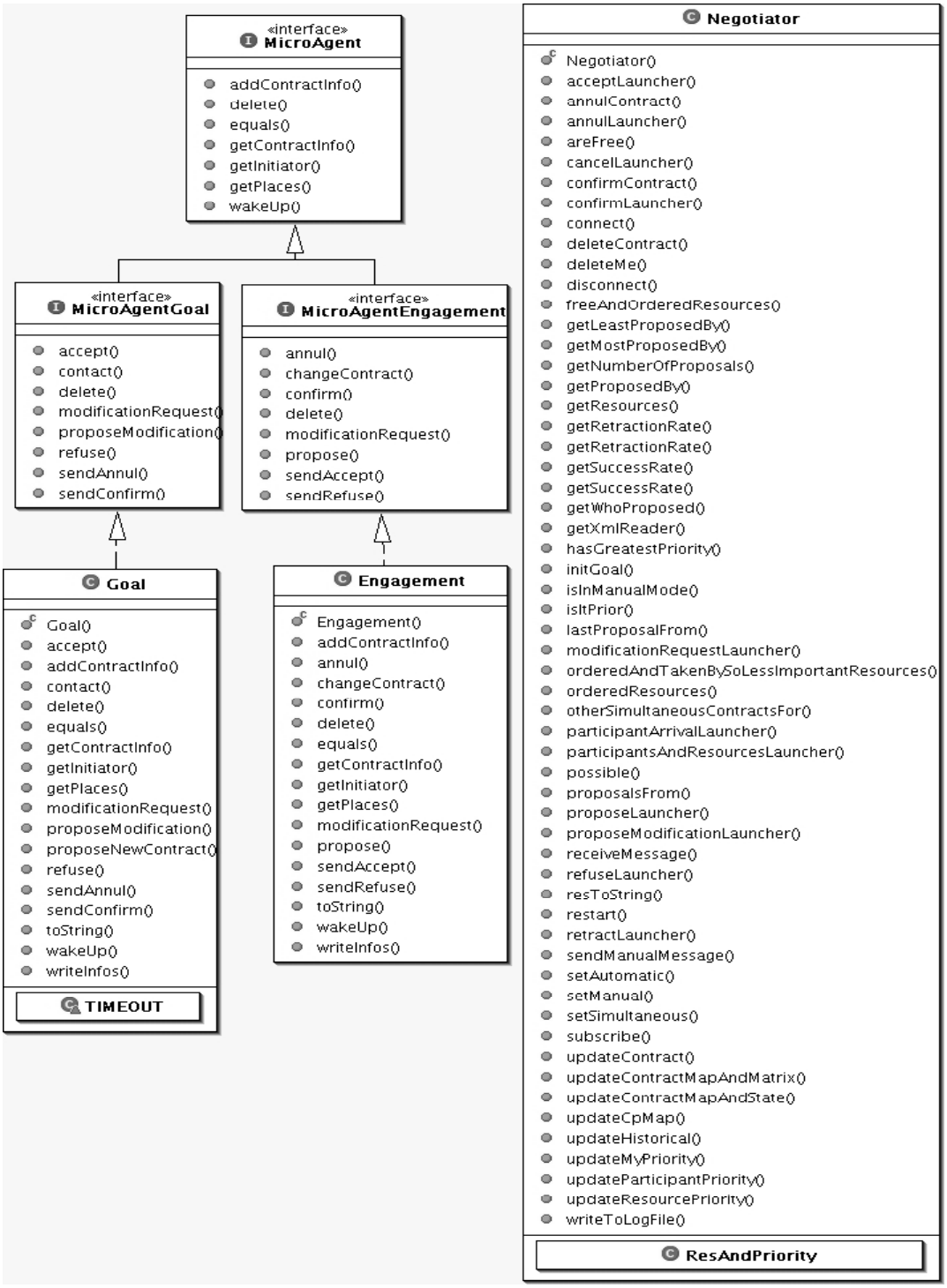
Les objets pour la dynamique de la négociation comprennent évidemment le négociateur (*Negotiator*) et les *micro-agents* que sont les buts (*Goal*) et les engagements (*Engagement*). Le négociateur est la classe principale de ce niveau, puisque c'est elle qui

gère les différentes négociations et reçoit les messages destinés à l'agent. Le négociateur garde aussi la trace des négociations passées, ce qui permet de fonder une stratégie de négociation sur les résultats des négociations passées.

