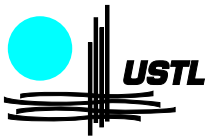

Laboratoire d'Informatique
Fondamentale de Lille

Rapport de stage de DEA

Etude de la négociation entre agents autonomes

Marie-Hélène Verrons

Février - Juin 2001

Table des matières

Table des figures

Liste des tableaux

Remerciements

Je tiens à remercier Mr le Professeur Philippe Mathieu pour m'avoir proposé ce sujet de stage. Je remercie également tous les membres de l'équipe SMAC pour leur accueil et leur disponibilité.

Je remercie plus particulièrement Philippe pour nos échanges de points de vue et d'idées, et pour son aide dans les démarches de recherche et ses directives pour le développement de l'API. Je remercie également Jean-Christophe et Yann pour les explications qu'ils m'ont données, tant sur l'implémentation Magique que sur le langage Java. Je remercie enfin Bruno pour les problèmes matériels qu'il a résolu.

Avant-propos

Ce mémoire présente l'activité que j'ai menée durant mon stage de DEA au sein de l'équipe Systèmes Multi-Agents et Coopération (SMAC), sous la direction de Mr le Professeur Philippe Mathieu. L'équipe SMAC mène de front trois axes de recherche :

- l'étude des architectures des systèmes multi-agents (SMA),
- l'étude de la modélisation du comportement,
- l'étude des mécanismes de négociation.

Ce stage s'inscrit dans le cadre de la négociation dans les SMA. Ce sujet est actuellement très porteur pour deux raisons. La première est que la négociation s'est imposée comme étant la meilleure solution pour résoudre des conflits au sein d'un SMA. D'autre part, la négociation, de son point de vue économique, est à la base de nombreuses applications commerciales, notamment dans le commerce électronique avec les systèmes de vente aux enchères, mais elle est aussi présente dans de nombreux jeux (comme par exemple Diplomacy), thème que le CNRS nous conseille de développer.

Le but de ce stage était de trouver les parties communes au plus grand nombre d'applications de négociation possibles, afin de créer des outils informatiques réutilisables permettant de faciliter la mise en oeuvre de la négociation au sein des SMA.

Notre démarche a été la suivante :

1. Etudier les précédentes implémentations.
2. Dégager les points communs aux différents exemples d'applications.
3. En déduire un protocole de négociation générique.
4. Implémenter les différents exemples en utilisant ce protocole.

Ce travail avait déjà fait l'objet du stage de DEA de Julien Fau l'année passée, nous y avons apporté notamment la généricité du protocole et l'écriture d'une API permettant la négociation de ressources.

La deuxième partie de mon stage consistait à effectuer un tour d'horizon des différentes plate-formes existantes et plus particulièrement étudier la plate-forme ZEUS de British Telecommunications, afin de comparer leur approche de la négociation avec la nôtre.
Nous en avons maintenant une bonne connaissance qui nous permet de présenter dans ce rapport une comparaison pertinente.

Chapitre 1

Introduction

Ce stage est dans la continuation des travaux réalisés par A. Taquet et J. Fau. A. Taquet [Taquet 96] avait réalisé une API de négociation spécifique à la prise de rendez-vous. J. Fau [Fau 00] avait quant à lui essayé d'élaborer une API générique pour la négociation, et avait conclu qu'il était impossible d'en créer une. Il avait pour cela étudié la prise de rendez-vous, la vente aux enchères et le troc.
Nous pensons cependant qu'il est possible de réaliser une API générique et portable pour effectuer de la négociation.

L'objectif de ce stage était de concevoir une API Java polyvalente pour la négociation entre agents autonomes.
En effet, la négociation est de plus en plus présente (sur Internet par exemple), et on la retrouve dans différents types d'applications tels que la prise de rendez-vous, la vente aux enchères, les places de marché ou encore dans les jeux tels Diplomacy.
La négociation est un processus de dialogue entre différentes personnes. Elle s'effectue sur un contrat pour obtenir des ressources qui sont communes aux différents acteurs de la négociation, et sur demande d'un initiateur. Elle réunit un ensemble de participants et se déroule jusqu'à ce qu'un accord satisfaisant une majorité de participants soit trouvé.

Nous avons étudié différentes applications de négociation et nous avons conclu que le système mis en place est toujours le même. Nous avons donc élaboré un protocole générique de négociation, qui permet de négocier à plusieurs sur plusieurs ressources, de prendre en compte des priorités pour les participants et pour les ressources, et qui renégocie automatiquement les contrats qui n'ont pas abouti lors de leur proposition.

Nous avons par la suite réalisé une API Java qui met en œuvre ce protocole.
Afin de proposer une API générique, nous avons créé des ressources et des contrats abstraits. Nous avons également défini un comportement par défaut pour les initiateurs et les participants ; ce qui leur permet de prendre les décisions nécessaires, par exemple lors de la réception d'une proposition de contrat ou d'une réponse.

Ainsi, il suffit à l'utilisateur de définir les ressources à négocier et les contrats qu'il veut proposer pour utiliser notre API.

L'API que nous avons réalisée est portable, polyvalente et adaptable aux différentes applications de négociation, même si quelques réglages sont parfois nécessaires pour réaliser une application précise. En effet, nous proposons un comportement par défaut pour les initiateurs et les participants qui ne peut parfois pas convenir à l'application réalisée.
Notre API fonctionne aussi bien sur Internet pour des places de marché, que dans les jeux vidéos, par exemple. De plus, elle est la plus générique possible afin de n'avoir à écrire que le strict minimum. Cette API permet également de gérer de multiples ressources simultanément, et ce de façon transparente vis à vis de l'utilisateur.

Grâce à notre API, il est possible de réaliser différentes applications de négociation, et de mélanger des agents virtuels et des agents humains. Cette API fonctionne en réseau hétérogène et les communications se font de façon asynchrone.

Chapitre 2

Présentation des SMA

2.1 Présentation générale

2.1.1 Qu'est-ce qu'un agent ?

De nombreux travaux ont été réalisés sur cette notion d'agent et il en résulte de nombreuses définitions.
Smith définit un agent comme étant "une entité logicielle persistante dédiée à un but spécifique", *Selker* considère les agents comme "des programmes informatiques qui simulent les relations humaines en faisant des choses qu'une autre personne pourrait faire", et *Janca* définit l'agent comme "une entité logicielle à qui l'on peut déléguer des tâches". Ces définitions sont assez pauvres, les agents sont considérés comme des subalternes, tandis que pour *Jennings* [Jennings 96] un agent est un programme autonome capable de contrôler ses faits et gestes, il se base sur la perception qu'il a de son environnement, et il est à la poursuite d'un ou plusieurs objectifs.
Dans son livre [Ferber], *Ferber* définit l'agent comme "une entité physique ou virtuelle

1. qui est capable d'agir dans un environnement,
2. qui peut communiquer directement avec d'autres agents,
3. qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),
4. qui possède des ressources propres,
5. qui est capable de percevoir (mais de manière limitée) son environnement,
6. qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),
7. qui possède des compétences et offre des services,
8. qui peut éventuellement se reproduire,
9. dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit."

Pour nous, un *agent* est une entité logicielle autonome et réactive. Il possède ses propres connaissances et son propre savoir-faire qu'il met au service des autres. Il a la capacité de communiquer que ce soit avec d'autres agents ou un opérateur humain.

Un agent possède quatre caractéristiques principales :
- l'*autonomie* : les agents doivent être capables de réaliser la majorité de leurs tâches sans intervention directe d'un humain, ou d'autres agents. Ils doivent également avoir un certain degré de contrôle sur leurs actions et leur état interne.
- la *sociabilité* : les agents doivent être capables d'interagir, lorsqu'ils le jugent utile, avec d'autres agents et des humains dans le but de résoudre leur problème et d'aider les autres dans leurs activités si nécessaire.
- la *réactivité* : les agents doivent percevoir leur environnement (qui peut être le monde physique, un utilisateur, une collection d'agents, internet...) et réagir automatiquement à un changement qui s'y produirait.
- la *proactivité* : les agents ne doivent pas seulement réagir face à leur environnement, mais aussi être capables d'exhiber un comportement opportun et prendre des initiatives quand il le faut.

2.1.2 Qu'est-ce qu'un système multi-agents ?

Si la notion d'agent est discutable et discutée, celle de *système multi-agents* est beaucoup plus claire : un système multi-agents (ou SMA) est un ensemble d'agents logiciels ou humains qui communiquent entre eux et travaillent ensemble pour résoudre un objectif commun. Travailler ensemble signifie aussi bien collaborer que négocier ou rivaliser selon le problème.

Trois thèmes fondamentaux émergent des SMA :
- Interactions
- Coopération
- Action et réaction

Les interactions

Les agents qui agissent dans un même système provoquent des changements de celui-ci, et ceux-ci influencent tous les occupants de ce système. *Ferber* [Ferber] propose la définition des interactions suivante, basée sur ce principe : "on appellera situation d'interaction un ensemble de comportements résultant du regroupement d'agents qui doivent agir pour satisfaire leurs objectifs en tenant compte des contraintes provenant des ressources plus ou moins limitées dont ils disposent et de leurs compétences individuelles."
Différents types d'interactions existent, et nous allons reprendre quelques critères proposés par *Ferber* afin de les identifier.

Notion de compatibilité des buts Le but d'un agent est l'état qu'il veut atteindre parmi tous ceux qui lui sont accessibles. Pour atteindre son but, depuis son état initial, l'agent doit passer par des états transitoires. Dans un système multi-agents, chaque agent a un but. Le problème est de savoir si les buts des différents agents sont concordants ou non, c'est-à-dire si les objectifs des agents sont contradictoires ou non.
définition : Le but d'un agent A est incompatible avec celui d'un agent B si les agents A et B ont comme buts respectifs d'atteindre les états décrits respectivement par p et q, et que p $\Rightarrow \neg$ q, c'est-à-dire que : satisfait(but(A,p)) $\Rightarrow \neg$ satisfait(but(B,q)).

Cette distinction entre buts compatibles et incompatibles permet de dire que des agents sont dans une situation de coopération ou d'indifférence, si leurs buts sont compatibles, et dans une situation d'antagonisme sinon. Les situations d'antagonisme peuvent mener à des conflits dont les causes principales sont :

La relation aux ressources : chaque agent dispose, dans son environnement, de ressources limitées. Il peut s'agir de valeurs énergétiques (RealTimeBattle), de valeur financières (vente aux enchères), d'outils, de matières premières, ou plus basiquement de temps (prise de rendez-vous) ou d'espace. Lorsque plusieurs agents ont besoin des mêmes ressources, au même moment et au même endroit, des situations de conflit apparaissent.

La notion de capacité des agents par rapport aux tâches : il s'agit de savoir si un agent peut effectuer seul la tâche qu'on lui a demandé d'accomplir. Dans le cas contraire, il a besoin des autres pour parvenir à son but. Cette situation, sans mener les agents à des conflits, va nécessiter une interaction : pourquoi me détournerai-je de mon but et dépenserai-je des ressources pour t'aider ?

Afin de résoudre ces conflits, il existe différents comportements, tels la loi du plus fort, l'arbitrage et la négociation que nous aborderons dans le prochain chapitre.

La coopération

Nous avons cité ci-dessus une notion importante pour les SMA : la coopération. Nous allons maintenant la définir. Nous avons dit que quand les buts des agents étaient compatibles, deux situations se produisaient :

- *l'indifférence* : quand les ressources et les compétences des agents sont suffisantes, ils n'ont aucun besoin de communiquer.
- *la coopération* : quand les agents ne possèdent pas les ressources et/ou les compétences nécessaires à la résolution de leur problème, ils doivent alors s'organiser pour partager ressources et compétences afin d'atteindre leur but.

C'est cette organisation induite par les insuffisances des agents que nous appellerons coopération. Les avantages attendus sont nombreux :

- *augmentation des performances globales du système* : par amélioration des délais de réponse par exemple.
- *amélioration de la tolérance aux pannes* : c'est le cas de robots explorateurs, avec un seul robot, toute panne met fin à l'exploration. Mais avec plusieurs robots, il est tout à fait possible d'imaginer une réorganisation de la colonie en cas de panne de l'un des siens. Ainsi, l'exploration pourrait se poursuivre.
- *améliorations individuelles des agents* : elle peut être obtenue par la spécialisation. D'autres améliorations sont bien entendu envisageables, comme l'échange de compétences. Dans le cas de la spécialisation celle-ci peut permettre à un agent de mieux accomplir un type de tâche. Attention toutefois à l'augmentation induite de la dépendance de l'agent envers les autres.
- ...cette liste n'est pas exhaustive, et de nouveaux avantages peuvent apparaître avec l'évolution des SMA.

Interaction et coopération des agents doivent permettre au système d'agir de la façon la plus autonome possible pour résoudre un problème. De même, cela doit permettre une meilleure réactivité aux situations nouvelles.

Les actions et réactions

Nous avons présenté les avantages de l'utilisation de plusieurs agents. Mais nous n'avons encore rien dit de leur intelligence. Les systèmes multi-agents sont composés de deux familles présentant une gradation de l'intelligence individuelle : les SMA réactifs et les SMA cognitifs.

Les SMA réactifs Il s'agit des SMA les plus simples. Les agents ne diffèrent pas des programmes simples. Ils sont composés d'un ensemble de méthodes dont les appels sont prédéfinis. On sait que telle méthode appelle telle autre, et ainsi de suite. L'agent, donc le système, ne peut prendre d'initiatives. Il se contente de réagir, d'où sa qualification de réactif. Si le système arrive dans un cas qui n'a pas été prévu, il se bloque. Il faut donc envisager des agents plus évolués.

Les SMA cognitifs Dans cette famille, on trouve les agents dotés de bases de connaissances, de bases d'actions et de langages évolués tels que KQML [KER 99]. Ils sont autonomes, et plus aptes à improviser que leurs congénères. Bien sûr, ils peuvent disposer de plus ou moins "d'intelligence" selon les besoins du système. Souvent, l'implantation d'un ou deux algorithmes issus de l'IA suffisent amplement à leur permettre de résoudre les problèmes auxquels ils sont confrontés.

Les SMA peuvent donc être adaptés, du point de vue de la complexité des raisonnements qu'ils peuvent produire, aux problèmes qu'ils doivent résoudre. Les SMA sont une réponse

à des problèmes de conception, ils représentent une nouvelle approche de la résolution des problèmes. En effet, la vie courante nous fournit un nombre important d'exemples de systèmes qui peuvent être appelés "multi-agents". Il s'agit des écosystèmes, des chaînes de production robotisées dans les usines, des voitures sur la route,...Souvent, le modèle multi-agents s'impose naturellement.

Les SMA présentent trois intérêts principaux :
- La distribution de la connaissance
- La distribution du savoir-faire
- La puissance de calcul

La distribution de la connaissance La distribution de la connaissance présente l'intérêt de ne pas obliger les agents à se référer à une base de connaissances unique. Le temps d'accès à ces connaissances est ainsi supprimé. Prenons l'exemple des agents explorateurs. Plutôt que de toujours rendre compte à une base de connaissances, ils gardent leurs informations. Ces informations peuvent être remontées régulièrement, et dans ce cas, il n'y aura pas de problèmes de communication, puisque la demande d'information viendra du sommet de la hiérarchie. Les accès concurrents sont supprimés. De plus, en se rencontrant, les agents peuvent s'échanger leurs connaissances respectives, obtenant ainsi des informations sur la région voisine qui leur seront vraiment utiles. Il n'aurait en effet servi à rien de leur redescendre les connaissances relatives à l'autre extrémité du domaine à explorer. Ici, les connaissances seront formalisées sous la forme d'une carte du territoire.

La distribution du savoir-faire Cette distribution est évidente dans le cas des chaînes de montage dans les usines automobiles. Il n'existe aucun robot capable de faire le montage seul, mais si nous comparons les robots à des agents, ceux-ci savent accomplir des tâches individuelles qui participent au montage d'une voiture. Aucun agent ne peut le faire, mais le système en est capable.

La puissance de calcul Cette distribution est destinée à permettre aux agents de tirer partie de la puissance des réseaux. Puisque nous savons qu'avec un réseau hétérogène, des performances similaires à celles de certaines machines parallèles, en temps de calcul, peuvent être obtenues, nous allons en tirer partie. L'approche multi-agents doit permettre de disperser sur le réseau des agents calculateurs. Ceux-ci peuvent être simples ou avoir des notions de la charge de la machine, de la quantité de travail restant à acomplir ...Cette application des SMA se situe à la frontière de la programmation parallèle. Les termes d'évaluation des performances et d'amélioration du système seront repris à ce domaine.

2.2 L'API Magique

Magique est une plate-forme de développement de systèmes multi-agent, développée par l'équipe SMAC du LIFL à l'université de Lille1. Les concepts fondamentaux de cette

plate-forme ont fait l'objet de la thèse de N. Bensaid [Bensaid 99, Bensaid 97] soutenue au LIFL. Elle constitue l'un des principaux thèmes de recherche de l'équipe.