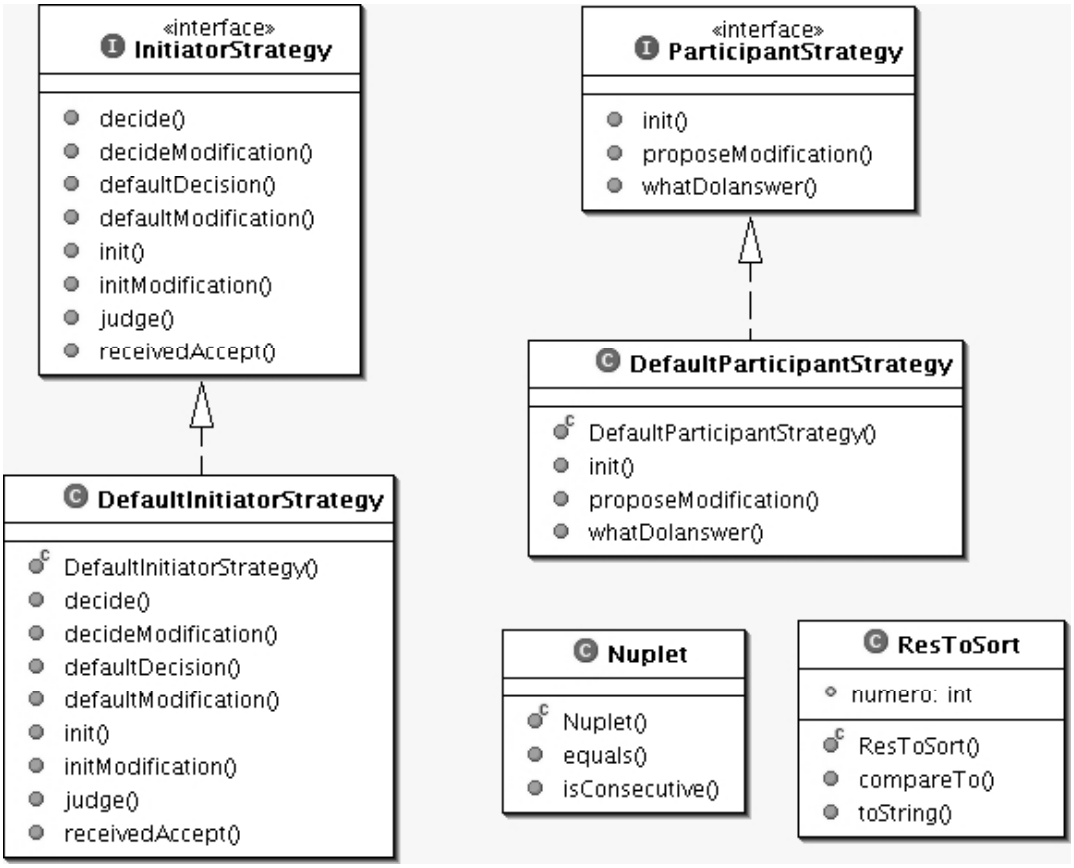
Enfin, différentes classes sont des outils pour la gestion des négociations, comme la matrice (Matrix) qui gère les négociations courantes et permet de savoir si les négociations peuvent se dérouler tout de suite ou doivent attendre la libération des ressources mises en jeu. La classe *DecisionsTaken* contient les contrats qui ont été conclus sur les ressources disponibles, les classes *ProposalsReceived* et *HistoricalResults* servent à mémoriser les négociations passées, la première concerne les négociations où l'agent tenait le rôle de participant, la seconde concerne les négociations dont l'agent était l'initiateur.

Les classes *MyDate* et *XMLReader* sont des utilitaires qui permettent de gérer le temps et de lire les fichiers de configuration au format XML.




«interface»
MicroAgent
addContractInfo()
delete()
equals()
getContractInfo()
getInitiator()
getPlaces()
wakeUp()
«interface»
MicroAgentGoal
accept()
contact()
delete()
modificationRequest()
proposeModification()
refuse()
sendAnnul()
sendConfirm()
«interface»
MicroAgentEngagement
annul()
changeContract()
confirm()
delete()
modificationRequest()
propose()
sendAccept()
sendRefuse()
Goal
Goal()
accept()
addContractInfo()
contact()
delete()
equals()
getContractInfo()
getInitiator()
getPlaces()
modificationRequest()
proposeModification()
proposeNewContract()
refuse()
sendAnnul()
sendConfirm()
toString()
wakeUp()
writeInfos()
TIMEOUT
Engagement
Engagement()
addContractInfo()
annul()
changeContract()
confirm()
delete()
equals()
getContractInfo()
getInitiator()
getPlaces()
modificationRequest()
propose()
sendAccept()
sendRefuse()
toString()
wakeUp()
writeInfos()
Negotiator
Negotiator()
acceptLauncher()
annulContract()
annulLauncher()
areFree()
cancelLauncher()
confirmContract()
confirmLauncher()
connect()
deleteContract()
deleteMe()
disconnect()
freeAndOrderedResources()
getLeastProposedBy()
getMostProposedBy()
getNumberOfProposals()
getProposedBy()
getResources()
getRetractionRate()
getRetractionRate()
getSuccessRate()
getSuccessRate()
getWhoProposed()
getXmlReader()
hasGreatestPriority()
initGoal()
isInManualMode()
isItPrior()
lastProposalFrom()
modificationRequestLauncher()
orderedAndTakenBySoLessImportantResources()
orderedResources()
otherSimultaneousContractsFor()
participantArrivalLauncher()
participantsAndResourcesLauncher()
possible()
proposalsFrom()
proposeLauncher()
proposeModificationLauncher()
receiveMessage()
refuseLauncher()
resToString()
restart()
retractLauncher()
sendManualMessage()
setAutomatic()
setManual()
setSimultaneous()
subscribe()
updateContract()
updateContractMapAndMatrix()
updateContractMapAndState()
updateCpMap()
updateHistorical()
updateMyPriority()
updateParticipantPriority()
updateResourcePriority()
writeToLogFile()
ResAndPriority

A.3 Paquetage `fr.lifl.genca.strategy`



Ce paquetage correspond au niveau de stratégie de notre modèle. On y trouve les squelettes de stratégies pour l'initiateur et le participant (*InitiatorStrategy* et *ParticipantStrategy*), ainsi qu'une implémentation par défaut de chacun d'eux. Deux classes utilitaires permettent de conserver l'ensemble des ressources formant une proposition (*Nuplet*) et de trier les ressources selon leur note (*ResToSort*).

Annexe B

Utilisation de *GeNCA* au sein de Madkit

Afin de montrer la facilité d'utilisation de *GeNCA*, nous présentons dans cette annexe les classes qu'il a fallu écrire pour l'utiliser au sein de la plateforme multi-agents Madkit. Il est bien évidemment tout aussi facile d'intégrer *GeNCA* au sein d'autres plateformes multi-agents comme Magique par exemple. Nous présentons tout d'abord le communicateur nécessaire pour la communication au sein de Madkit, puis la classe de l'agent serveur de noms et d'un agent négociateur. Ces trois classes sont les seules classes nécessaires pour utiliser *GeNCA* au sein de Madkit.

B.1 Le communicateur

Le communicateur (Figure B.1) utilise le moyen communication qui est employé au sein de Madkit. Le principe est de créer un groupe et des rôles au sein du groupe. Ainsi, il est possible d'envoyer des messages à un agent possédant un rôle particulier au sein d'un groupe. Nous avons donc créé un groupe nommé *genca* ainsi que les rôles *routeur* et *participant* au sein de ce groupe. Le rôle de *routeur* est tenu par l'agent serveur de noms. Ainsi, pour envoyer un message au serveur de noms, on utilise l'adresse de l'agent ayant le rôle *routeur* au sein du groupe *genca*. Les agents négociateurs tiennent tous le rôle de *participant* au sein du groupe *genca*. Pour envoyer un message à un agent négociateur, le serveur de noms utilise directement son adresse qu'il a obtenu lors de l'inscription de l'agent.

```
package fr.lifl.genca.madkit;

import fr.lifl.genca.communication.*;
import madkit.lib.messages.*;
import madkit.kernel.*;

/**
 * This class provides a communicator for the Madkit platform.
 *
 * @author Marie-Hélène Verrons
 */
public class MadkitCommunicator implements Communicator{

    private AgentAddress routeurAddress;
    private Agent myAgent;

    public MadkitCommunicator(Agent myAgent){
        this.myAgent=myAgent;
        routeurAddress=myAgent.getAgentWithRole("genca","routeur");
    }

    /** to send a message (synchronous)
        @param to the user/agent address
        @param msg the content of the message
    **/
    public void sendToAgent(Object to,ServerMessage msg){
        myAgent.sendMessage((AgentAddress)to, new ObjectMessage(msg));
    }

    /** to send a message to the names server
                @param message the message to send
    **/
    public void sendToServer(ServerMessage message) {
        myAgent.sendMessage(routeurAddress,new ObjectMessage(message));
    }
}
```

FIG. B.1 – *Le communicateur Madkit*

B.2 L'agent serveur de noms

```
package fr.lifl.genca.madkit;

import madkit.kernel.*;
import madkit.lib.messages.*;
import fr.lifl.genca.communication.*;
import java.util.Vector;

/**
 * This class provides an agent that plays the role of names server.
 *
 * @author Marie-Hélène Verrons
 */
public class MadkitRouteurAgent extends Agent {

    private NamesServer myServer;
    private Communicator myCommunicator;

    public void activate() {
        myCommunicator = new MadkitCommunicator(this);
        myServer = new NamesServer(myCommunicator);
        createGroup(true, "genca", null, null);
        requestRole("genca", "routeur", null);
    }

    public void live() {
        while (true) {
            madkit.kernel.Message e = waitNextMessage();
            if (e instanceof ObjectMessage) {
                myServer.receiveFromAgent((ServerMessage)
                                (((ObjectMessage) e).getContent()));
            }
        }
    }

    public void end() {
        println("I am dead, arghhh");
    }
}
```

FIG. B.2 – *L'agent serveur de noms dans Madkit*

Afin de définir un agent Madkit, il faut implémenter trois méthodes : `activate`, `live` et `end`. La méthode `activate` sert à initialiser l'agent, la méthode `live` décrit le comportement de l'agent durant sa *vie* et la méthode `end` sert à terminer l'agent lorsqu'il *meurt*.

L'agent serveur de noms (Figure B.2) possède un communicateur Madkit et utilise le serveur de noms que nous avons défini au sein de notre paquetage. Il crée le groupe *genca* et demande à tenir le rôle de *routeur* au sein de ce groupe.

L'agent attend les messages qui lui sont destinés et les traite au fur et à mesure qu'ils arrivent. Le traitement du message se fait par le serveur de noms de notre paquetage.

B.3 L'agent négociateur

```
package fr.lifl.genca.madkit;
import madkit.kernel.*;
import madkit.lib.messages.*;
import fr.lifl.genca.negotiation.*;
import fr.lifl.genca.communication.*;
/**
 * This class provides a negotiating agent for the Madkit platform.
 * @author Marie-Hélène Verrons
 */
public class MadkitNegotiatorAgent extends Agent{
    private Negotiator myNegotiator;

    public void activate(){
        AgentAddress myAddress=getAddress();
        String myName=getName();
        Communicator myCommunicator=new MadkitCommunicator(this);
        myNegotiator= new Negotiator(myName,myAddress,myCommunicator,
                                     myName+".xml","genca.xml");
        requestRole("genca","participant",null);
    }

    public void live(){
        while(true){
            madkit.kernel.Message e = waitNextMessage();
            if (e instanceof ObjectMessage)
                myNegotiator.receiveMessage((ServerMessage)
                                    ((ObjectMessage)e).getContent());
        }
    }

    public void end(){
        println("I am dead, arghhh");
    }
}
```

FIG. B.3 – *L'agent négociateur dans Madkit*

L'agent négociateur (Figure B.3) possède un communicateur Madkit et un négociateur que nous avons défini dans notre paquetage. Le négociateur prend en paramètre le nom et l'adresse de l'agent, le communicateur qu'il utilise ainsi que les noms des fichiers de configuration de l'application et de l'agent. Il demande à tenir le rôle *participant* au sein du groupe *genca*.