Motivations

Ses motivations étaient de fournir un contexte, c'est-à-dire un environnement, une méthodologie et un langage, facilitant le développement de SMA et donc d'agents. Cette capacité s'appuie sur 5 points :

- faciliter l'organisation des agents dans un SMA, et en particulier répondre au problème du choix de la structure du SMA.
- favoriser la réutilisabilité des agents : le développement d'un agent doit être au maximum indépendant du développement d'un(e) (application) SMA particulier(ère).
- proposer un environnement de développement et de déploiement des agents.
- portabilité et distribution sur réseau.
- se baser sur un langage existant (ne pas réinventer la roue).

Magique est l'acronyme de Multi-AGent hiérarchIQUE :

Un SMA est organisé à sa création en une hiérarchie stricte d'agents. Les liens hiérarchiques représentent les canaux de communication par défaut entre agents. Le contrôle est distribué et la structure hiérarchique permet de le factoriser et de contrôler la granularité de cette distribution. L'évolution du SMA est dynamique : la structure hiérarchique peut être remise en cause au profit de relations privilégiées (accointances), et les agents peuvent être ajoutés et retirés dynamiquement.

Superviseurs et spécialistes

On distingue deux statuts d'agents :

- `superviseur` :il est à la racine d'une (sous) hiérarchie.
- `spécialiste` : il est une feuille de la hiérarchie.

L'acheminement des messages et le contrôle sont à la charge des superviseurs. Un agent possède des compétences, il a une tâche à accomplir dont la réalisation requiert l'exploitation des compétences que l'agent possède ou non. Un mécanisme de délégation de compétences est donc mis en place.

Délégation de compétences

Prenons l'exemple de la Figure 2.1, l'agent `S1` doit accomplir une tâche nécessitant les compétences `c`,`c1` et `c2`.

La compétence `c` est connue en interne, il n'y a donc pas de problème.
La compétence `c1` est possédée par un agent "en-dessous" de lui, `S1` la délègue donc à qui de droit.
En revanche, la compétence `c2` n'est possédée ni par `S1`, ni par un agent qu'il supervise, `S1` transmet donc une requête de réalisation à son superviseur `S` qui répète le processus. `S`

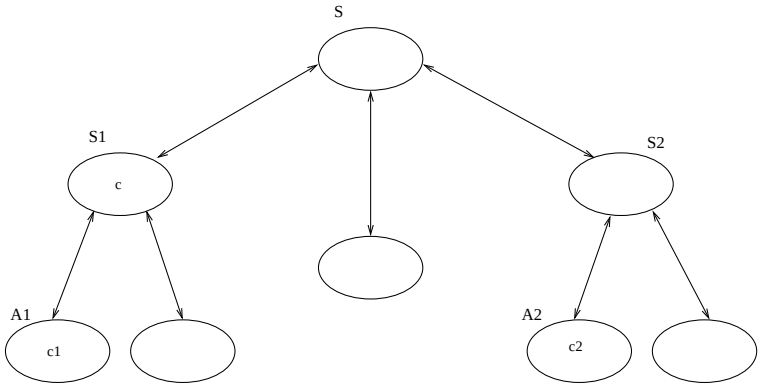

FIG. 2.1 – Exemple de délégation de compétences.

transmet à `S2` qui connaît un agent compétent, `S2` applique le même processus : la requête arrive à l'agent compétent `A2` qui la réalise.

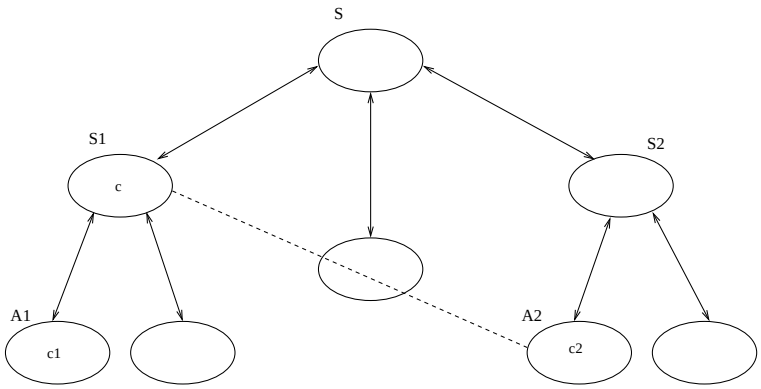

FIG. 2.2 – Exemple de délégation de compétences avec accointance.

Prenons maintenant l'exemple de la Figure 2.2 : l'agent `S1` doit accomplir une tâche nécessitant les compétences `c`,`c1` et `c2`.

La compétence `c` est connue en interne, il n'y a donc pas de problème.
La compétence `c1` est possédée par un agent "en-dessous" de lui, `S1` la délègue donc à qui de droit.
`S1` a pour accointance `A2` qui est compétent, il lui transmet directement la requête.

Intérêt

Deux intérêts principaux sont à dégager :

1. La souplesse de développement : qui comprend la réutilisabilité (l'agent réalisateur n'a pas à être connu a priori), et l'indépendance agent↔ SMA, en effet, il suffit d'avoir dans le SMA un agent compétent sans présupposer de sa localisation dans le système.

2. L'accroissement de fiabilité : l'agent réalisateur peut changer, et les agents peuvent être réorganisés, ce qui implique une résistance aux pannes, une efficacité (en cas de surcharge d'un agent compétent) et la gestion de la concurrence.

La dynamicité

La dynamicité se situe à trois niveaux :

1. structurel : par l'ajout et le retrait d'agents,
2. communication :
   - création de relations d'accointances privilégiées.
   - auto-organisation du SMA en fonction de son "exécution" : réponse au problème du choix de la "bonne" structure et non déterminisme de l'organisation d'une session à l'autre.
3. individuel : échange de compétences entre agents : évolutivité et autonomie.

Magique : l'API

L'API Magique est une sur-couche de Java qui permet la distribution des agents sur réseau hétérogène. Elle fournit un environnement de construction et déploiement de SMA où les communications peuvent être au choix synchrones ou asynchrones et où la délégation de compétences est transparente. L'API ne fournit pas de moyen particulier de gestion de la connaissance, de la représentation des autres ou de l'expression du contrôle ; pour l'équipe SMAC, tout ceci est au niveau du service à mettre dans l'Agent.
L'agent Magique au départ est minimal, il ne sait qu'apprendre et communique simplement avec ses semblables. Le rôle du concepteur de l'application consiste à lui ajouter les compétences nécessaires pour mener à bien sa tâche.

L'API est disponible via le web : http ://www.lifl.fr/MAGIQUE/

Chapitre 3

Qu'est-ce que la négociation ?

3.1 Définition de la négociation

Avant de présenter notre protocole, il convient de définir ce qu'est la négociation.
D'après le *Petit Robert*, la négociation est *une série d'entretiens, d'échanges de vues, de démarches qu'on entreprend pour parvenir à un accord, pour conclure une affaire.*
Le dictionnaire *Larousse* propose la définition suivante : *art, action de mener à bonne fin les grandes affaires, les affaires publiques.*
De ces deux définitions, celle du *Petit Robert* nous paraît être la plus proche du modèle que nous voulons proposer. En effet, elle prend en compte la notion de *dialogue* et celle d'*accord* entre les différents participants.

Pour le monde informatique, il est nécessaire de formaliser la notion de négociation. De nombreux travaux ont été menés sur le sujet et il en résulte différentes définitions que nous allons présenter maintenant.

Shwartz & Kraus [Kraus 97] : "La négociation est vue, dans l'Intelligence Artificielle Distribuée (IAD) comme un moyen donné aux agents pour communiquer et faire des compromis, afin d'arriver à un état d'accord mutuel bénéfique."

Cette définition est celle qui se rapproche le plus de notre perception intuitive de la négociation. Elle veut se rapporter à l'IAD, un des domaines qui a le plus inspiré l'approche multi-agents. Elle propose également la notion très intéressante *d'accord mutuel bénéfique.* Bien que la notion d'accord reste à définir, au même titre que la notion de compromis, les termes *mutuel* et *bénéfique* sont très importants. L'idée d'accord mutuel implique le consentement de tous les agents impliqués, donc qu'il y a une majorité d'agents satisfaits. Le terme *bénéfique* implique que la négociation permet d'économiser temps et ressources dans la résolution d'un conflit, et même qu'elle permet aux agents négociateurs d'obtenir plus d'avantages par la négociation.

Beer & others [KER 99] : "Comme les agents sont autonomes et ne peuvent être supposés bénévoles, ils doivent influencer les autres agents à agir de certaines manières ; et ici, la négociation est critique pour gérer de telles interdépendances."

Ici, les relations entre agents sont vues comme étant commerciales par nature. La négociation est alors vue comme un moyen d'obtenir des autres agents un comportement qui ne soit pas forcément le meilleur pour eux...Cette définition reste très vague quant à la nature même de la négociation. Elle précise seulement quels en sont les causes et les effets. L'idée d'influence est importante, car négocier, c'est exactement cela : exercer une influence. Amener un ou plusieurs autres agents à prendre une décision qui sera dans l'intérêt de l'initiateur de la négociation. Il est tout de même capital d'insister sur le fait que la négociation doit être une relation où tous les acteurs sont gagnants. Ce n'est pas forcément acquis d'après cette définition.

Straus [Straus] : "La négociation est un processus par lequel des agents s'accordent sur un arrangement mutuel pour résoudre ou éviter des conflits."

Straus donne une définition également vague sur la nature de la négociation : un processus. Il définit donc la négociation par son but : "résoudre ou éviter des conflits". Bien sûr, cette vision est un peu réductrice. Elle ne prend pas en compte les cas où des agents négocieraient juste pour améliorer leur situation, sans qu'il y ait risque de conflit. Il faut prendre ce cas en compte.

Nick R. Jennings : "La négociation peut être vue comme une recherche distribuée dans un espace d'agréments potentiels."

Cette définition est très intéressante car elle insiste sur le caractère distribué de la négociation. En effet, chacun des agents, en annonçant ses prétentions, renseigne les autres sur ce qu'il veut. Le but final étant de trouver un accord mutuel. Les volontés de chaque agent sont donc vues comme des contraintes sur la solution. Ceci permet de dire que si les contraintes sont antinomiques, le processus de négociation est voué à l'échec.

Comme ces précédentes définitions restent floues et ne nous conviennent pas vraiment, nous proposons donc notre propre définition de la négociation :
La négociation s'effectue sur un *contrat* pour des *ressources* communes et sur demande d'un *initiateur*. Elle réunit un ensemble de *participants* et se déroule jusqu'à ce qu'un accord satisfaisant un pourcentage minimum de participants soit trouvé. Les participants cherchent à obtenir la meilleure solution possible pour partager les ressources, c'est-à-dire celle qui satisfait au mieux tout le monde.

3.2 La négociation dans les SMA

Nous avons vu dans le chapitre précédent que les interactions entre les agents peuvent engendrer des conflits, et il apparaît qu'il est plus intéressant de les résoudre grâce à la négociation, c'est-à-dire d'inciter les agents à communiquer pour trouver une solution qui les satisfait tous, ou qui en satisfait la plupart. Ainsi, la négociation évite de se retrouver avec quelques "super-agents" (les plus forts) et une majorité d'autres agents qui n'atteindront jamais leur but. De plus, elle évite de paralyser le système en immobilisant un certain temps un ou plusieurs agents pour leur faire arbitrer un conflit. En négociant, seuls les agents impliqués dans le conflit donnent de leur temps pour résoudre le problème.

A l'heure actuelle, la négociation dans les SMA a déjà fait l'objet de très nombreux travaux. La motivation de ces travaux diffère beaucoup. Des travaux américains tels ceux de Pattie Maes [Maes 97] ou Katia Sycara ont été basés sur la négociation du point de vue économique. Ils ont ensuite inspiré beaucoup de travaux introduisant la négociation dans l'intersection entre le monde des SMA et celui de l'e-commerce.

Mais il y a également eu beaucoup de travaux formels réalisés, afin d'essayer de fournir des modèles de négociation [KER 99]. De tous les travaux réalisés, il ressort un certain nombre de points communs sur lesquels il faut aujourd'hui s'appuyer. Trois éléments prédominent : un protocole de négociation, des ressources à négocier et un modèle de raisonnement.

Un protocole D'après *Beer & others* [KER 99], "un protocole de négociation est l'ensemble des règles qui gouvernent cette interaction."

Toutefois, le foisonnement des travaux réalisés montre qu'il existe un protocole bien particulier à chaque type de négociation. En fait, bien souvent, le protocole est créé en fonction du type de négociation qui va être implémenté par l'équipe. C'est pourquoi, pour faire un protocole global, capable de représenter tous les types de négociation, il faudrait soit les inclure tous, soit trouver des règles abstraites capables de représenter tout type de négociation. La plupart des chercheurs considèrent qu'il est impossible de trouver un seul protocole pour toutes les applications de négociation existantes.

Les ressources à négocier Ceci recouvre tout ce qui va être négocié. Ce sont soit des objets à proprement parler, pour les enchères, ce seront des articles ; soit des services que peut rendre l'agent. Chaque ressource possède des caractéristiques, qui peuvent être variables ou non. Dans le cas de caractéristiques variables, comme par exemple le prix d'un article dans le cadre de la vente aux enchères, cela engendre une complexité supplémentaire dans le processus de négociation.

Un modèle de raisonnement des agents Ce modèle fournit un algorithme de décision aux agents. La sophistication du modèle étant directement fonction du protocole envisagé,

de la nature des ressources et des opérations qui peuvent être accomplies sur ces dernières. La prise de décision peut être très simple et aller jusqu'à des cas très complexes. La stratégie adoptée par l'agent peut faire appel à la théorie des jeux. Ce thème, abordé par l'équipe SMAC, a aussi été traité du point de vue de son application à la prise de décision dans la négociation [Jennings 00-1].

D'autres équipes ajoutent d'autres éléments, tels les actes de langage, l'évaluation de la satisfaction ou de la valeur des contrats et la capacité d'argumenter sa position.

Les actes de langage Ces actes couvrent l'ensemble des messages qui seront échangés lors du processus de négociation. Parmi les diverses représentations des actes de langage, les prédicats sont très employés. Ils permettent de représenter simplement des faits tels que : possède($Article_1$,$Agent_x$) $\rightarrow$ vrai. Une fois échangés, ils sont naturellement exploitables par des agents cognitifs possédant la base de connaissances appropriée.

Dans les applications réalisées au sein des différents laboratoires, des langages de communication ont été utilisés pour permettre aux agents de s'acquitter des actes de langage. Parmi eux, il faut citer KQML et FIPA ACL [KER 99]. Ces deux langages sont dits "de haut niveau". Ils formalisent les communications entre agents par envois de chaînes de caractères. Le format de ces chaînes permet ensuite de retrouver le corps du message et l'ontologie du message qui est expédiée en même temps. Un bilan est fait dans l'article [KER 99]. Les deux leçons tirées sont les suivantes : ACL comporte des défauts mineurs qui empêchent son utilisation dans les SMA de grande échelle. Et d'autre part, "un autre point de loin plus important est que KQML et ACL sont trop académiques pour beaucoup d'applications (...) Quand des humains ou des agents logiciels négocient des enchères, ils n'ont pas nécessairement besoin d'actes de langage et de logique pour mener à bien des transactions sophistiquées. En fait, la plupart des applications ne les utilisent pas du tout."

Donc, nécessaires, les actes de langage sont souvent formalisés spécifiquement pour une application. A chaque application ses actes.

L'évaluation de la satisfaction ou de la valeur des contrats Pour raisonner, suivre des stratégies ou bien savoir s'ils sont satisfaits, les agents ont besoin d'avoir une fonction d'évaluation. Toute la théorie sur ce sujet est donnée par l'équipe Sierra, Faratin, Jennings [Jennings 99-1, Jennings 97-2]. Nous dirons seulement que les agents doivent disposer d'un moyen d'évaluation des contrats proposés. Cette évaluation est subjective. L'intérêt est que deux agents n'accorderont alors pas la même valeur à un même contrat, facilitant ainsi le processus de négociation.

La capacité d'argumenter sa position Cet élément est apporté par Nick R. Jennings [KER 99, Jennings]. Il ne nous paraît pas être une absolue nécessité. En effet, argumenter sa position est difficile. Cela nécessite des capacités de raisonnement avancées que

tous les agents ne possèdent pas selon le type de système auquel ils appartiennent. Nous dirons donc que l'argumentation est très intéressante mais ne doit constituer pour nous qu'une évolution possible du protocole en vue de son adaptation à des agents cognitifs.

Nous avons vu qu'il existe plusieurs points de vue sur la définition de la négociation et que sa formalisation diffère selon les équipes de recherche. Nous allons maintenant examiner les différents protocoles et implémentations réalisés jusqu'à présent.

Chapitre 4

L'existant

La négociation dans les SMA a déjà fait l'objet de nombreux travaux, nous allons commencer par examiner la première forme sous laquelle elle est apparue : le Contract Net Protocol ; puis nous allons faire un tour d'horizon des différentes plate-formes de négociation qui ont été implémentées.

4.1 Le Contract Net Protocol

Dans les travaux sur la négociation, le *contract net* [Smith] est la première forme assimilable à de la négociation. En tout cas, il permet de créer facilement des applications de négociation.

Le *contract net* est basé sur les échanges qui régissent le commerce : l'offre et la demande. Un agent appelé "manager" veut déléguer une tâche à un "contracteur".
Le manager va :

1. annoncer la tâche dont il veut qu'elle soit accomplie,
2. recevoir et évaluer les propositions des contracteurs,
3. choisir une de ces offres,
4. recevoir et synthétiser les résultats.