L'agent attend les messages qui lui proviennent du serveur de noms et les donne au négociateur afin qu'il les traite.

B.4 Lancement d'une application

Madkit permet de lancer une application en mode console grâce à la ligne de commande :

```
java madkit.platform.console.Booter -config config.cfg
```

où le fichier `config.cfg` contient la liste des agents présents dans l'application. La figure B.4 présente un fichier de configuration pour une application de négociation avec quatre agents négociateurs (MH, Philippe, JC et Yann) et l'agent serveur de noms (routeur). Il faut bien sûr fournir le fichier de configuration de l'application nommé `genca.xml` et optionnellement un fichier de configuration pour chaque agent négociateur.

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
<launch-agent name="routeur" type="Java"
code="fr.lifl.ants.madkit.MadkitRouteurAgent" gui="true" />
<launch-agent name="MH" type="Java"
code="fr.lifl.ants.madkit.MadkitNegotiatorAgent" gui="true" />
<launch-agent name="Philippe" type="Java"
code="fr.lifl.ants.madkit.MadkitNegotiatorAgent" gui="true" />
<launch-agent name="JC" type="Java"
code="fr.lifl.ants.madkit.MadkitNegotiatorAgent" gui="true" />
<launch-agent name="Yann" type="Java"
code="fr.lifl.ants.madkit.MadkitNegotiatorAgent" gui="true" />
</config>
```

FIG. B.4 – *Fichier de configuration Madkit*

Bibliographie

[fip,] http ://www.fipa.org.

[mag,] http ://www.lifl.fr/smac/projects/magique.

[mad,] http ://www.madkit.org.

[Aknine, 2002] Aknine, S. (2002). New Multi-Agent Protocols for M-N-P Negotiations in Electronic Commerce. In *National Conference on Artificial Intelligence, AAAI, Agent-Based Technologies for B2B Workshop*, Edmonton, Canada.

[Aknine, 2004] Aknine, S. (2004). Algorithmes de Recherche du Gagnant dans une Négociation Combinée. In *14ème édition du congrès francophone de Reconnaissance des Formes et d'Intelligence Artificielle (RFIA)*, Toulouse, France.

[Aknine et al., 2001] Aknine, S., Pinson, S., and Shakun, M. F. (2001). Négociation multi-agent : Analyse, modèle et expérimentations. *Revue d'Intelligence Artificielle, RIA*, 15(2) :173–217.

[Aknine et al., 2004] Aknine, S., Pinson, S., and Shakun, M. F. (2004). An extended multi-agent negotiation protocol. *International Journal on Autonomous Agents and Multi-agent Systems*, 8(1) :5–45.

[Alty et al., 1994] Alty, J. L., Griffiths, D. F., Jennings, N. R., Mamdani, E., Struthers, A., and Wiegand, M. E. (1994). ADEPT - Advanced Decision Environment for Process Tasks : Overview and Architecture. In *Proc BCS Expert Systems Conference (Applications Track, ISIP Theme)*, pages 359–371, Cambridge, UK.

[Amgoud and Prade, 2004a] Amgoud, L. and Prade, H. (2004a). Generation and evaluation of different types of arguments in negotiation. In *10th International Workshop on Non-Monotonic Reasoning, NMR'2004, Session Argument, Dialogue, Decision*, Whistler, Canada.

[Amgoud and Prade, 2004b] Amgoud, L. and Prade, H. (2004b). Un modèle de négociation basé sur la logique possibiliste. In *Congrés Francophone Reconnaissance des Formes et Intelligence Artificielle, RFIA'2004*, Toulouse, France.

[Andersson and Sandholm, 1998] Andersson, M. and Sandholm, T. (1998). Leveled commitment contracting among myopic individually rational agents. In *Proceedings of the Third International Conference on Multiagent Systems (ICMAS)*, pages 26–33, Paris, France.

[ASA, 2003] ASA (2003). Emploi du temps. Technical report, groupe ASA du GDR-I3.

[Barbuceanu and Fox, 1995] Barbuceanu, M. and Fox, M. S. (1995). COOL : A Language for Describing Coordination in Multi Agent Systems. In Lesser, V. and Gasser, L., editors, *Proceedings of the First International Conference oil Multi-Agent Systems (ICMAS-95)*, pages 17–24, San Francisco, CA, USA. AAAI Press.

[Bartolini and Preist, 2001] Bartolini, C. and Preist, C. (2001). A Framework for Automated Negotiation. Technical Report HPL-2001-90, HP Laboratories Bristol.

[Bartolini et al., 2002a] Bartolini, C., Preist, C., and Jennings, N. R. (2002a). Architecting for reuse : A software framework for automated negotiation. In *Proc. 3rd Int Workshop on Agent-Oriented Software Engineering*, pages 87–98, Bologna, Italy.

[Bartolini et al., 2002b] Bartolini, C., Preist, C., and Jennings, N. R. (2002b). A Generic software framework for automated negotiation. Technical Report HPL-2002-2, HP Laboratories Bristol.

[Bauer et al., 2001] Bauer, B., Müller, J. P., and Odell, J. (2001). Agent uml : A formalism for specifying multiagent interaction. In Ciancarini, P. and eds., M. W., editors, *Agent-Oriented Software Engineering*, pages 91–103, Berlin. Springer.

[Beer et al., 1999] Beer, M., d'Inverno, M., Luck, M., Jennings, N., Preist, C., and Schroeder, M. (1999). Negotiation in Multi-Agent Systems. *Knowledge Engineering Review*, 14(3) :285–289.

[Bensaid, 1999] Bensaid, N. (1999). *MAGIQUE, une architecture multi-agents hiérarchique*. Thèse de Doctorat, Université de Lille 1.

[Benyoucef, 2000] Benyoucef, M. (2000). Support for Combined Negotiations in E-Commerce : Concepts, Architecture, and Evaluation.

[Benyoucef et al., 2001] Benyoucef, M., Alj, H., Vézeau, M., and Keller, R. K. (2001). Combined Negotiations in E-Commerce : Concepts and Architecture. *Electronic Commerce Reseach Journal – Special Issue on Theory and Application of Electronic Market Design*, 1(3) :277–299.

[Benyoucef et al., 2000] Benyoucef, M., Keller, R. K., Lamouroux, S., Robert, J., and Trussart, V. (2000). Towards a Generic E-Negotiation Platform. In *Proceedings of the Sixth International Conference on Re-Technologies for Information Systems*, pages 95–109, Zurich, Switzerland.

[Bichler et al., 1999] Bichler, M., Kaukal, M., and Segev, A. (1999). Multi-attribute auctions for electronic procurement. In *Proc. 1st IBM IAC Workshop on Internet Based Negotiation Technologies*, Yorktown Heights.

[Bussmann and Muller, 1992] Bussmann, S. and Muller, J. (1992). A Negotiation Framework for Co-operating Agents. In M, D. S., editor, *Proc. CKBS-SIG*, pages 1–17, Dake Centre, University of Keele.

[Carbogim and Robertson, 1999] Carbogim, D. and Robertson, D. (1999). Contract-based Negotiation via Argumentation (preliminary report).

[Chavez and Maes, 1996] Chavez, A. and Maes, P. (1996). Kasbah : An Agent Marketplace for Buying and Selling Goods. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK.

[Collins et al., 1998] Collins, J., Youngdahl, B., Jamison, S., Mobasher, B., and Gini, M. (1998). A Market Architecture for Multi-Agent Contracting. In *2nd Int'l Conf on Autonomous Agents*, pages 285–292, Minneapolis.

[Collis and Ndumu, 1999a] Collis, J. C. and Ndumu, D. T. (1999a). The Role Modelling Guide. Technical report, British Telecommunications.

[Collis and Ndumu, 1999b] Collis, J. C. and Ndumu, D. T. (1999b). ZEUS Technical Manual. Technical report, British Telecommunications.

[Conry et al., 1992] Conry, S., Kuwabara, K., Lesser, V., and Meyer, R. (1992). Multistage Negotiation in Distributed Constraint Satisfaction. *IEEE Transactions on Systems, Man and Cybernetics*, 21(6) :1462–1477.

[Durfee and Lesser, 1989] Durfee, E. H. and Lesser, V. R. (1989). Negotiating task decomposition and allocation using partial global planning. In Gasser, L. and Huhns, M. N., editors, *Distributed Artificial Intelligence*, volume II, pages 229–243. Morgan Kaufmann, San Mateo, California.

[Faratin et al., 1999] Faratin, P., Sierra, C., Jennings, N. R., and Buckle, P. (1999). Designing Responsive and Deliberative Automated Negotiators. In *Proc AAAI workshop on negotiation settling conflicts and identifying opportunities*, pages 12–18, Orlando, FL.

[Gamble and Sen, 1994] Gamble, R. and Sen, S. (1994). Using Formal Specification to Resolve Conflicts between Contracting Agents. In *Proc. AAAI-94 Wokshop on Conflict Management in Cooperative Problem Solving*, pages 33–38, Seattle, Washington.

[Gerin-Lajoie, 2000] Gerin-Lajoie, R. (2000). Architecture informatique de gnp version 1.0. Technical report, CIRANO.

[Guttman and Maes, 1998] Guttman, R. and Maes, P. (1998). Cooperative vs. Competitive Multi-Agent Negotiations in Retail Electronic Commerce. In *Proceedings of the Second International Workshop on Cooperative Information Agents (CIA'98)*, Paris, France.

[Huget, 2001] Huget, M.-P. (2001). *Une ingénierie des protocoles d'interaction pour les systèmes multi-agents*. PhD thesis, Université Paris 9 Dauphine.

[Huget, 2002a] Huget, M.-P. (2002a). Generating Code for Agent UML Sequence Diagrams. Technical Report ULCS-02-020, Department of computer science, University of Liverpool.

[Huget, 2002b] Huget, M.-P. (2002b). A Language for Exchanging Agent UML Protocol Diagrams. Technical Report ULCS-02-009, Department of computer science, University of Liverpool.

[Huget et al., 2003] Huget, M.-P., Bauer, B., Odell, J., Levy, R., Turci, P., Cervenka, R., and Zhu, H. (2003). Fipa modeling : Interaction diagrams. Technical report, FIPA.

[Ishida, 2002] Ishida, T. (2002). Q : A Scenario Descripion Language for Interactive Agents. *IEEE Computer*, 35(11) :42–47.

[Ito and Shintani, 1997a] Ito, T. and Shintani, T. (1997a). An Agenda-scheduling System Based on Persuasion Among Agents. In *Information Systems and Technologies for Network Society*, Fukuoka, Japan.

[Ito and Shintani, 1997b] Ito, T. and Shintani, T. (1997b). Persuasion among Agents : An Approach to Implementing a Group Decision Support System Based on Multi-Agent Negotiation. In *Proceedings of the Fifteenth Int. Joint Conference on Artificial Intelligence (IJCAI97)*, pages 592–597.

[Jennings et al., 2000] Jennings, N. R., Faratin, P., Lomuscio, A. R., Parsons, S., Sierra, C., and Wooldridge, M. (2000). Automated Negotiation : Prospects, Methods and Challenges. *Int. Journal of Group Decision and Negotiation*, 10(2) :199–215.

[Jennings and Jackson, 1995] Jennings, N. R. and Jackson, A. (1995). Agent based meeting scheduling : A Design and Implementation. *Electronics Letters, The Institution of Electrical Engineering*, 31(5) :350–352.