Un contracteur va :

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.

Gerhard Weiss définit ainsi le *contract net* : "un protocole supportant la mise en relation des tâches à accomplir avec les agents (contracteurs) qui veulent et sont capables de les réaliser. Le "contract net" sert souvent de référence à des négociations basées sur des algorithmes d'allocation de tâches."

Nous en dirons donc qu'il est basé sur un échange de contrats. C'est là-dessus également que se base notre protocole. Il est important de remarquer que ce protocole permet d'effectuer de la négociation de 1 agent (le manager) vers n agents (les contracteurs). En effet, il est plus difficile de faire négocier plusieurs agents que d'en faire négocier uniqument deux car il faut tenir compte de plusieurs avis. La principale critique formulée contre le *contract net* est qu'il ne permet pas de formuler des contre-propositions, les contracteurs ne peuvent pas donner leur "avis", si la première proposition échoue, le manager , s'il le désire, peut proposer un nouveau contrat mais sans savoir s'il va dans le sens des contracteurs.
De nombreuses reprises de ce protocole ont été réalisées, et c'est un des travaux de recherche les plus plébiscités dans la négociation dans les SMA.

4.2 Kasbah de Pattie Maes

Kasbah [Maes 96] a été développé au MIT Media Lab, 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/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.

4.3 AuctionBot

AuctionBot [AuctionBot] est un serveur d'enchères expérimental développé et opératif au laboratoire d'intelligence artificielle de l'université du Michigan. Son but est de permettre à n'importe quel internaute de participer aux enchères sur le net.
AuctionBot est une plate-forme 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 ...) 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.

4.4 Traconet de Tuomas Sandholm

Le projet de Tuomas Sandholm [Sandholm 93] 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 companies 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 de la négociation de 1 vers n agents.

4.5 Le projet ADEPT

Ce projet [Adept] a été réalisé afin de développer un modèle de négociation orientée service [Jennings 97]. Le cadre de cette application 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 paramètrés de la négociation au cours du temps.

Plusieurs tactiques ont été implémentées, insistant sur la valeur du temps (ressource limitée lui aussi), les ressources matérielles,...La convergence du système n'est pas automatique. Ces travaux ont ensuite beaucoup inspirés leurs concepteurs qui ont ensuite défini la "Foundation of Intelligent Physical Agent" [Jennings 99-2] : un autre protocole qui a servi de base à une autre implémentation.

4.6 Conclusion

Les différents systèmes présentés dans ce chapitre utilisent des protocoles propres à leur application. Certains sont spécialisés dans les enchères, d'autres sur les services mais aucun d'eux ne se présente comme étant générique. De plus, ils n'offrent pas tous les mêmes possibilités de communication, certains utilisant un modèle bilatéral, d'autres un modèle de 1 vers n agents. Enfin, ils utilisent des actes de langages spécifiques, ce qui limite leur portabilité.

Nous allons maintenant présenter notre protocole de négociation qui permet de gérer plusieurs négociations simultanées ainsi que des négociations de 1 vers 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.

Chapitre 5

Le protocole de négociation proposé

Nous allons proposer ici un protocole de négociation entre agents autonomes qui permet à plusieurs agents de négocier entre eux sur des ressources variées.

5.1 Présentation du protocole

5.1.1 Le protocole

Notre protocole de négociation est caractérisé par une suite de messages échangés entre l'initiateur et les participants (Figure 5.1).

1. Comme l'indique le schéma, la négociation commence toujours lors de la création d'un contrat par un agent initiateur et de sa proposition aux participants (message *propose contrat*).
2. Lorsqu'un agent participant reçoit une proposition de contrat, celui-ci l'examine et décide de l'accepter ou non (messages *accepte* et *refuse*).
3. L'agent initiateur collecte les réponses et lorsqu'elles sont toutes arrivées, ou que le temps d'attente a expiré, il prend une décision.

Trois solutions sont alors envisageables :

1. Il confirme le contrat auprès des participants (message *confirme*). Il se peut alors qu'un contrat moins prioritaire doive être annulé (message *rétractation*).
2. Il annule le contrat (message *annule*).
3. Il demande aux participants de proposer une modification pour ce contrat (message *modifier contrat*). Lorsqu'il a reçu toutes les modifications (messages *contrat modifié*), il effectue une synthèse et propose un nouveau contrat ou annule le précédent le cas échéant.

Lorsqu'un initiateur reçoit une *rétractation*, il peut soit annuler le contrat auprès de tous les participants, soit demander à le modifier.

Remarque : Le nombre de demandes de modifications est un paramètre de la négociation.

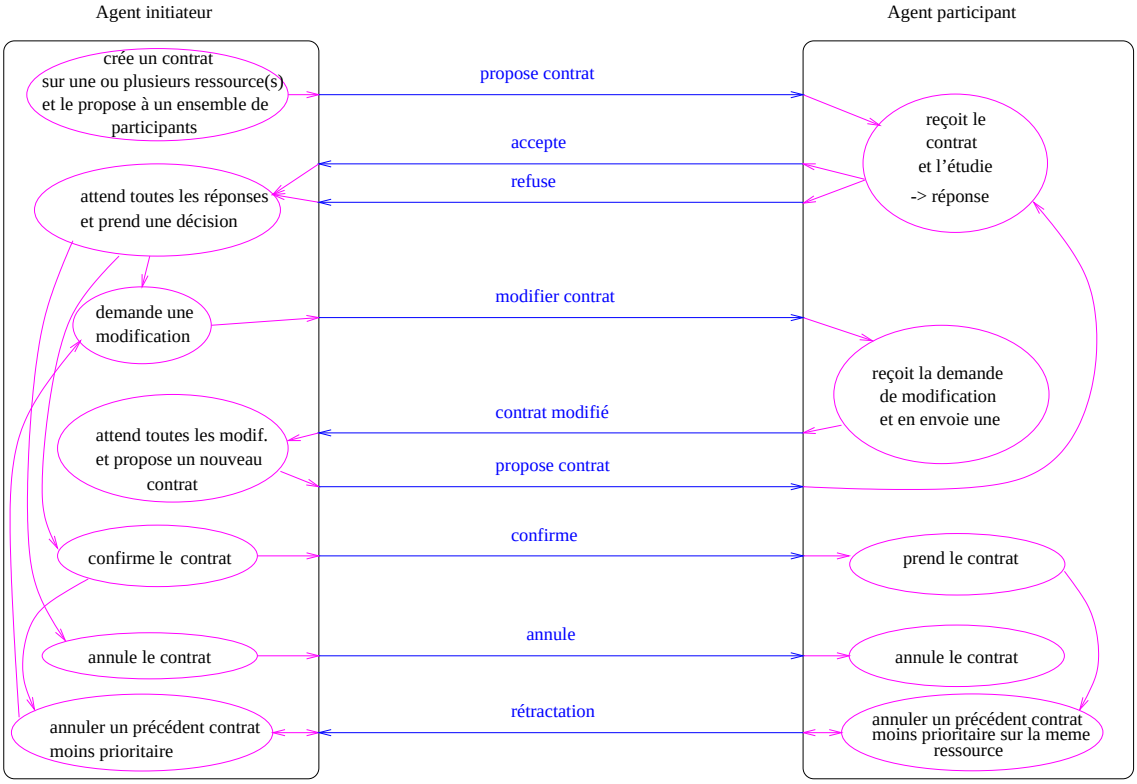

FIG. 5.1 – Schéma de notre protocole de négociation

Notre protocole utilise deux listes de priorité, l'une pour les participants, l'autre pour les ressources. Ainsi, un participant peut définir des préférences sur le choix du contrat à accepter en cas de conflit.

5.1.2 Exemples de négociations

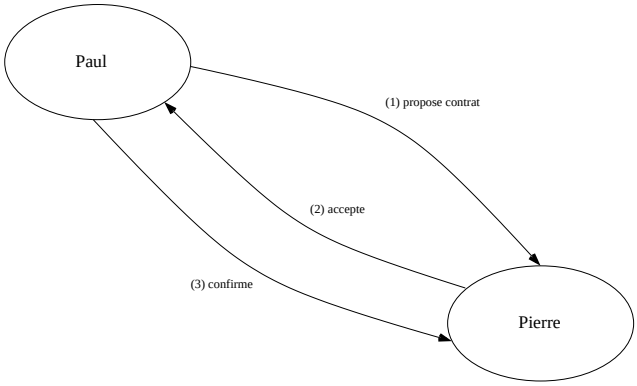

FIG. 5.2 – Exemple simple de négociation

Prenons l'exemple de la Figure 5.2 qui représente un exemple simple de négociation qui fait intervenir deux personnes : Paul et Pierre. Paul joue le rôle de l'agent initiateur et Pierre celui de l'agent participant (Figure 5.1). Paul crée le contrat et envoie à Pierre le message *propose contrat*. Pierre reçoit le contrat, l'étudie et envoie le message *accepte* à Paul pour lui signaler qu'il accepte les termes du contrat. Paul a donc reçu toutes les réponses (puisque l'ensemble des participants est constitué uniquement de Pierre) et prend la décision de confirmer le contrat. Il envoie donc à Pierre le message *confirme*.

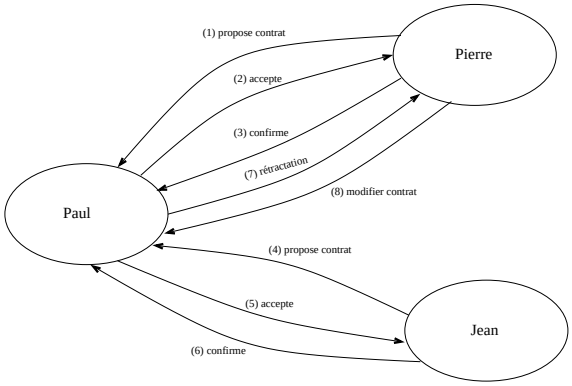

FIG. 5.3 – Exemple de négociation avec conflit et remise en question

Examinons à présent un exemple de négociation faisant intervenir 3 personnes : Paul, Pierre et Jean (Figure 5.3). Pour cet exemple, Paul considère que Jean est plus prioritaire que Pierre. Dans un premier temps, Pierre propose un contrat à Paul qui accepte (même démarche que pour l'exemple précédent). Puis Jean propose un contrat à Paul pour les mêmes ressources. Paul, considérant Jean plus prioritaire, envoie un message *accepte* à Jean. Celui-ci confirme le contrat auprès de Paul (message *confirme*). Paul prend alors le contrat avec Jean et annule le précédent contrat pris avec Pierre (message *rétractation*). Lorsque Pierre reçoit ce message, il décide de demander une modification pour ce contrat à Paul (message *modifier contrat*).

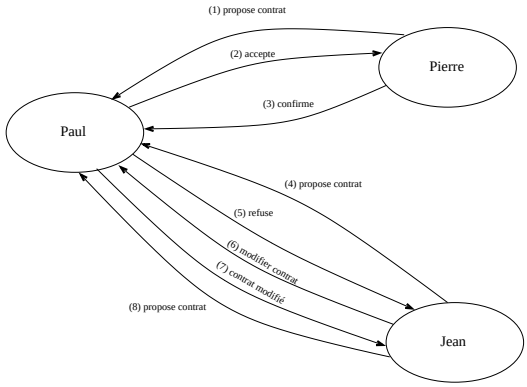

FIG. 5.4 – Exemple de négociation avec conflit

Cette fois-ci, considérons le cas où Pierre est plus prioritaire que Jean pour Paul (Figure 5.4). Pierre demande toujours en premier lieu un contrat à Paul qui accepte et donc le contrat est confirmé. Jean propose à son tour un contrat à Paul pour les mêmes ressources. Jean est moins prioritaire que Pierre, Paul refuse donc le contrat proposé par Jean (message *refuse*). Jean demande alors à Paul de proposer une modification pour ce contrat (message *modifier contrat*). Paul lui envoie à son tour une modification de contrat (message *contrat modifié*). Jean examine alors la modification proposée par Paul et lui propose un nouveau contrat (message *propose contrat*).

5.1.3 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. Notre protocole permet à un initiateur de proposer un contrat à un ensemble de participants, il s'agit donc de négociation de 1 vers n agents.

5.2 Exemples d'applications réalisées avec ce protocole

Nous avons réalisé 3 applications différentes qui utilisent ce protocole de négociation, la prise de rendez-vous, les enchères et les places de marché. Nous ne présenterons ici que la prise de rendez-vous et les enchères.

5.2.1 La prise de rendez-vous

Cette application, récurrente dans l'équipe SMAC, permet aux utilisateurs de négocier des rendez-vous. 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.

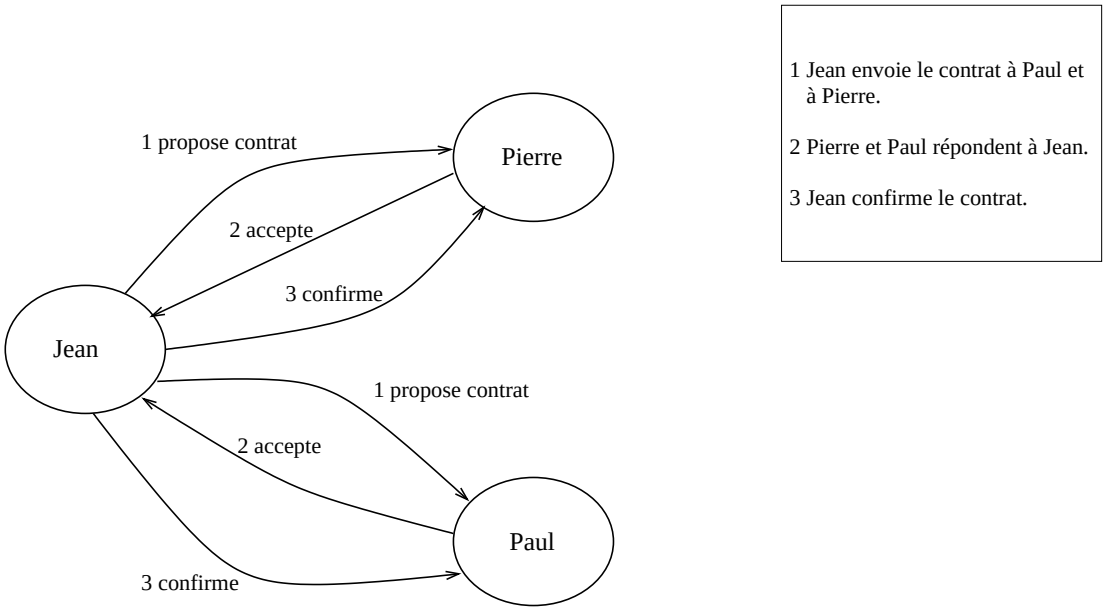

FIG. 5.5 – Exemple simple de négociation pour la prise de rendez-vous

Prenons l'exemple de 3 utilisateurs Pierre, Paul et Jean (Figure 5.5). Considérons le cas où Jean veut prendre rendez-vous avec Pierre et Paul de 15h à 16h. Jean est l'initiateur du rendez-vous (à gauche sur la Figure 5.1), il crée donc un contrat sur la ressource 15h-16h et dont l'ensemble des participants correspond à Pierre et Paul. Il envoie ensuite ce contrat aux 2 participants requis puis attend leur réponse. Chaque participant (à droite sur la Figure 5.1) reçoit le contrat et l'examine. Chacun décide alors de sa réponse, soit il accepte le contrat, soit il le refuse. Supposons que Pierre et Paul ont accepté le contrat. Jean ayant reçu 2 réponses positives, il décide de confirmer le contrat auprès d'eux. Chacun

reçoit alors un message de confirmation du contrat et peut inscrire le rendez-vous sur leur agenda.