[Jennings et al., 1998] Jennings, N. R., Norman, T. J., and Faratin, P. (1998). ADEPT : An Agent-based Approach to Business Process Management. *ACM SIGMOD Record*, 27(4) :32–39.

[Jonker and Treur, 2001] Jonker, C. M. and Treur, J. (2001). An Agent Architecture for Multi-Attribute Negotiation. In Nebel, B., editor, *Proceedings of the 17th International Joint Conference on AI, IJCAI'01*, pages 1195 – 1201.

[Kersten et al., 2001] Kersten, G. E., Noronha, S. J., and Teich, J. (2001). Are All E-Commerce Negotations Auctions ? In *Proceedings of the Fourth International Conference on the Design of Cooperative Systems (COOP'2000)*, Sophia-Antipolis, France.

[Klein, 1991] Klein, M. (1991). Supporting conflict resolution in cooperative design systems. *IEEE Transactions on System, Man, Cybernetics*, 21(5) :1379–1390.

[Korf, 1985] Korf, R. (1985). Depth-first iterative-deepening : An optimal admissible tree search. *Artificial Intelligence*, 27 :97–109.

[Kraus, 2001] Kraus, S. (2001). *Strategic Negotiation in Multiagent Environments*. MIT Press.

[Kraus et al., 1998] Kraus, S., Sycara, K., and Evenchik, A. (1998). Reaching agreements through argumentation : a logical model and implementation. *Artificial Intelligence*, 104 :1–69.

[Kuwabara et al., 1995] Kuwabara, K., Ishida, T., and Osato, N. (1995). AgenTalk : Describing Multiagent Coordination Protocols with Inheritance. In *Proceedings of the Seventh International Conference on Tools with Artificial Intelligence (TAI'95)*, pages 460–465. IEEE.

[Mathieu and Verrons, 2002] Mathieu, P. and Verrons, M.-H. (2002). A generic model for contract negotiation. In *Proceedings of the AISB'02 Convention*, pages 1–8, London, UK.

[Mathieu and Verrons, 2003a] Mathieu, P. and Verrons, M.-H. (2003a). ANTS : an API for creating negotiation applications. In *Proceedings of the 10th ISPE International Conference on Concurrent Engineering : Research and Applications (CE2003), track on Agents and Multi-agent systems*, pages 169–176, Madeira Island, Portugal.

[Mathieu and Verrons, 2003b] Mathieu, P. and Verrons, M.-H. (2003b). A Generic Negotiation Model for MAS using XML. In *Proceedings of the ABA workshop Agent-based Systems for Autonomous Processing, held by the IEEE International Conference on Systems, Man and Cybernetics.*, pages 4262–4267, Washington, USA. IEEE Press.

[Mathieu and Verrons, 2004] Mathieu, P. and Verrons, M.-H. (2004). 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.

[Mathieu and Verrons, 2005a] Mathieu, P. and Verrons, M.-H. (2005a). GeNCA : Un modèle général de négociation de contrats. *Revue d'Intelligence Artificielle*. à paraître.

[Mathieu and Verrons, 2005b] Mathieu, P. and Verrons, M.-H. (2005b). A General Negotiation Model using XML. *Artificial Intelligence and Simulation of Behaviour Journal (AISBJ)*. à paraître.

[Mathiot, 2004] Mathiot, C. (2004). Distribution : les fournisseurs asphyxiés sur le Net. *Libération*.

[Meyer et al., 1988] Meyer, R. A., Conry, S. E., and Lesser, V. R. (1988). Multistage negotiation in distributed planning. In Bond, A. and Gasser, L., editors, *Reading In Distributed Artificial Intelligence*, pages 367–386, Los Altos, CA. Morgan Kaufmann Publishers.

[Neumann et al., 2003] Neumann, D., Benyoucef, M., and Vachon, S. B. J. (2003). Applying the montreal taxonomy to state of the art e-negotiation systems. *Group Decision and Negotiation*, 12(4) :287–310.

[Noriega, 1998] Noriega, P. (1998). *Agent mediated auctions : The Fishmarket Metaphor*. PhD thesis, University of Barcelona.

[Nwana et al., 1999] Nwana, H. S., Ndumu, D. T., Lee, L. C., and Collis, J. C. (1999). ZEUS : A Toolkit for Building Distributed Multi-Agent Systems.

[Parsons and Jennings, 1996] Parsons, S. and Jennings, N. R. (1996). Negotiation through argumentation - a preliminary report. In *Proc. Second Int. Conf. on Multi-Agent Systems*, pages 267–274, Kyoto, Japan.

[Pruitt, 1981] Pruitt, D. (1981). *Negotiation Behavior*. Academic Press.

[Rodriguez-Aguilar et al., 1997] Rodriguez-Aguilar, J. A., Noriega, P., Sierra, C., and Padget, J. (1997). FM96.5 A Java-based Electronic Auction House. In *Proceedings of the Second International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'97)*, London, UK.

[Sandholm, 1993] Sandholm, T. (1993). An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations. In *Eleventh National Conference on Artificial Intelligence, AAAI-93*, pages 256–262, Washington DC. (Acceptance Rate 24%).

[Sandholm, 2002] Sandholm, T. (2002). Algorithm for Optimal Winner Determination in Combinatorial Auctions. *Artificial Intelligence*, 135 :1–54.

[Sandholm and Lesser, 1995] Sandholm, T. and Lesser, V. (1995). Issues in Automated Negotiation and Electronic Commerce : Extending the Contract Net Framework. In *First International Conference on Multiagent Systems (ICMAS-95)*, pages 328–335, San Francisco. (Acceptance rate 33%).

[Sandholm and Lesser, 1996] Sandholm, T. and Lesser, V. (1996). Advantages of a leveled commitment contracting protocol. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 126–133, Portland, OR. AAAI Press.

[Secq, 2003] Secq, Y. (2003). *RIO : Rôles, Interactions et Organisations, une méthodologie pour les systèmes multi-agents ouverts*. PhD thesis, Université des Sciences et Technologies de Lille.

[Sen, 1997] Sen, S. (1997). Developping an automated distributed meeting scheduler. *IEEE Expert*, 12(4) :41–45.

[Sen and Durfee, 1990] Sen, S. and Durfee, E. H. (1990). A Formal Study of Distributed Meeting Scheduling. *Group Decision and Negotiation*, 7 :265–289.

[Sen and Durfee, 1994a] Sen, S. and Durfee, E. H. (1994a). On the design of an adaptive meeting scheduler. In *Proceedings of the Tenth IEEE Conference on Artificial Intelligence for Applications*, pages 40–46, San Antonio, Texas.

[Sen and Durfee, 1994b] Sen, S. and Durfee, E. H. (1994b). The role of commitment in cooperative negotiation. *International Journal on Intelligent & Cooperative Information Systems*, 3(1) :67–81.

[Sen et al., 1997] Sen, S., Haynes, T., and Arora, N. (1997). Satisfying User Preferences While Negotiating Meetings. *Inernational Journal of Human-Computer Studies, special issue on Group Support Systems*, 47 :407–427.

[Sierra et al., 1997] Sierra, C., Faratin, P., and Jennings, N. R. (1997). A service oriented negotiation model between autonomous agents. In *Proc 8th European workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW-97*, pages 17–35, Ronneby, Sweden.

[Smith, 1980] Smith, R. G. (1980). The Contract Net Protocol : high-level communication and control in a distributed problem solver. *IEEE Transactions on computers*, C-29(12) :1104–1113.

[Ströbel, 2000] Ströbel, M. (2000). A Framework for Electronic Negotiations Based on Adjusted Winner Mediation. In *Proceedings of the DEXA 2000 Workshop on e-Negotiations*, pages 1020–1028, London, UK.

[Ströbel, 2001a] Ströbel, M. (2001a). Communication Design for Electronic Negotiations on the Basis of XML Schema. In *Proceedings of the 10th World Wide Web Conference*, pages 9–20, Hong Kong.

[Ströbel, 2001b] Ströbel, M. (2001b). Design of Roles and Protocols for Electronic Negotiations. *Electronic Commerce Research, Special Issue on Market Design*, 1(3) :335–353.

[Ströbel and Stolze, 2001] Ströbel, M. and Stolze, M. (2001). A Matchmaking Component for the Discovery of Agreement and Negotiation Spaces in Electronic Markets. In *Proceedings of Group Decision and Negotiation*, pages 61–75, La Rochelle, France.

[Ströbel and Weinhardt, 2003] Ströbel, M. and Weinhardt, C. (2003). The montreal taxonomy for electronic negotiations. *Group Decision and Negotiation*, 12(2) :143–164.

[Sycara, 1988] Sycara, K. (1988). Resolving goal conflicts via negotiation. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 245–250, St. Paul, Minnesota.

[Sycara, 1990] Sycara, K. (1990). Persuasive argumentation in negotiation. *Theory and Decision*, 28 :203–242.

[Taylor, 1995] Taylor, A. D. (1995). *Mathematics and Politics - Strategy, Voting, Power and Proof*. Springer-verlag.

[Tsvetovatyy et al., 1997] Tsvetovatyy, M., Gini, M., Mobasher, B., and Wieckowski, Z. (1997). MAGMA : An Agent-Based Virtual Market for Electronic Commerce. *Journal of Applied Artificial Intelligence*, 6.

[Verrons, 2001] Verrons, M.-H. (2001). Etude de la négociation entre agents autonomes. Université des sciences et technologies de Lille. Rapport de stage de DEA.

[Verrons, 2004] Verrons, M.-H. (2004). Une forme de négociation entre entités virtuelles. In *Conférence MajecSTIC'04*.

[Wise et al., 2000] Wise, A., Cass, A. G., Lerner, B. S., McCall, E. K., Osterweil, L. J., and Jr., S. M. S. (2000). Using Little-JIL to Coordinate Agents in Software Engineering. In *Proc. of the Automated Software Engineering Conference (ASE 2000)*, pages 155–163, Grenoble, France.

[Wurman et al., 1998] Wurman, P. R., Wellman, M. P., and Walsh, W. E. (1998). The Michigan Internet AuctionBot : A Configurable Auction Server for Human and Software Agents. In *Proceedings of the Second International Conference on Autonomous Agents (Agents-98)*, Minneapolis, MN, USA.

Résumé

Lorsque de nombreux agents interagissent, des conflits peuvent survenir. Pour les résoudre, différentes méthodes peuvent être utilisées, comme la coordination, les systèmes de vote et la négociation. Celle qui nous intéresse dans cette thèse est la négociation à base de contrats portant sur des ressources.

L'objectif de notre travail est de concevoir un modèle général de négociation (appelé *GeNCA* : *Generic Negotiation of Contracts API*) et une implémentation de ce modèle. Ceci permet à un utilisateur souhaitant développer une application de négociation de ne pas devoir tout réaliser mais de pouvoir utiliser un modèle qui lui facilitera le travail.