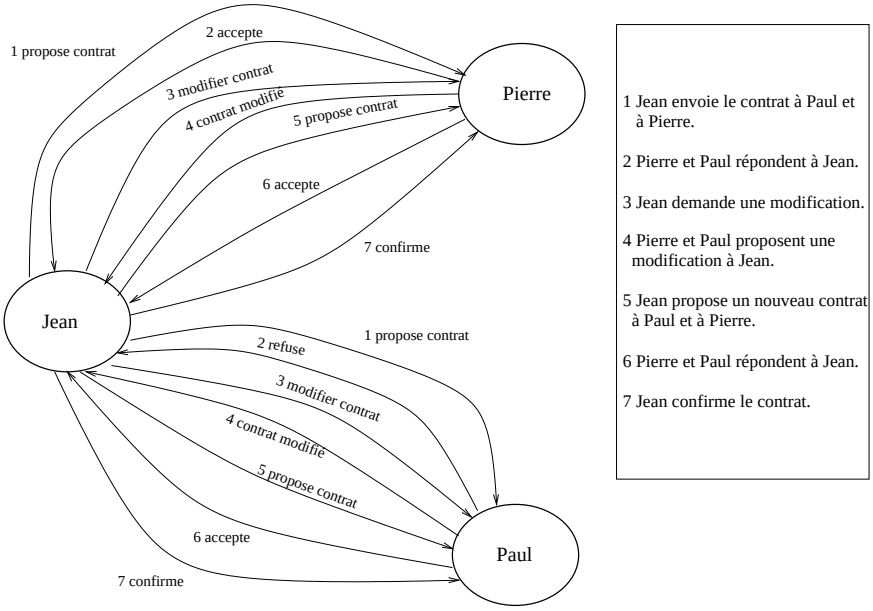

FIG. 5.6 – Exemple de négociation avec conflit et remise en question pour la prise de rendez-vous

Prenons le cas où l'un des participants, Paul par exemple, aurait refusé le contrat. Jean aurait alors reçu une réponse positive et une négative. Il aurait du décider s'il prenait le rendez-vous avec Pierre uniquement ou s'il annulait le rendez-vous. Une troisième possibilité s'offre à lui avec notre protocole : il peut demander une modification du contrat à Paul et à Pierre. Examinons ce cas précis (Figure 5.6). Jean envoie une demande de modification de contrat à Paul et à Pierre. Cette modification sera une proposition de plages horaires libres d'une durée d'une heure. Chaque participant reçoit la demande et consulte alors son agenda afin de trouver des plages horaires d'une heure encore libres et qui lui conviennent le mieux selon sa liste de priorité. Il renvoie alors à Jean sa liste de plages horaires, ordonnée de l'heure la plus prioritaire à la moins prioritaire. Jean reçoit donc 2 listes de plages horaires libres et décide alors de proposer un nouveau contrat sur une plage horaire préférée par tous en tenant compte des priorités des participants. Chaque participant reçoit la nouvelle proposition et décide si elle est acceptable ou non. Si ce n'est pas le cas, Jean pourra redemander une modification du contrat jusqu'à ce qu'un accord soit trouvé ou que le nombre de renégociations défini soit atteint. Si ce nombre est atteint, Jean annulera son contrat, de même s'il ne reste aucune plage horaire libre commune aux 3 participants.

Cet exemple montre que notre protocole permet à 3 agents de négocier entre eux sur une même ressource, et qu'il permet de faire des contre-propositions afin d'arriver à un accord commun à tous.

Il est important de noter que la prise de rendez-vous est à nos yeux l'une des applications de négociation les plus complexes à réaliser.

5.2.2 Les enchères

Le principe de la vente est le suivant :
un vendeur déclare aux participants qu'il met en vente un article. Les participants lui indiquent alors s'ils sont intéressés ou non par cet article, si oui, ils envoient le prix qu'ils sont prêts à payer pour celui-ci. Le vendeur attend toutes les réponses et mémorise le prix maximal reçu ainsi que le participant qui l'a proposé. Si ce prix dépasse le prix minimal voulu par l'acheteur, celui-ci confirme la vente auprès de ce participant et l'annule pour les autres. Dans le cas contraire, il demande aux participants intéressés de surenchérir (selon le nombre de renégociations convenu).

Ce type d'enchères correspond au modèle Français classique mais il existe d'autres modèles comme les enchères hollandaises, les enchères à la bougie etc...que nous pourrions aussi implémenter.

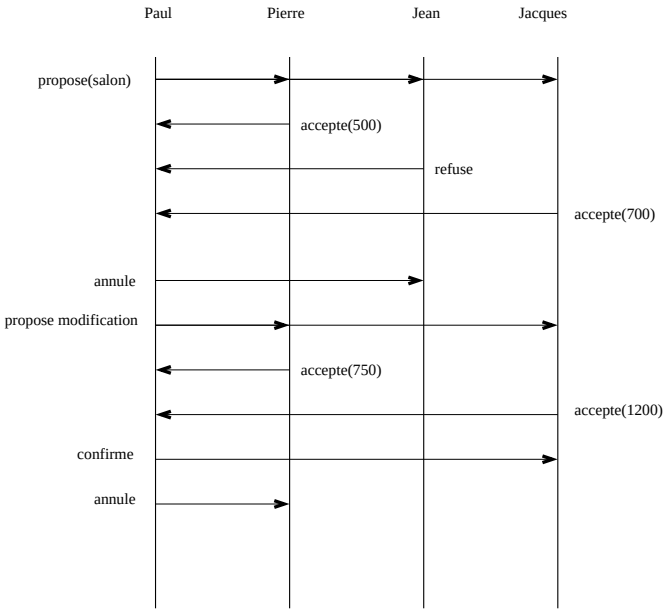

FIG. 5.7 – Exemple de vente

Prenons l'exemple d'un vendeur : Paul et de trois acheteurs potentiels : Pierre, Jean et Jacques (Figure 5.7).
Paul décide de vendre un salon et désire obtenir au moins 1000Frs pour celui-ci. Il crée donc un contrat pour son salon et envoie une proposition à Pierre, Jean et Jacques (Paul est donc un agent initiateur et Pierre, Jean et Jacques des agents participants cf. Figure 5.1).

Chacun des participants reçoit donc la proposition de Paul et décide s'il est intéressé ou non.
Pierre répond à Paul qu'il est intéressé par ce salon et qu'il lui propose de l'acheter 500Frs.
Jean répond qu'il n'est pas intéressé et Jacques lui propose 700Frs.
Paul collecte les réponses et retient que Pierre et Jacques sont intéressés mais qu'il n'ont pas proposé un prix supérieur à celui désiré et que Jean n'est pas intéressé par le salon. Il annule donc le contrat auprès de Jean et demande à Pierre et à Jacques de lui faire une autre proposition.
Pierre propose alors 750Frs et Jacques 1200Frs. Paul choisit donc de vendre son salon à Jacques pour 1200Frs, il envoie alors une confirmation pour le contrat à Jacques et annule le contrat auprès de Pierre.

Ici, la modification demandée est la proposition d'un autre prix pour le salon.

5.3 Conclusion

Nous avons présenté dans ce chapitre notre protocole de négociation. Celui-ci permet de réaliser différentes applications qui mettent en jeu des types de ressources différents et des communications entre agents de 1 vers n. Nous allons maintenant aborder la structure interne de notre modèle.

Chapitre 6

Structure interne

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 plusieurs objets communs à toute forme de négociation :

- Les ressources et leur représentation.
- Les contrats et leurs propriétés.
- Les buts.
- Les engagements.

6.1 Les ressources

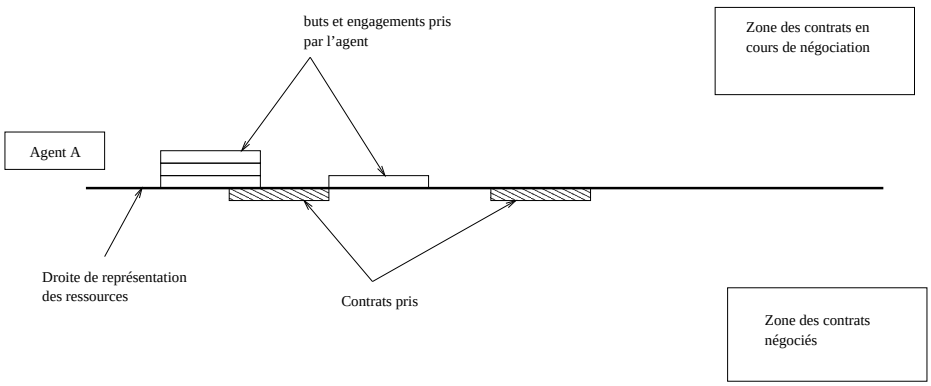

FIG. 6.1 – Représentation des ressources

Les ressources sont abstraites et peuvent aussi bien représenter des articles que des tranches horaires ou encore des cageots de fruits.

Afin de représenter les ressources, nous avons défini un modèle graphique tel que le représente la Figure 6.1. Une droite segmentée représente l'ensemble des ressources dis-

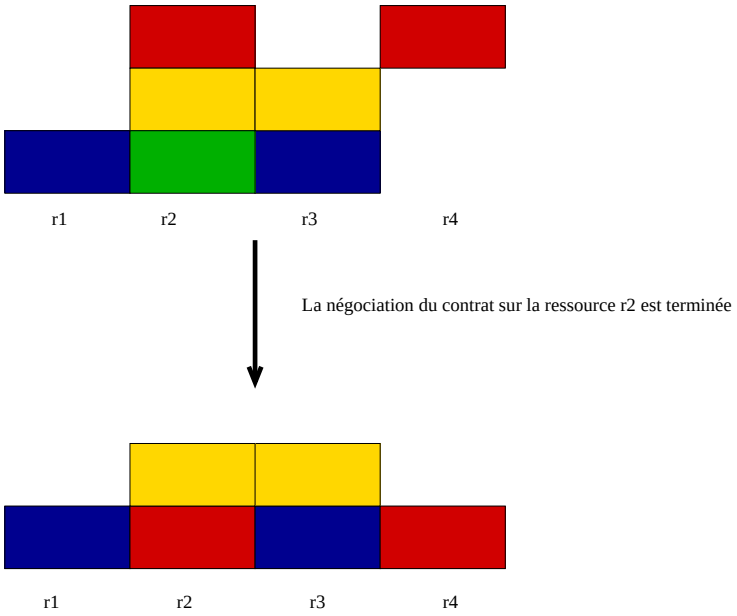

FIG. 6.2 – Exemple de progression des contrats dans la matrice d'attente.

ponibles, 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 correspondant n'étant alors plus libres. La partie supérieure contient les buts et engagements qui sont en cours de négociation sur les ressources correspondant. Il s'agit d'une matrice dont le comportement est calqué sur le jeu Tetris. Les contrats qui arrivent s'empilent en cas de conflit sur les ressources. C'est cette matrice qui indique aux objets (but ou engagement) si leur processus de négociation peut commencer (c'est le cas lorsqu'ils se situent à la surface de la matrice). Si un contrat est déjà en cours de négociation sur les ressources nécessaires pour le contrat arrivant, celui-ci est alors mis en attente. La matrice le réveillera lorsque les ressources seront libérées. Détaillons ce fonctionnement sur l'exemple de la Figure 6.2. Un premier contrat portant sur les ressources `r1` et `r3` arrive. Ces ressources n'étant pas en cours de négociation, ce contrat se retrouve à la surface de la matrice (ligne inférieure), et peut donc entamer son processus de négociation. Puis arrive un second contrat pour l'obtention de la ressource `r2`. Là encore, la ressource n'étant pas en cours de négociation, le contrat se retrouve à la surface de la matrice et entame sa négociation. Un troisième contrat est proposé pour les ressources `r2` et `r3`. Ces deux ressources sont en cours de négociation, le contrat s'empile donc au dessus, se trouvant donc sur la deuxième ligne de la matrice (il est alors mis en veille par la matrice). Un quatrième contrat arrive, celui-ci portant sur les ressources `r2` et `r4`. La ressource `r4` n'est pas en cours de négociation mais la ressource `r2` l'est. Le contrat est alors bloqué et s'empile au dessus des autres en troisième ligne de la matrice ; se retrouvant lui aussi en attente de la libération des ressources qu'il veut négocier.
Le deuxième contrat termine sa négociation, libérant ainsi la ressource `r2`. Le troisième

contrat ne peut toujours pas entamer sa négociation car la ressource `r3` est toujours occupée. En revanche, le quatrième contrat peut commencer sa négociation car les deux ressources qu'il veut obtenir ne sont pas en cours de négociation. Il se retrouve donc en surface de la matrice, et clle-ci le réveille pour qu'il commence son processus de négociation.

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. Le but ou l'engagement en cours est celui situé à la surface (base de la colonne ainsi formée), les autres étant en attente. Lors de la fin d'une négociation, le but ou l'engagement est retiré de la surface et celui qui le suivait est alors débloqué, c'est-à-dire que la négociation qu'il représente peut alors commencer.

6.2 Les contrats et leurs propriétés

Un *contrat* est un objet créé à l'initialisation de la négociation. Il comporte les *ressources* à négocier, l'*initiateur* de la négociation et la *date limite* de réponse.
Le contrat ne contient pas l'ensemble des participants de sorte à préserver leur "vie privée".
Les *propriétés* du contrat sont constituées du *contrat*, de la liste des *participants* à la négociation, du *délai* de réponse, du *nombre minimal d'accords* pour confirmer le contrat et du *nombre de renégociations autorisées*. Il mémorise également les participants qui ont accepté le contrat, le nombre d'accords reçus et le nombre de renégociations effectuées. Ce sont des objets "mémoire" de base qui contiennent les propriétés communes à tout type de négociation.

6.3 Les buts

Un *but* est un objet créé à l'initialisation d'un contrat. En fait, il représente l'automate de l'agent initiateur de la Figure 5.1.
L'agent initiateur crée le contrat puis un but auquel il passe en paramètre le contrat et les informations relatives à celui-ci. Une fois le but créé, toute la gestion du contrat est faite par celui-ci.

En premier lieu, le but envoie aux participants la proposition de contrat et lance un timer associé à une réponse par défaut, pour le cas où toutes les réponses n'arriveraient pas ; ce qui évitera les deadlocks dus à l'attente des réponses. La réponse par défaut que nous avons prise est le refus du contrat. Puis le but attend toutes les réponses.

Chaque fois qu'une réponse est reçue, le but traite la réponse (accord ou refus), il retient les participants qui ont accepté et donc ceux qui ont refusé ainsi que le nombre d'accords afin de pouvoir prendre une décision.
Le traitement des réponses consiste à mettre à jour le nombre de réponses reçues, le nombre de réponses positives et la liste des participants ayant accepté la proposition de contrat.

Dès que tous les participants ont répondu (ou que le délai d'attente a expiré), une décision est prise : le contrat est soit confirmé, soit annulé, soit une demande de modification

est envoyée.

Si le nombre de participants ayant accepté le contrat est supérieur ou égal au nombre minimal d'accord défini en même temps que le contrat, alors celui-ci est confirmé auprès des participants.

Dans le cas contraire, le but demande une modification pour le contrat à l'ensemble des participants si le nombre de renégociations effectuées n'a pas atteint le maximum défini lui aussi à la création du contrat par l'agent initiateur. Un timer est là aussi lancé, associé à une modification par défaut, là encore afin d'éviter les deadlocks. De nouveau, le but est en attente des modifications proposées par les participants.

Dès qu'une modification est reçue, le but attribue une note à chaque ressource proposée selon la formule suivante :

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

où *priorité(r_i,init/$part_j$)* est la priorité de la ressource i pour l'initiateur/le participant(si la ressource ne fait pas partie de la liste reçue,on lui attribue une priorité discriminante),
$coef_{init/part}$ est le coefficient multiplicatif de l'initiateur/du participant,
priorité($part_j$) est la priorité du participant j pour l'initiateur.
Cette note est remise à jour à chaque réception de modification : la note discriminante donnée au tour précédent du fait de la non proposition de cette ressource par ce participant est retranchée puis la note est augmentée par le calcul effectué dans la deuxième partie de la formule.
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.
Bien sûr, d'autres formules peuvent être utilisées mais nous avons choisi d'utiliser celle qui nous paraissait tenir le plus compte de l'avis de chacun et de l'importance donnée à cet avis par l'agent initiateur.
Lorsque toutes les modifications sont arrivées (ou lorsque le délai de réception a expiré), l'initiateur choisit la(les) nouvelle(s) ressource(s) qui convient(ennent) le mieux et *propose* le nouveau contrat.

Si le nombre de renégociations a atteint le maximum, le contrat est alors annulé auprès de tous les participants.

Lorsqu'un initiateur reçoit une *rétractation*, il regarde si le nombre minimal d'accords est encore atteint, et demande une modification du contrat aux participants si ce n'est pas le cas.

6.4 Les engagements

Lorsqu'un agent reçoit une proposition de contrat, il crée un *engagement* et lui passe le contrat en paramètre. C'est alors celui-ci qui s'occupe de la négociation du contrat, c'est-

à-dire qu'il représente l'automate de l'agent participant de la Figure 5.1.

L'engagement étudie le contrat et décide de l'accepter ou de le refuser. Il envoie cette réponse à l'initiateur du contrat. Pour décider si le contrat sera accepté ou refusé, l'engagement vérifie si un contrat est déjà pris pour cette ressource. Si c'est le cas, il regarde si l'initiateur du contrat proposé est plus prioritaire que l'initiateur du contrat déjà pris. Si c'est le cas, il accepte le contrat, sinon il le refuse. Si aucun contrat n'a été pris pour cette (ces) ressource(s), alors il accepte la proposition.

Si le contrat est confirmé, alors l'engagement l'ajoute aux contrats pris par l'agent qu'il représente et envoie, si nécessaire, un message de *rétractation* à l'initiateur du (des) contrat(s) moins prioritaire(s) nécessitant les mêmes ressources. S'il est annulé, il le retire de la liste des contrats en cours de négociation.

Si une demande de modification lui parvient, il classe les ressources qui n'ont pas encore été prises par un contrat par ordre de priorité et propose celle(s) qui est (sont) le plus prioritaire(s). Le nombre de ressources à proposer est un paramètre de la négociation.

6.5 Gestion simultanée

Lorsque l'on veut négocier, il se pose vite le problème des négociations simultanées. En effet, pourquoi traiter toutes les négociations séquentiellement alors que certaines peuvent être traitées simultanément ? C'est le cas lorsque les contrats portent sur des ressources différentes. On peut alors exécuter leur processus de négociation simultanément.