Afin de concevoir un tel modèle, nous commençons par étudier les formes de négociation les plus courantes. L'analyse de ces négociations nous permet de dégager leurs points communs et nous fournit une base pour concevoir notre modèle. Grâce à la collecte de ces points communs et à l'étude du déroulement de ces négociations, nous montrons qu'il est possible de concevoir un protocole général de négociation, paramétrable, qui offre la possibilité de formuler des contre-propositions et qui permet de décrire le déroulement d'une négociation particulière.

Nous proposons un modèle général de négociation utilisant ce protocole et possédant un mécanisme de gestion des négociations, qui permet de négocier simultanément des contrats portant sur des ressources disjointes, et de négocier séquentiellement les contrats entrant en conflit. Ce modèle permet également de renégocier automatiquement des contrats qui ne peuvent plus être honorés.

Notre proposition est basée sur une architecture à trois niveaux, séparant la partie communication entre agents, la partie négociation et la partie stratégie de négociation d'une application. En effet, la façon dont communiquent les agents n'a pas d'influence sur la façon de négocier, et différents moyens de communication peuvent être utilisés pour une même application de négociation qui serait exécutée dans des environnements différents.

Nous montrons aussi qu'il est important de séparer la stratégie de négociation des deux autres niveaux pour permettre à un utilisateur de choisir comment il va négocier sans que cela ne porte à conséquence au reste de l'application. De plus, la stratégie de négociation est intrinsèquement liée à l'application de négociation, et il est bien évident que négocier une tonne de pommes de terre ne se fait pas de la même façon que de négocier un créneau horaire pour un rendez-vous ou encore l'utilisation d'une ressource partagée en exclusivité pendant une heure.

Notre modèle a été implémenté sous la forme d'une API Java également appelé *GeNCA*. Afin de valider notre modèle, nous avons utilisé notre API pour réaliser plusieurs applications comme un système de vente aux enchères, un système de prise de rendez-vous, et un système pour négocier le choix d'un restaurant pour une sortie commune.

Mots-clés: négociation, agent, protocole d'interaction, intelligence artificielle, génie logiciel.

Abstract

When numerous agents interact, conflicts may arise. To solve them, different ways can be used such as cooperation, voting systems or negotiation. The one which interests us in this thesis is contract-based negotiation over resources.

The aim of our work is to conceive a general negotiation model (called *GeNCA* : *Generic Negotiation of Contracts API*), and to give an implementation of it. This allows a user wishing to develop a negotiation application not to have to do the whole job but to have a model that will facilitate his work.

In order to conceive such a model, we begin by studying the different types of negotiation most used. The analysis of these negotiations leads to the collect of their common points, and provides us a basis to conceive our model. Thanks to the collect of these common points and the study of the progress of these negotiations, we show that it is possible to conceive a general negotiation protocol, parameterable, which offer the possibility to make counter-proposals and which can describe the progress of a particular negotiation.

We propose a general negotiation model using this protocol and having a management of negotiations mechanism, which allows to negotiate contracts on disjoint sets of resources in parallel, and to negotiate contracts having conflicts on resources sequentially. This model also allows to automatically renegotiate contracts that cannot be met any longer.

Our proposition is based on a three-level architecture, that separates the communication part between agents, the negotiation part and the negotiation strategy part of an application. As a matter of fact, the way agents communicate doesn't play a role in the way negotiation is made, and different communication ways can be used in a same application executed on different environments.

We equally show that it is important to separate the negotiation strategy from the two other levels, to allow a user to choose which negotiation strategy he will use without disturbing the remaining of the application. Moreover, the negotiation strategy is intrinsically linked to the negotiation application, and it is obvious that negotiating a ton of potatoes is not the same as negotiating a slot-time for an appointment, nor the same as negotiating the exclusive use of a shared resource for an hour.

Our model has been implemented by a Java API also called *GeNCA*. In order to validate our model, we used our API to achieve different applications such as an auction system, an appointment taking system and a system to negotiate the choice of a restaurant for a common trip.

Keywords: negotiation, agents, interaction protocol, artificial intelligence, software engineering.