Le fait d'avoir des buts et des engagements qui gèrent la négociation d'un contrat permet d'effectuer plusieurs négociations simultanément, et d'être à la fois initiateur et participant à des contrats. En effet, l'agent est toujours libre pour lancer une négociation, puisqu'il la confie à un but s'il est l'initiateur, ou à un engagement s'il est participant. Afin de savoir si la négociation peut avoir lieu, le but ou l'engagement créé est placé dans la matrice de gestion des ressources (cf. section 6.1). Si le contrat géré par le but ou l'engagement en question ne porte pas sur des ressources en cours de négociation, alors il peut être négocié , sinon il est mis en attente jusqu'à la libération des ressources qui sont en cours de négociation.

Nous avons présenté dans ce chapitre les objets fondamentaux de mise en place de notre protocole. Nous allons maintenant présenter l'implémentation que nous avons réalisé.

Chapitre 7

Implémentation

Nous avons choisi, pour des raisons pratiques, d'implémenter notre protocole avec la plate-forme Magique, cependant, notre approche étant polyvalente, d'autres plate-formes telles MadKit ou Jatlite auraient pu convenir. A terme, on se passera de plate-forme multi-agents et la communication pourra se faire via l'envoi d'e-mail ou grâce à une architecture client/serveur ou bien encore par connexion à un serveur web.
L'agent Magique ne possède au départ aucune compétence spécifique, il faut donc définir et implémenter toutes les compétences nécessaires à la réalisation de l'application.

7.1 Les objets du protocole

Nous avons vu dans le chapitre précédant les différents objets nécessaires pour notre protocole. Nous avons donc défini les classes *contrat* et *contratEtPropriétés*, ces classes ne sont pas abstraites mais peuvent être étendues s'il est nécessaire de mémoriser des informations supplémentaires spécifiques au type de négociation réalisé.

Les *but* et *engagement* sont des classes qui gèrent le protocole de communication défini. Les fonctions de décision par défaut, présentées dans le chapitre précédent, peuvent ne pas convenir pour le type de négociation en cours. On peut alors étendre ces classes afin d'améliorer le comportement de l'agent lors de la réception d'une réponse ou d'une modification ou lorsqu'elles ont toutes été reçues.

Il y a enfin la représentation des ressources qui nécessite 3 objets : la liste des ressources disponibles pour la négociation (l'ensemble des ressources qui seront négociées), la liste des contrats pris sur ces ressources (1 contrat par ressource ou groupe de ressources) et enfin une matrice pour stocker l'ensemble des buts et engagements qui sont en cours de négociation sur les ressources (ce qui représente la partie supérieure de la droite de la Figure 6.1).

7.2 La compétence de négociation

Cette compétence est la classe principale de notre application. En effet, c'est elle qui permet de gérer l'ensemble des négociations en cours.
Si l'agent est l'initiateur d'un contrat à négocier, il crée ce contrat ainsi qu'un but qui gérera toute la négociation concernant ce contrat. Si l'agent reçoit une proposition de contrat, il est alors participant et crée un engagement qui gérera toute la négociation sur ce contrat.
Par la suite, l'agent transmettra les messages de négociation qui lui arrivent à qui de droit (but ou engagement propre au contrat en question), s'ils sont encore valides (afin d'ignorer les messages reçus en double ou reçus après le délai d'attente).
C'est aussi cette compétence qui permet de mémoriser les contrats négociés, ceux en cours de négociation et les listes de priorité pour les ressources et les participants.

Remarque Cette classe devra être étendue si les but et engagement l'ont été (car c'est elle qui les crée), ou s'il est nécessaire d'implémenter des fonctions spécifiques au type de négociation exercé.

Le Tableau 7.1 récapitule les différentes classes et les principales méthodes implémentées.

7.3 L'agent superviseur

Cet agent a été spécifiquement créé à titre de test. Il représente le *facteur* qui distribue les messages aux agents. Son rôle est de mémoriser les "adresses" des différents participants et de gérer la distribution du "courrier". Si un agent est déconnecté du système, ses messages seront en "poste restante" jusqu'à ce qu'il se reconnecte.
Dans la réalité, un agent peut envoyer ses messages par exemple par e-mail à des accointances qu'il connaît et gère lui-même.

Le second rôle de cet agent est de mémoriser les ressources, et de les communiquer à tout nouvel agent arrivant dans le système de négociation.

7.3.1 La compétence de serveur de noms

Cette compétence permet de mémoriser les différents participants aux négociations. Chaque agent voulant négocier s'inscrit au préalable à l'annuaire auprès du superviseur (Figure 7.1). Lors de l'inscription, l'agent précise son identifiant, son adresse, le type de négociation qu'il va effectuer ainsi que ses ressources à négocier (dans le cas de négociation sur des ressources individuelles).
En retour, l'agent reçoit la liste des participants à ce type de négociation ainsi que les ressources disponibles pour ce type.
Le superviseur ajoute l'agent dans l'annuaire général (qui peut être vu comme les pages blanches) et dans l'annuaire spécifique du type de négociation exercée (pages jaunes).
Le serveur de noms joue le rôle du facteur pour les différents messages échangés. Lorsqu'un

Classe	Attributs	Méthodes
Ressource	nom	toString()
Contrat	initiateur ressources date limite	getInitiator() getRessources() getLimitDate()
Contrat et propriétés	contrat participants participantsAgree délai nb min accords nb accords nb max renegociation nb renegociation	
But	contrat et propriétés nb réponses reçues timer	contact(...) accordReçu(...) décision() réponseReçue(...) demandeModification(...) ModificationReçue(...) jugeModification(...) décideModification() envoyerConfirmation(...) envoyerAnnulation(...)
Engagement	contrat	déterminerRéponse() envoyerAccord(...) envoyerRefus() déciderModification() envoyerModification(...) confirmer() annuler()
Compétence de négociation	matrice de gestion des ressources contrats pris contrats en cours de négociation contrats initiés liste des buts et engagements liste des ressources liste de priorité ressources liste des participants liste de priorité participants	créerContrat(...) messageReçu(...) réceptionParticipantsEtRessources(...) prendreContrat(...) fixerPrioritéParticipants(...) fixerPrioritéRessources(...)

TAB. 7.1 – Tableau récapitulatif des différentes classes implémentées.

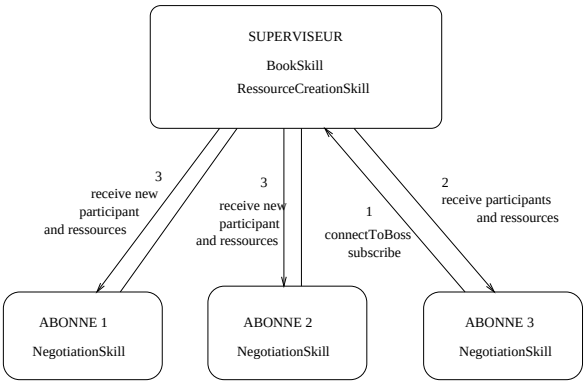

FIG. 7.1 – Arrivée d'un nouvel abonné

message arrive pour un participant, il regarde s'il possède son adresse actuelle (ie. si le participant est connecté) et dans ce cas il lui transmet directement le message ; dans le cas contraire, le message est stocké dans la boîte aux lettres du participant qui le recevra lors de sa prochaine connexion.

7.3.2 La compétence de gestion des ressources

Cette compétence permet de mémoriser les ressources par type. Par exemple, le type *Rdv* correspondra aux ressources représentant les tranches horaires possibles pour un rendez-vous ; le type *vente* regroupera les différents articles mis en vente.
Remarque : les ressources d'un même type peuvent appartenir à plusieurs participants. Typiquement, les ressources de type *Rdv* seront communes à tous les participants, elles seront donc créées une fois pour toutes par le superviseur et il les transmettra à chaque participant lors de son inscription. En ce qui concerne la vente, chaque vendeur transmettra ses articles lors de son inscription et ceux-ci seront stockés dans un seul et même type *vente*, et chaque participant à ce type de négociation recevra les différents articles à pourvoir.

Le superviseur fait donc le lien entre les différents agents, c'est grâce à lui que tous peuvent communiquer en permanence. Il est donc indispensable qu'il soit persistant puisqu'il est la *mémoire* de tout le système.

7.4 L'interface utilisateur

Nous avons implémenté une interface utilisateur de base qui permet de se connecter au système, de trier les participants et les ressources par ordre de priorité et de créer des contrats. Elle permet également de visualiser les messages reçus et les contrats pris par l'agent. On peut aussi décider de gérer manuellement les contrats proposés, c'est-à-dire de répondre soi-même si on les accepte ou non.

Ceci présente l'avantage d'utiliser plus rapidement et simplement l'API, et de manière plus conviviale. L'interface permet également de pouvoir revoir les messages qui ont été échangés auparavant. Cela représente un côté pratique pour la gestion des négociations par un agent humain. Il peut ainsi se remémorer le déroulement des précédentes négociations et choisir de modifier ses préférences en fonction des expériences antérieures.

Pour fournir cette interface, nous avons défini 2 classes spécifiques : la classe *graphe* et la classe *négociateurGraphe*. La première sert à construire l'interface, la seconde est une compétence qui permet d'interagir avec la compétence de négociation.
Ces deux classes sont abstraites, elles contiennent toutes les parties communes aux différents types de négociation, tels que les fenêtres de visualisation de l'ensemble des participants ou de l'ensemble des ressources, les contrats pris, les messages reçus. Il n'est nécessaire que d'implémenter 2 fonctions spécifiques pour la création de l'interface : celle qui permet d'obtenir les informations nécessaires pour la création du contrat du type de négociation effectué et celle pour l'affichage des propositions de contrat pour le mode manuel et la récupération de la réponse donnée par l'utilisateur. Si l'utilisateur accepte un contrat sur une ressource déjà prise, il en est averti et doit confirmer sa réponse afin de tenir compte des priorités sur les participants.
Pour l'interaction entre l'interface et la compétence de négociation, il faut implémenter la fonction qui récupère les paramètres spécifiques lors de la création d'un contrat, ainsi que la fonction qui affiche la proposition de contrat pour le mode manuel et celle qui envoie l'acceptation du contrat.

7.5 Points forts de l'application

Plusieurs points forts peuvent être dégagés pour notre application :
- Plusieurs négociations peuvent avoir lieu simultanément.
- Fonctionnement en réseau hétérogène.
- Mélange d'agents virtuels et humains.
- Communication asynchrone.
- Renégociation automatique.
- Listes de priorité pour les participants et les ressources.
- Le même langage est utilisé par tous.
- Prise en compte des deadlocks pouvant survenir.

Chapitre 8

Exemples d'applications de notre modèle

Notre modèle se voulant générique, nous avons étudié trois applications de type différent :
- la prise de rendez-vous,
- la salle de vente,
- un sous-ensemble de Diplomacy.

8.1 La prise de rendez-vous

Le modèle de la prise de rendez-vous avait fait l'objet du stage de DEA d'Alain Taquet en 1996. Il avait réalisé l'application ANTS avec une structure à base de fourmis.
Cette application a été présentée en sous section 5.2.1. Nous allons examiner le comportement d'un agent réalisant de la négociation pour la prise de rendez-vous.

8.1.1 Du point de vue de l'initiateur : le but

La première action que fait le but est d'envoyer la proposition de rendez-vous aux participants. Il lance simultanément un timer associé à une réponse par défaut (ici, un refus) afin de ne pas attendre éternellement les réponses des différents participants.

Lorsqu'une réponse positive a été reçue, on incrémente le nombre d'accords reçus. Lorsque toutes les réponses ont été reçues, il décide de *confirmer* le rendez-vous si le seuil minimal de réponses positives est atteint. S'il n'est pas atteint et que le nombre de renégociations autorisées ne l'est pas non plus, il envoie une *demande de modification* pour ce contrat. Dans le cas contraire, il *annule* le contrat auprès des participants.

Lors de la réception d'une *modification de contrat*, l'initiateur attribue une note aux tranches horaires proposées selon la priorité du participant qui les propose et leur priorité pour ce participant.

Chaque tranche horaire est notée de la façon suivante :

$$note(h_i) = priorité(h_i, init) * coef_{init} + \sum_{j=1}^{n} priorité(h_i, part_j) * priorité(part_j) * coef_{part}$$

où *priorité(h_i,init/$part_j$)* est la priorité de la tranche horaire i pour l'initiateur/le participant (si elle ne fait pas partie de la liste reçue,on lui attribue une priorité discriminante),
$coef_{init/part}$ est le coefficient multiplicatif de l'initiateur/du participant,
priorité($part_j$) est la priorité du participant j pour l'initiateur.
Cette note est remise à jour à chaque réception de modification : la note discriminante donnée au tour précédent du fait de la non proposition de cette tranche horaire par ce participant est retranchée puis la note est augmentée par le calcul effectué dans la deuxième partie de la formule.
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 tranche horaire proposée. De plus, les tranches horaires non proposées sont notées de façon discriminatoire afin qu'elles ne soient pas proposées dans le nouveau contrat.
Bien sûr, d'autres formules peuvent être utilisées mais nous avons choisi d'utiliser celle qui nous paraissait tenir le plus compte de l'avis de chacun et de l'importance donnée à cet avis par l'agent initiateur.
Lorsque toutes les modifications sont arrivées (ou lorsque le délai de réception a expiré), l'initiateur choisit la(les) nouvelle(s) tranche(s) horaire(s) qui convient(ennent) le mieux et *propose* le nouveau 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.

8.1.2 Du point de vue du participant : l'engagement

Lorsqu'un participant reçoit une *proposition* de rendez-vous, il commence par vérifier si la tranche horaire est encore libre. Si tel est le cas, il *accepte* le rendez-vous. Sinon, il regarde si l'initiateur du rendez-vous est plus prioritaire que celui du rendez-vous déjà pris. Si oui, il *accepte* le rendez-vous, sinon il 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 participant consulte son agenda et envoie un nombre prédéfini de tranches horaires libres et par ordre de priorité à l'initiateur du rendez-vous.

Ces comportements sont exactement ceux proposés par défaut. Il ne faut donc pas étendre de classe pour cet exemple d'application.

8.1.3 Illustrations

La surface représentant l'agenda

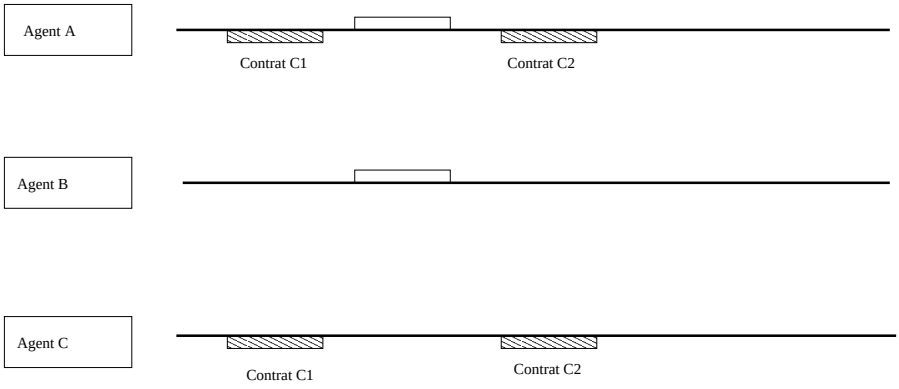

FIG. 8.1 – Etat initial de la négociation

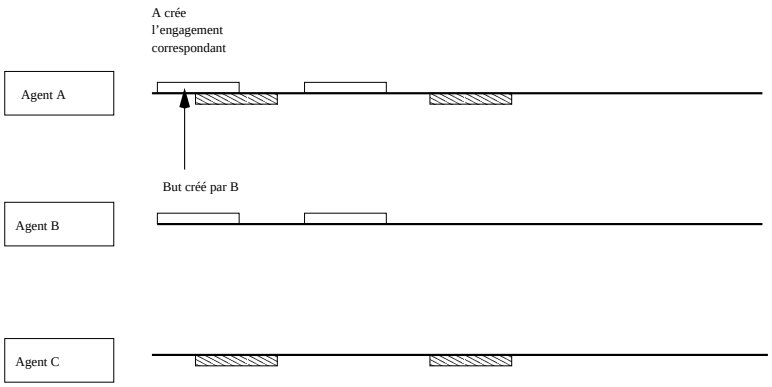

FIG. 8.2 – B demande un rendez-vous à A

Trois agents A, B et C sont présents dans la négociation (Figure 8.1). L'agent B demande un rendez-vous avec l'agent A. A crée alors un engagement pour ce contrat (Figure 8.2). Pour l'agent A, B est plus prioritaire que C, donc A envoie un message *accepte* à B. Si celui-ci confirme, A enverra à C un message *rétractation* (Figure 8.3).

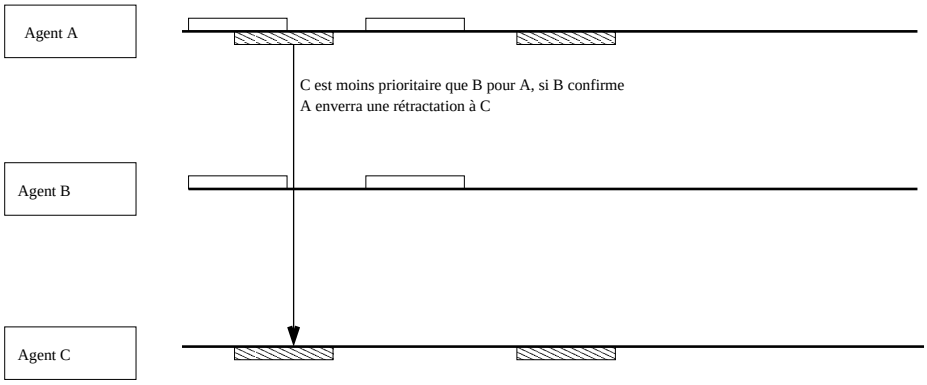

FIG. 8.3 – Si B confirme, A se rétracte auprès de C

L'interface utilisateur proposée

La Figure 8.4 représente les interfaces de 4 agents réalisant de la négociation pour la prise de rendez-vous avec notre API.
L'écran en haut à gauche est la fenêtre de visualisation des messages concernant l'agent. Elle permet de visualiser les différentes propositions reçues et l'avancement de la négociation (réponse envoyée, confirmation ou annulation, demande de modification,...). L'écran en haut à droite représente l'interface de saisie d'un nouveau contrat, celui en bas à gauche affiche les contrats pris avec le nom de l'initiateur et les ressources négociées. Le dernier écran représente l'affichage de la proposition de contrat pour le mode manuel.

8.2 La vente aux enchères

Le principe de la vente aux enchères a été décrit en sous section 5.2.2. Les ressources sont ici des articles, seul leur nom descriptif suffit pour créer les ressources. En revanche, il faut ajouter aux propriétés du contrat de base le prix minimum que le vendeur veut obtenir. Il faut donc créer une classe *PropriétésContratEnchères* qui étend la classe *PropriétésContrat*.
Pour cette application, il a fallu implémenter une compétence de négociation spécifique aux enchères : la classe *CompétenceNégociationEnchères* qui étend la classe *CompétenceNégociation*. En effet, il faut mémoriser les articles possédés par l'agent, l'ensemble des articles qui seront proposés et qui l'intéressent, la somme d'argent qu'il possède et les enchères qu'il a proposé.

Pour cette application, il a fallu définir des fonctions spécifiques dans la compétence de négociation. Tout d'abord, une méthode qui permette de choisir les articles qui intéressent l'agent, puis une fonction qui crédite (ou débite) la somme d'argent qu'il possède. De plus, une fonction pour déterminer le prix à proposer pour un article est nécesaire.

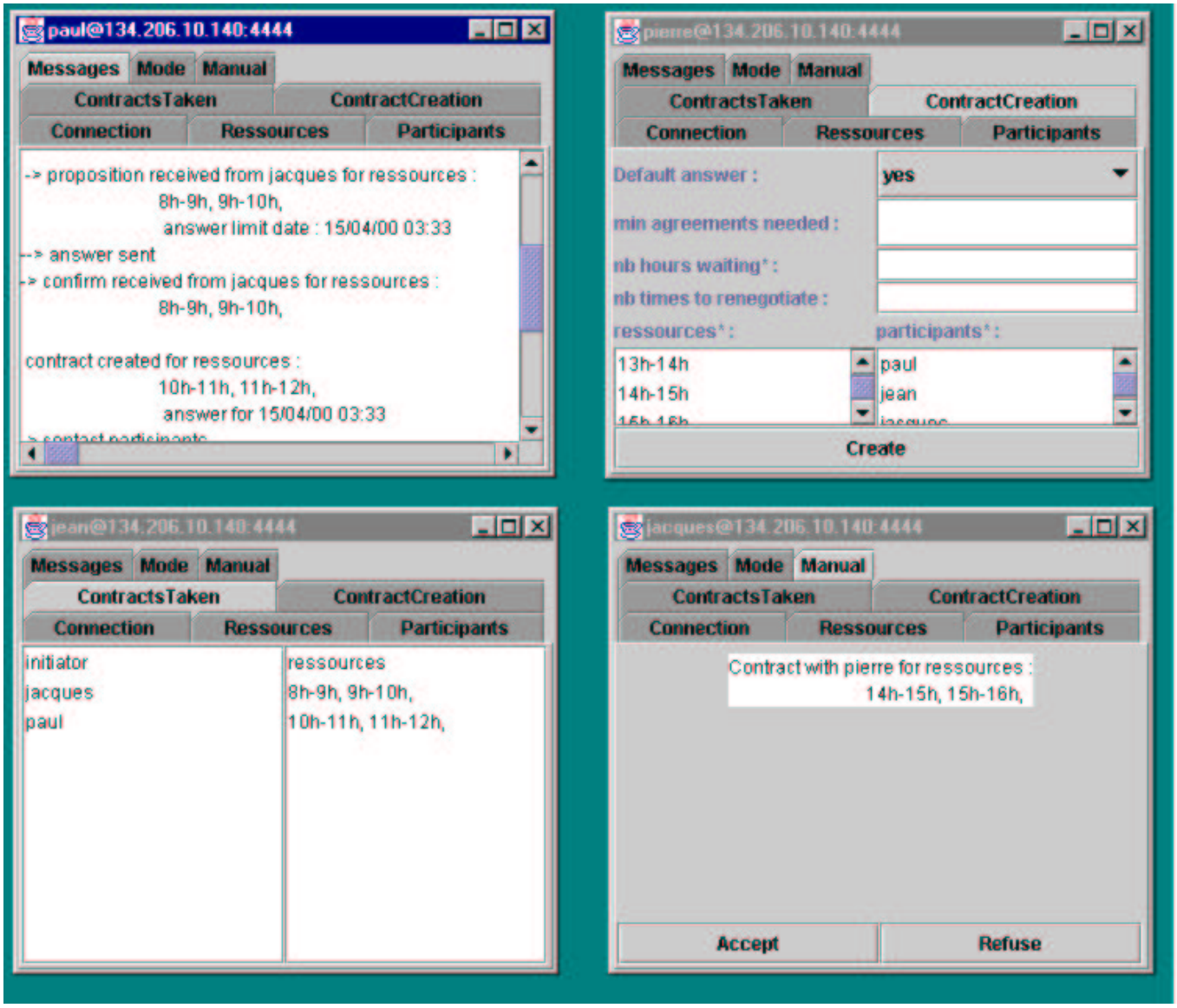

FIG. 8.4 – Exemple d'implémentation de la prise de rendez-vous avec notre API

Examinons maintenant le comportement de l'initiateur puis celui du participant.

8.2.1 Le but

Le comportement par défaut n'est pas tout à fait approprié aux enchères. En effet, pour cette application, le contrat sera confirmé pour un participant (celui qui a remporté l'enchère) et annulé pour tous les autres. De plus, il faut ici examiner le prix de la resource et non la ressource elle même. Ce ne sera donc pas la ressources qui sera notée, mais le prix proposé.
Lors de la réception d'une réponse positive, ou d'une modification, l'initiateur regarde si le prix proposé est supérieur au prix maximum proposé jusqu'alors. Si tel est le cas, il retient ce prix et le participant l'ayant proposé.

Lorsque toutes les réponses (ou modifications) ont été reçues, ou lorsque le délai d'attente a expiré, si le prix maximum proposé est supérieur ou égal au prix minimum désiré par l'initiateur, alors le participant qui l'a proposé remporte le contrat (ie. l'initiateur confirme le contrat auprès de celui-ci), et le contrat est annulé pour les autres participants. Si le prix minimum n'est pas atteint et que le nombre de renégociations ne l'est pas non plus, l'initiateur annule le contrat auprès des participants l'ayant refusé et envoie une demande de modification aux participants l'ayant accepté. Dans les autres cas, le contrat est annulé pour tous les participants.

Avant d'envoyer une demande de modification, le prix maximum proposé est remis à 0.

8.2.2 L'engagement

Lors de la réception d'une proposition de contrat, l'engagement commence par regarder si l'objet proposé l'intéresse ou non. Si cet objet l'intéresse, il estime son prix et le propose à l'initiateur. Il retient ce prix afin d'en proposer un supérieur en cas de demande de modification. S'il ne reste pas assez d'argent pour surenchérir alors il propose une somme nulle, ce qui sera considéré comme un refus par l'initiateur.

Il faut donc étendre les but et engagements afin de surcharger les méthodes de prise de décision et d'ajouter les éléments nécessaires à celle-ci. Nous avons donc implémenté la classe *ButEnchères* qui étend la classe *But* et surcharge les méthodes où sont prises les décisions, et la classe *EngagementEnchères* qui étend la classe *Engagement* et surcharge elle aussi les méthodes de prise de décision.

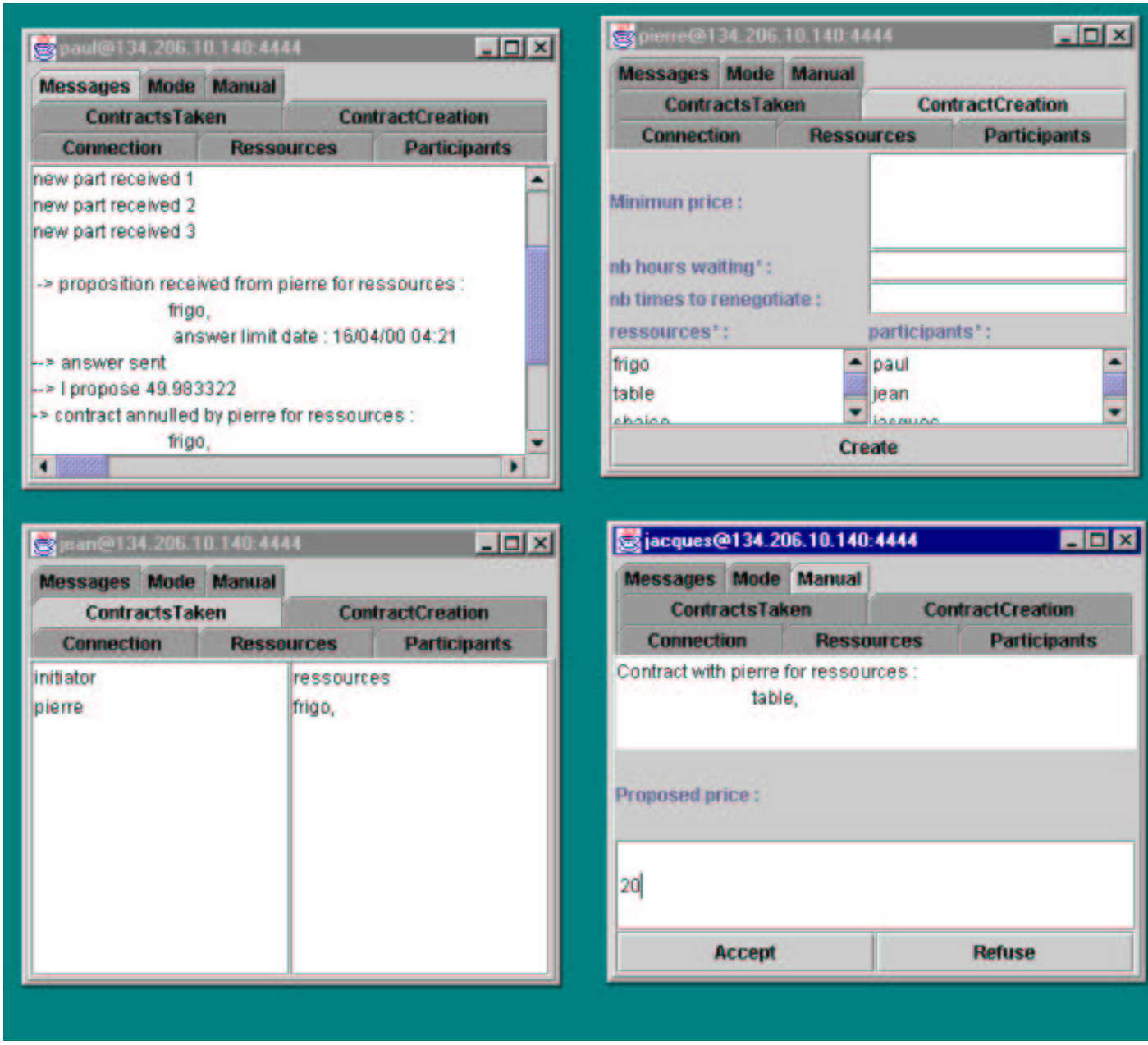

FIG. 8.5 – Exemple d'implémentation de la vente aux enchères avec notre API

8.2.3 Exemple

La Figure 8.5 représente l'interface graphique de 4 utilisateurs réalisant des enchères. Dans la fenêtre des messages, on voit la proposition d'un vendeur puis la réponse que l'on a envoyé, ici c'est le prix proposé pour l'article en question. Dans la fenêtre de création de contrat, on remarque qu'un prix minimum pour la vente de l'article est demandé. La fenêtre de réponse à une proposition de contrat pour le mode manuel possède un champ "prix proposé" afin de pouvoir enchérir..

8.3 Conclusion

Ces exemples que nous avons implémentés illustrent la réalisation d'une application de négociation. Mais une analyse fine du problème à résoudre est un passage obligatoire pour mener à bien une telle réalisation. Nous allons présenter ce cheminement dans le chapitre suivant.

Chapitre 9

De l'analyse à la conception

Afin de réaliser une application de négociation, il faut tout d'abord définir les objets de la négociation (par exemple des rendez-vous ou des articles). Il faut ensuite définir quelles sont les propriétés de ces objets, et quels sont les paramètres à négocier (tranches horaires, prix). Il faut ensuite définir la stratégie de négociation à mettre en place du point de vue de l'initiateur et du point de vue du participant.

Nous allons illustrer ce cheminement avec l'exemple d'un sous ensemble du jeu Diplomacy. Ce jeu est communément reconnu comme nécessitant une phase de négociation.

9.1 Analyse du problème

9.1.1 Présentation du problème

Diplomacy est un jeu qui met en scène plusieurs personnes dont le but est de maximiser la valeur des ressources qu'elles possèdent. Ce jeu fonctionne de façon asynchrone, chaque joueur parle quand il le souhaite.
Il y a 6 ressources différentes qui valent dans l'ordre et par convention de 1 à 6 points :

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 $ressource_i$ et *nbCartes(r_i)* le nombre de cartes représentant la $ressource_i$ que le joueur possède. La valeur de la main d'un joueur, notée *valeur(main)*, est calculée par la formule suivante :

$$valeur(main) = \sum_{j=1}^{6} valeur(r_i) * nbCartes(r_i)^2 \tag{9.1}$$

Cette formule 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 $2 * 3^2 = 18$ alors que la seconde vaut $2 * 1^2 + 1 * 1^2 + 3 * 1^2 = 6$.

Pour parvenir à obtenir une main de valeur maximale, les joueurs s'échangent des cartes, la seule obligation est d'en échanger 2. Ce qui 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 d'une ressource nommée contre une autre ressource nommée, par exemple du bois contre de la pierre,
2. l'échange d'une ressource nommée contre une autre ressource non nommée, par exemple du bois contre n'importe quoi,
3. l'échange d'une ressource non nommée contre une autre ressource nommée, par exemple n'importe quoi contre du bois.

9.1.2 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. 9.1 – Etat initial du jeu.

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. 9.2 – Etat du jeu après le premier échange.

Prenons l'exemple de 3 joueurs, J_1, J_2 et J_3. Leur jeu initial est donné en Figure 9.1. Supposons que le premier joueur décide de se débarrasser de sa pierre et de son blé. Le

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. 9.3 – Etat du jeu après le second échange.

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 9.2.
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 9.3).

9.1.3 Analyse du problème

Après une première lecture du problème, il apparaît clairement que le contrat de base 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).
L'initiateur devra en outre retenir les participants ayant répondu à sa demande et si l'une des ressources n'était pas nommée, ce que l'autre joueur lui a proposé.
Tous les joueurs devront pouvoir évaluer leur main, et estimer si la carte proposée est intéressante pour son jeu, ainsi que décider quelle est la ressource qu'il va céder. Afin de savoir quelle est la ressource qui m'intéresse plus, je peux utiliser la liste de priorité sur les ressources pour les classer.

9.2 Implémentation

Nous allons maintenant passer en revue les différentes classes à implémenter pour réaliser une application qui réalisera la négociation d'échanges de cartes. Nous supposerons ici que la décision de créer un contrat (ie. proposer un échange) se fera par un agent humain.

9.2.1 Le contrat et ses propriétés

Tout d'abord il faut écrire une nouvelle classe *ContratDiplomacy* qui étend la classe *Contrat* et qui possède en plus un attribut *échangéContre* qui contiendra la ressource

désirée par l'initiateur en échange de celle qu'il propose (attribut *ressource* de la classe *Contrat*). Ceci constitue le contrat tel qu'il sera envoyé aux participants.

Nous allons maintenant nous intéresser aux propriétés relatives au contrat que l'agent doit retenir en plus de celles déjà contenues dans la classe *ContratEtPropriétés*, à savoir le contrat, les participants, les participants ayant accepté le contrat, le délai d'attente des réponses, le nombre maximal de renégociations et le nombre de renégociations effectuées. Il faudra donc écrire une classe *ContratEtPropriétésDiplomacy* qui étendra la classe *ContratEtPropriétés*, et qui contiendra en plus un attribut *proposé* qui retiendra ce que chaque participant ayant accepté le contrat propose en échange de la carte désirée ou proposée par l'initiateur. Ceci permettra à celui-ci d'effectuer un choix sur le participant qui remportera le contrat.

9.2.2 La compétence de négociation

Il faudra créer une classe *CompétenceNégociationDiplomacy*, qui étendra la classe de base *CompétenceNégociation*. En effet, il faudra ajouter une méthode 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.

9.2.3 Le but

Là encore, la classe *But* sera étendue par une classe *ButDiplomacy* qui implémentera les méthodes de décision à prendre au vu des réponses ou modifications obtenues.
En effet, le but évaluera la main qu'il possèderait après l'échange afin de décider si il va accepter ou non le contrat. Il s'agit donc de définir la stratégie de négociation du joueur en tant qu'initiateur. Plusieurs solutions sont donc possibles et nous avons choisi de présenter ici une stratégie simple. Celle-ci se décompose en trois cas correspondants aux trois types de contrats définis dans la présentation du problème.

Echange de 2 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.

Echange d'une ressource proposée contre n'importe quoi

Dans ce cas, il s'agit de céder la carte proposée et d'en obtenir une plus intéressante en échange.
Si personne n'est intéressé par cette ressource, le contrat est annulé. Si plusieurs joueurs m'ont proposé une ressource en échange, je confirme le contrat auprès de celui qui m'a proposé celle qui m'intéresse le plus (celle qui maximise ma main évaluée par la formule 9.1 et ne la diminue pas trop), et j'annule alors le contrat auprès des autres. Si aucune ressource proposée ne m'intéresse, je demande alors à ces participants de m'en proposer une autre (demande de modification).
Le principe est le même lorsque toutes les modifications ont été reçues.

Echange d'une ressource désirée contre n'importe quoi

Il s'agit ici d'obtenir la carte désirée et d'en fournir une autre en échange.
Si personne ne veut me céder cette ressource, le contrat est annulé.
Si plusieurs joueurs veulent bien me l'échanger, j'examine ce qu'ils me demandent en échange en évaluant par la formule 9.1 la main que j'aurais si je réalisais l'échange. Si l'un d'eux désire une carte que j'ai et qui ne m'intéresse pas, alors il remporte le contrat. Si je ne veux céder aucune des cartes demandées ou si je n'ai aucune de ces cartes, alors je demande à ces participants de modifier leur demande.
Ce principe est conservé lors de la réception des modifications.

9.2.4 L'engagement

Là encore, il faut écrire la classe *EngagementDiplomacy* qui implémentera les méthodes des réponses et modifications à envoyer à l'initiateur du contrat reçu en proposition. Elle évaluera elle aussi les mains qu'elle aurait si elle réalisait l'échange. Elle retiendra également les ressources que j'ai proposé afin de ne pas reproposer les mêmes en cas de demande de modification.
C'est donc la stratégie du joueur en tant que participant à une négociation qu'il faut implémenter. Ici encore, nous allons décrire une stratégie simple qui se décompose elle aussi selon les trois types d'échanges pouvant être réalisés.

Echange de 2 ressources nommées

Si je ne possède pas la ressource désirée par l'initiateur ou si celle qu'il me propose en échange ne m'intéresse pas (si elle n'augmente pas la valeur de ma main), je refuse le contrat. Sinon, je l'accepte.

Echange d'une ressource proposée contre n'importe quoi

Si la ressource proposée m'intéresse (j'en ai déjà), j'accepte le contrat et je propose en échange la ressource que je possède qui ne m'intéresse pas. Ce peut être par exemple la

ressource que je possède en unique exemplaire et/ou celle qui vaut le moins de points. Si la ressource proposée ne m'intéresse pas, je refuse le contrat.
En cas de demande de modification, je propose une autre ressource qui ne m'intéresse pas ou rien si je décide de ne me séparer d'aucune autre carte.

Echange d'une ressource désirée contre n'importe quoi

Si j'ai la ressource désirée et qu'elle ne m'intéresse pas, alors j'accepte le contrat et je demande en échange une carte qui m'intéresse (celle qui augmenterait la valeur de ma main). Si je ne possède pas la ressource désirée par l'initiateur, je refuse le contrat.
Si l'on me demande de modifier ma requête, alors je demande la ressource qui m'intéresse le plus à ce moment et qui est différente de celle que j'avais proposée auparavant. Si aucune autre ressource ne m'intéresse, alors je ne propose aucune autre ressource en échange.

9.3 Conclusions

Les classes citées ci-dessus suffisent pour effectuer la négociation. Si l'on veut utiliser l'interface graphique, il faudra étendre en plus les deux classes correspondant à son implémentation, à savoir la classe *Graphe* et la classe *GrapheNégociation*.
Cet exemple montre bien que seules suffisent à être implémentées les actions spécifiques au problème posé, le protocole étant implémenté une fois pour toutes.

Nous allons maintenant comparer notre approche à celle de Zeus, de façon formelle puis applicative.

Chapitre 10

Etude comparative entre ZEUS et notre API de négociation

Jusqu'à présent, nous avons présenté notre protocole puis nous avons détaillé son implémentation à travers différents exemples. Nous allons maintenant examiner ce qui est proposé par Zeus afin de comparer ces 2 approches.

10.1 Présentation de la plate-forme multi-agents ZEUS de British Telecommunications

ZEUS [Zeus] est une API Java qui permet de programmer une application multi-agents. 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.

Afin de réaliser une application à l'aide de ZEUS, la méthodologie suivante est proposée (Figure 10.1) :

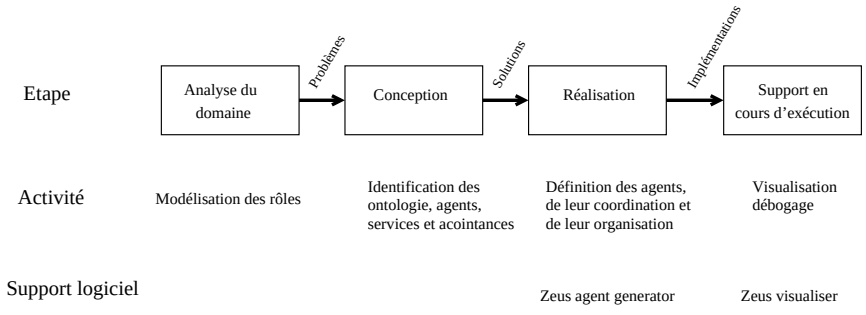

FIG. 10.1 – Méthodologie pour la réalisation d'applications avec ZEUS.

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 institutionelles.
- Le domaine des processus industriels :
  - La chaîne de production simple.
  - La chaîne de production contractuelle.

Nous allons maintenant examiner les protocoles proposés par Zeus.

10.2 Les protocoles fournis avec Zeus

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 multi-threading 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.

10.2.1 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 noeud 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 sera alors remis dans la file d'attente que lorsque le mesage sera arrivé ou que le délai ait expiré.

Examinons à présent les différents graphes fournis avec Zeus.

10.2.2 Le graphe de résolution de but proposé par défaut

Afin de mener à bien un but, le protocole de la Figure 10.2 est proposé. Ses arcs et nœuds sont décrits dans la Table 10.1. Le graphe contrôle un comportement pour résoudre des problèmes basiques pour atteindre un but, et peut être vu comme étant logiquement composé de trois parties : une phase d'allocation de ressources, une phase de négociation et une phase de gestion d'engagement. Nous allons détailler ce graphe sur un exemple.

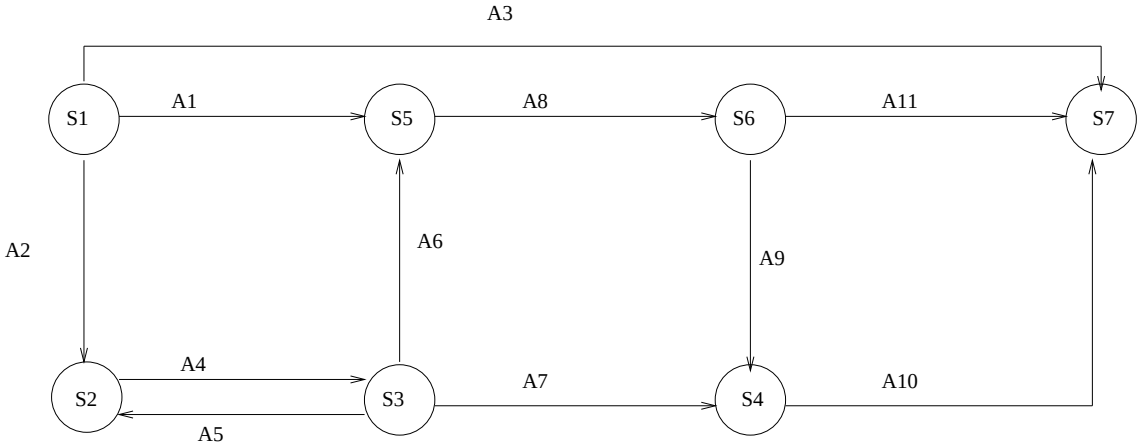

FIG. 10.2 – Graphe par défaut de résolution d'un but.

Considérons qu'un message est reçu par un agent pour réaliser le but x. Le traitement de ce message implique le lancement du graphe de résolution de but proposé par défaut avec en argument d'entrée le message reçu. Le lancement du graphe implique la création d'une instance du nœud désigné initial (S_1 de la Figure 10.2), dont l'argument d'entrée est le message.

Lors de son exécution, S_1 commence par créer une structure de données qui contiendra les informations contextuelles qui seront générées en résultat du traitement du but. Ensuite, cette structure sera initialisée avec les paramètres du but (contenus dans le message d'entrée). A présent, le nœud appelle le *Planner/Scheduler* pour planifier une séquence d'actions afin de réaliser le but. Si le *Planner* n'a aucune compétence pour traiter ce but, alors le nœud échoue, ce qui abouti à l'échec du graphe.

Admettons cependant que le *Planner* a planifié une séquence d'actions pour atteindre le but, mais a besoin de faire réaliser les sous-buts y et z par d'autres agents, ceci à cause d'un manque de temps, de compétences, d'informations ou d'autres ressources. A présent, A_2 est le seul arc valide depuis S_1, puisque sa condition de traversée est vérifiée. Nous arrivons donc en S_2, où l'agent se prépare à déléguer les sous-buts y et z ; ce qui est réalisé par l'arc A_4.

A_4 est défini comme étant un graphe parallèle 0 parmi n. Comme son argument d'entrée consiste en les sous-buts y et z, deux copies non parallèles du graphe sont créées, l'une pour

Nœud/Arc	Description/Condition de transition	Phase
S1	Crée et initialise la structure qui contient les informations contextuelles du but. Invoque le Planner.	allocation de ressources
A1	Pas besoin de participations externes et l'agent N'est PAS initiateur du but.	allocation de ressources
A2	Besoin de participations externes.	allocation de ressources
A3	Pas besoin de participations externes et l'agent EST initiateur du but.	allocation de ressources
S2	Se prépare à placer des contrats externes.	négociation
A4	Invoque le comportement de négociation du côté initiateur.	négociation
S3	Synthétise la planification avec les résultats de la négociation.	allocation de ressources
A5	Besoin supplémentaire de participations externes.	allocation de ressources
A6	Pas besoin de participations externes supplémentaires et l'agent N'est PAS initiateur du but.	allocation de ressources
A7	Pas besoin de participations externes supplémentaires et l'agent EST initiateur du but.	allocation de ressources
S4	Envoie les messages de confirmation et de refus aux agents contracteurs appropriés.	négociation
A10	VRAI	engagement
S5	Se prépare à négocier avec les agents qui ont demandé un but.	négociation
A8	Invoque le comportement de négociation du côté participant.	négociation
S6	Vérifie si le message de confirmation a été reçu de la part de l'agent qui a demandé le but.	négociation
A9	Des participations externes étaient nécessaires.	négociation
A11	Aucune participation externe n'était nécessaire.	engagement
S7	S'engage fermement à planifier la réalisation du but. Installe des moniteurs pour gérer l'exécution du plan.	engagement

TAB. 10.1 – Description des nœuds et arcs de la Figure 10.2

déléguer le sous-but y, l'autre pour déléguer le sous-but z. Le graphe étant défini 0 parmi n, A_4 sera traversé si 0, 1 ou 2 des sous-buts ont été délégués.

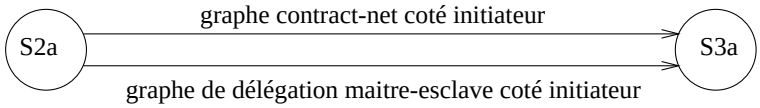

FIG. 10.3 – Graphe parallèle A4.

Le graphe défini par A_4 (dont un exemple est donné par la Figure 10.3), sert simplement à fournir un endroit où placer un des nombreux arcs/graphes potentiels qui définissent le comportement de l'initiateur des différents protocoles de négociation. Il pourrait, par exemple, inclure des arcs/graphes définissant le comportement du gestionnaire de contrat lors de la phase d'appel d'offres du protocole du contract-net, ou le comportement initial de délégation du gestionnaire de contrat du protocole de délégation maître-esclave.

Comme le processeur du graphe tente de traverser le premier arc valide possible de S_{2a} vers S_{3a}, le choix du protocole dépend de 2 facteurs. Premièrement, l'ordre dans lequel les arcs entre S_{2a} et S_{3a} sont listés dans la spécification du graphe A_4, et deuxièmement de la vérification de la condition de traversée de l'arc sélectionné. La capacité de backtracking assure que s'il existe des arcs dont la condition de traversée est vérifiée, alors au moins l'un d'entre eux sera traversé. Dans Zeus, les graphes de négociation (par exemple le graphe de négociation côté initiateur de la Figure 10.3), comportent typiquement 2 parties principales : une section d'*estimation d'applicabilité* et une section de *dialogue de négociation*. La première section vérifie simplement si le protocole est applicable dans le contexte donné, tandis que la seconde implémente effectivement le protocole.

Une fois l'arc A_4 traversé, le nœud S_3 appelle de nouveau le *Planner* avec les résultats du processus de délégation. Selon le cas, le *Planner* retourne l'un des trois résultats possibles :

1. le processus de planification a échoué et aucune solution n'a pu être trouvée pour le problème d'origine,
2. le processus de planification a réussi avec succès et aucune autre délégation n'est nécessaire,
3. une nouvelle liste de sous-buts à déléguer peut être retournée, ou une liste partitionnée de précédents sous-contrats, dont certains devront être acceptés et d'autres rejetés.

Si d'autres délégations sont nécessaires, l'arc A_5 amènera de nouveau le processeur du graphe en S_2 pour commencer une nouvelle phase de délégation. Si au contraire aucun sous-but ne doit être délégué et la planification a brillamment réussi, alors l'arc A_6 ou A_7 sera sélectionné. Ceci dépendra du fait que l'agent est en train d'essayer d'accomplir le but x pour lui-même ou pour un autre agent. Dans ce dernier cas, l'arc A_6 sera sélectionné et nous amènera en S_5.

En S_5, quelques fonctions de maintenance sont exécutées, en préparation pour la négociation avec l'agent ayant demandé la réalisation du but x. La véritable négociation du participant est réalisée par l'arc A_8. L'arc A_8 est en fait un endroit où placer un des nombreux

arcs/graphes potentiels qui définissent le comportement du participant lors des différents protocoles de négociation. Là encore, l'odre des arcs/graphes (qui remplacent A_8), et le contexte déterminent quel protocole de négociation sera sélectionné.

A la suite de cette négociation, des vérifications sont effectuées en S_6 afin de déterminer si l'agent a remporté le contrat, c'est-à-dire si un message de confirmation a été reçu de la part de l'agent ayant délégué le but x. Ensuite, l'arc A_9 ou A_{11} est traversé, selon que le but x ait eu besoin ou non de faire réaliser des sous-buts par d'autres agents. C'était le cas dans notre exemple, l'arc A_9 est donc sélectionné, nous conduisant en S_4. En S_4, les messages de confirmation sont envoyés aux agents sélectionnés pour réaliser les sous-buts y et z et ceux de refus sont envoyés aux autres agents. Finalement, en S_7, le plan construit pour réaliser le but x est ordonné pour l'exécution, et les moniteurs sont installés pour gérer le processus d'exécution du plan.

10.2.3 Le graphe du contract-net protocol

Il y a un graphe pour l'initiateur et un pour le participant. Ces graphes peuvent être insérés dans le graphe précédent entre S_{2a} et S_{3a} pour l'initiateur, et entre S_5 et S_6 pour le participant. La Figure 10.4 et la Table 10.2 décrivent l'implémentation du contract-net protocol réalisée par Zeus.

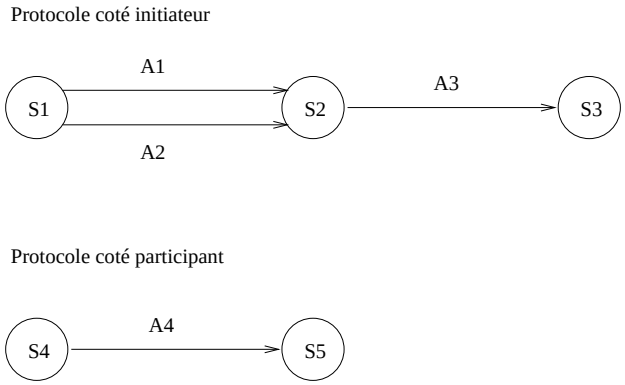

FIG. 10.4 – Graphe d'implémentation du contract net protocol.

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).

10.2.4 Remarques

Nous allons ici présenter quelques remarques importantes sur Zeus afin de pouvoir comparer notre implémentation à la leur.

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. 10.2 – Description des nœuds et arcs de la Figure 10.4

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.

Deuxièmement, 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. Eles 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).

Du point de vue de l'aide à l'implémentation, Zeus fournit un logiciel pour la création de l'ontologie, des agents et des tâches. En revanche, aucune aide n'est apportée pour l'écriture de nouveaux protocoles ou de nouvelles stratégies, ni aucune interface graphique. Notre API, quant à elle, fournit une interface graphique partielle et l'utilisation de Magique permet d'utiliser le support d'aide à la création d'agents : MagiqueGui. Nous ne sommes donc pas en reste vis à vis de Zeus.

10.3 Comparaison d'implémentations

Dans une première partie, nous allons comparer l'implémentation de la négociation pour la prise de rendez-vous que nous avions écrite avec notre API, puis avec Zeus.
Dans une seconde partie, c'est l'implémentation des places de marché pour la vente et l'achat de fruits fournie avec Zeus que nous avons codé grâce à notre API.

10.3.1 Comparaison de l'implémentation de la négociation pour la prise de rendez-vous

Remarque préliminaire : Notre implémentation a été présentée en Section 8.1, nous allons donc uniquement présenter ici celle que nous avons réalisée avec Zeus. Celle-ci ne comprend pas la renégociation en cas d'échec de la première proposition de contrat, ni les listes de priorités.

Dans le cas de Zeus, les protocoles de négociation sont représentés sous la forme de graphes, il faut donc définir ceux qui correspondent à notre protocole pour l'initiateur et le participant. Ces graphes sont présentés sur la Figure 10.5 et décrits dans la Table 10.3.

Comportement de l'initiateur	
Nœud/Arc	**Description/Condition de transition**
N1	Envoi de la proposition de contrat aux participants si les ressources ne sont ni prises, ni en cours de négociation. Marque les heures en cours de négociation. Attente de toutes les réponses.
A0	Les ressources sont en cours de négociation.
N2	Attente.
A1	Vrai.
N3	Décide si le contrat est confirmé ou non. Marque les heures non en cours de négociation et prises en cas d'accord. Envoie les messages de confirmation ou d'annulation aux participants.

Comportement du participant	
Nœud/Arc	**Description/Condition de transition**
N4	Si les ressources ne sont pas en cours de négociation, les marque en cours de négociation ; envoi d'une réponse positive à l'initiateur si elles sont libres, sinon, envoi d'un refus à l'initiateur. Attente de la réponse de l'initiateur. Marque les heures non en cours de négociation et prises en cas d'accord.
A0	Les ressources sont en cours de négociation.
N2	Attente.
A1	Vrai.
N5	Fait.

TAB. 10.3 – Description des nœuds et arcs de la Figure 10.5

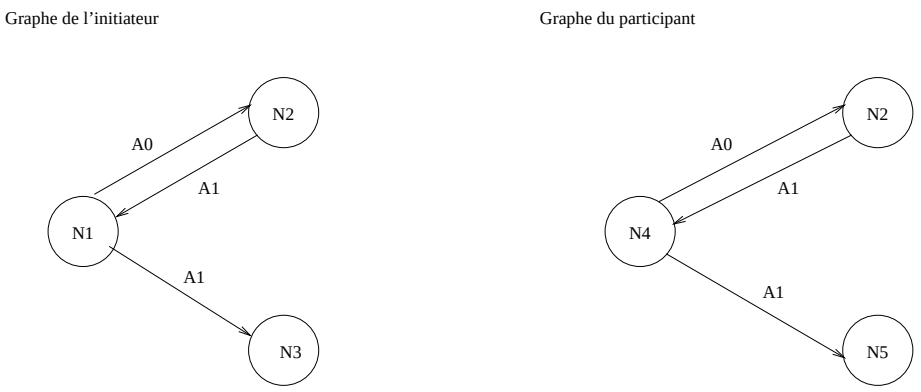

FIG. 10.5 – Graphe de négociation pour la prise de rendez-vous.

Résultats

L'implémentation que nous avons réalisé avec Zeus nous a posé plusieurs problèmes. Premièrement, aucun exemple de négociation entre plusieurs participants au sein d'un même graphe n'était fourni (Zeus ne propose que du 1 vers 1). De plus, tous les protocoles fournis n'effectuent de la négociation que sur une unique ressource, et donc, s'il y en a plusieurs, un graphe est créé par ressource. Ceci n'est pas cohérent pour la prise de rendez-vous où plusieurs tranches horaires sont souvent nécessaires.
Mis à part ces problèmes liés au protocole, il a aussi fallu modifier la classe *Engine* (celle qui s'occupe de la gestion des graphes). Il a fallu en effet lui ajouter une méthode pour lancer les graphes que nous avons défini, mais en plus, ajouter une table qui mémorise les objets dont nous avions besoin. En effet, le passage d'un nœud à un autre est le passage d'un objet à un autre et il faut pouvoir transmettre les informations nécessaires.

Zeus n'est donc pas aussi générique que notre protocole, car les classes que nous fournissons ne doivent pas être modifiés mais étendues, et notre protocole n'est pas à redéfinir et permet de négocier à plusieurs sur plusieurs ressources.

10.3.2 Implémentation des places de marché pour l'achat et la vente de fruits

Présentation de l'implémentation fournie avec Zeus

Les fruits sont vendus par cageot, il y en a 5 différents : des oranges, des poires, des bananes, des melons et des pommes. Cette implémentation met en scène 3 agents : l'OrchardBot, le SupplyBot et le ShopBot. Chacun de ces agents possède une somme d'argent pour commencer et un certain nombre de cageots de fruits. Leurs paramètres de négociation tels leur profit minimum, les pourcentages minimum et maximum par rapport au prix désiré...sont différents.
Un agent spécial : le Facilitator, sert d'intermédiaire pour mettre en relation les vendeurs

et les acheteurs. Concrètement, lorsqu'un agent décide de vendre un cageot d'oranges, par exemple, il le signale au Facilitator qui mémorise cette information. Si des acheteurs d'oranges sont présents sur le marché, le Facilitator leur signale l'arrivée de ce nouveau vendeur.
De même, lorsqu'un agent décide d'acheter des pommes, il le signale au Facilitator qui lui transmet alors la liste des vendeurs de pommes. Au cas où aucun vendeur de pommes n'est présent sur le marché, le Facilitator mémorise la demande de l'agent et lui signalera automatiquement l'arrivée d'un vendeur de pommes dès qu'il en arrivera un.
C'est toujours l'acheteur qui initie la négociation en proposant un prix d'achat au(x) vendeur(s) que lui a signalé le Facilitator. Commence alors une discussion avec chaque vendeur pour déterminer le prix d'achat. Lorsque toutes les discussions sont achevées, l'acheteur décide de choisir le vendeur à plus bas prix et conclut la négociation avec celui-ci, envoyant une réponse négative aux autres vendeurs.

Implémentation réalisée avec notre API

Notre API ne comporte pas d'agent tel que le Facilitator, nous en avons donc créé un pour cette implémentation. De plus, comme le comportement selon que l'agent vende ou achète est différent, nous avons du instancier 2 nouvelles méthodes qui seront appelées lors de l'initiation de la négociation à la place de celle que nous avions auparavant. Mais ce n'est que cette première méthode qui diffère, les autres méthodes de notre protocole restant valables pour la suite de la négociation.

De plus, pour réaliser le même protocole que Zeus, nous avons aussi engagé une négociation par vendeur, ceci nous a posé problème car notre protocole ne permet pas d'effectuer simultanément plusieurs négociations différentes sur une même ressource . En effet, l'objet d'une négociation ne peut être attribué qu'une seule fois.

Résultats

Nous avons réussi à implémenter cette application avec notre API sans opérer à des modifications majeures, puisque notre API se veut générique, nos classes de base sont abstraites et à étendre pour l'application spécifique, et donc par conséquent, il suffit de surcharger une méthode si nécessaire et ceci ne change en rien les applications définies par ailleurs.

10.3.3 Conclusions

A travers ces deux exemples d'application, nous pouvons remarquer que Zeus et notre API diffèrent. Zeus ne propose qu'un protocole de négociation de 1 vers 1, tandis que le nôtre est de 1 vers n (donc il permet le 1 vers 1). Zeus ne propose également que de négocier sur une unique ressource, en créant par conséquent plusieurs négociations si le nombre de ressources à négocier est supérieur à 1. De plus, Zeus permet d'effectuer plusieurs négociations simultanément sur une même ressource, alors que notre protocole

ne le permet pas. Cela nous paraît plus cohérent car l'objet d'une négociation ne peut être attribué qu'une seule fois.

Notre API est donc plus générique que Zeus car notre protocole sert à instancier différents problèmes alors qu'avec Zeus, il a fallu redéfinir un protocole pour l'application de la prise de rendez-vous.

Nous allons maintenant conclure ce rapport sur les perspectives que nous espèrons atteindre dans les 3 ans.

Chapitre 11

Conclusion et Perspectives futures

11.1 Conclusion

Nous avons présenté dans ce rapport le travail que nous avons réalisé sur la négociation. Dans un premier temps, nous avons étudié cette notion de négociation, à travers différents exemples d'applications (prise de rendez-vous, enchères, places de marché, jeux de type Diplomacy), mais également en étudiant ce qui a déjà été fait par d'autres équipes de recherche à travers le monde.
Nous en avons déduit un protocole de négociation générique, qui permet de réaliser de la négociation entre plusieurs agents et sur plusieurs ressources, et en prenant en compte des listes de priorité pour les participants et les ressources.

Nous avons ensuite implémenté une API Java qui met en œuvre ce protocole. Notre API est générique, portable et polyvalente. Elle utilise pour cela des ressources et des contrats abstraits. De plus, elle propose un comportement par défaut pour l'agent négociateur (initiateur ou participant), que l'on peut modifier si il ne convient pas, tout en gardant la gestion du protocole par lui-même.
Cette API permet de réaliser plusieurs négociations simultanément et de renégocier automatiquement les contrats qui n'ont pas abouti.

Grâce à cette API, nous sommes capables de réaliser différentes applications de négociation, telles la prise de rendez-vous, la vente aux enchères, les jeux de type Diplomacy ..., en ne spécifiant que les ressources et les contrats devant être négociés. Cette API fonctionne en réseau hétérogène, et les communications se font de manière asynchrone. Il est de plus possible de mélanger des agents virtuels et humains.

Nous allons maintenant présenter les perspectives que nous envisageons pour ce projet.

11.2 Perspectives futures

Nous envisageons de nombreuses perspectives pour ce travail, que nous allons organiser en quatre domaines.

11.2.1 Amélioration du protocole

Le protocole que nous avons défini reste assez simpliste, et il faudrait le doter de mécanismes plus sophistiqués afin de prendre en compte les résultats des négociations effectuées auparavant, ceci afin de changer dynamiquement les listes de priorité. Par exemple, si je sais que tous les contrats que j'ai voulu négocier avec le participant p_1 ont échoués, je ne le considérerais plus comme étant prioritaire. En revanche, si les contrats que je négocie avec p_2 ont abouti, alors je le considérerais plus prioritaire.
Pour pouvoir réaliser cela, on pourrait ajouter une base de connaissances, optimiste, pessimiste, sur les ressources, sur les participants.

Nous voudrions aussi permettre aux futurs utilisateurs d'implémenter graphiquement notre API. Ceci leur permettrait de l'utiliser plus facilement, et de n'avoir à définir que les entités nécessaires (ressources, participants, et une fonction d'évaluation si celle définie par défaut ne leur convient pas).

11.2.2 Amélioration de la structure du protocole

Pour des raisons pratiques, ce protocole a été implémenté grâce à la plate-forme Magique. En effet, elle permet de réaliser simplement des applications distribuées en utilisant RMI et des sockets.

Il faudrait séparer l'implémentation de la couche objet de l'implémentation de la couche communication afin de pouvoir par la suite utiliser cette API avec des communications par e-mail, ou l'utiliser avec une architecture client/serveur, ou encore par l'intermédiaire d'un serveur web. Ceci apporterait plus de portabilité à notre API, et elle pourrait ainsi être utilisée dans de nombreux contextes.

11.2.3 Amélioration de l'expressivité d'un problème

Nous avons présenté notre protocole sous la forme d'un automate mais cette représentation ne convient pas parfaitement. En effet, la prise en compte du temps n'est pas facile et il est donc difficile de représenter les timers et réponses par défaut considérées lors de la non réception de réponse. Il faudra donc trouver un formalisme pour exprimer notre protocole.

Un des moyens d'exprimer plus clairement ce protocole serait d'utiliser des réseaux de Pétri colorés temporels, ou encore un système de règles.

11.2.4 Perspectives industrielles

Le secteur industriel est très demandeur d'applications de négociation automatique. Nous avons vu que British Telecoms a réalisé sa propre plate-forme afin de réaliser de la négociation.
Les centrales d'achat (comme Auchan par exemple) ont tout intérêt à pouvoir négocier automatiquement les produits qu'ils veulent acheter. Ce serait un gain de temps considérable pour eux.

Un autre domaine où pourrait s'appliquer notre API est celui des jeux vidéos. Actuellement, ces jeux sont pauvres en négociation, mais de plus en plus de jeux fonctionnent par équipe. A l'intérieur d'une équipe, les agents coopérent, mais entre équipe, il serait avantageux de pouvoir négocier lors de situations de conflit.

Ceci ne constitue qu'un aperçu de toutes les perspectives que l'on peut envisager pour notre projet, mais résume bien nos ambitions et tout le travail qu'il reste à faire.

Bibliographie

[Ferber] Jacques Ferber. *Les systèmes multi-agents : vers une intelligence collective.* InterEditions, 1995.

[Weiss] Gerhard Weiss. *Multiagent Systems : A modern approach to distributed artificial intelligence.* Edited by Gerhard Weiss, 1999.

[Maes 97] Anthony Chavez, Daniel Dreilinger, Robert Guttman & Pattie Maes. *A real life experiment in creating an Agent Marketplace.* 1997.

[Jennings 99-1] Carles Sierra, Peyman Faratin & Nick R. Jennings. *Deliberative automated negotiators using fuzzy similarities.* Proc EUSFLAT-ESTYLF joint Conference on Fuzzy Logic, Palma de Mallorca, Spain, 155-158, 1999.

[Jennings 97] Carles Sierra, Peyman Faratin & Nick R. Jennings. *A service oriented negotiation model between autonomous agents.* Proc 8th European workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW-97, Ronneby, Sweden, 17-35, 1997.

[Jennings 00-1] Peyman Faratin, Carles Sierra & Nick R. Jennings. *Using Similarity Criteria to Make Negotiation Trade-Offs.* Proc 4th Int. Conf. on Multi-Agent Systems, ICMAS-2000, Boston, USA (to appear), 2000.

[Jennings 99-2] P. Faratin, C. Sierra, N.R. Jennings & P. Buckle. *Designing Responsive and Deliberative Automated Negotiators.* Proc AAAI workshop on negotiation settling conflicts and identifying opportunities, Orlando, FL, 12-18 , 1999.

[Jennings 00-2] N.R. Jennings, S. Parsons, C. Sierra & P. Faratin. *Automated Negotiation.* Proc 5th Int. Conf. on the Practical Application of Intelligent Agents and M.A.S., PAAM-2000, Manchester, UK, 23-30, 2000.

[KER 99] Martin Beer, Mark d'Inverno, Michael Luck, Nick Jennings, Chris Preist & Michael Schroeder. *Negotiation in Multi-Agent Systems.* Knowledge Engineering Review 14(3), 285-289, 1999.

[Sen 94] Rose Gamble & Sandip Sen. *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, August 1994.

[Sen 95] Mahendra Sekaran & Sandip Sen. *To help or not to help.* In the proc. of the Seventeeth Annual Conference of the Cognitive Science Society (pages 736-741), Pittsburgh, Pennsylvania, July 1995.

[Sen 99] Sandip Sen & Anish Biswas. *More than envy-free.* In the Working Papers of the AAAI-99 Workshop on Negotiation : Settling Conflicts and Identifying Opportunities, pages 44-49.

[Kraus 97] Rina Schwartz & Sarit Kraus. *Negotiation on Data Allocation in Multi-Agent Environments.* Proc. of the AAAI-97, pp.29-35, 1997.

[Sandholm 93] Tuomas Sandholm. *An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations.* Eleventh National Conference on Artificial Intelligence, AAAI-93, Washington DC, pp.256-262, (Acceptance Rate 24%), 1993.

[Straus] Straus & Al. *Negotiation, Varieties, Contexts, Processes, and Social Order.* Jossey-Bass publishers, San Francisco.

[Koning] Jean-Luc Koning, Guillaume François & Yves Demazeau. *An Approach for Designing Negotiation Protocols in a Multi-Agent System.*

[Maes 96] Anthony Chavez & Pattie Maes. *Kasbah : An Agent Marketplace for Buying and Selling Goods.* 1996.

[Jennings 96] Nick Jennings & Michael Wooldridge. *Software Agents.* IEEE Review, pp.17-20, January 1996.

[Jennings] Simon Parsons & N.R. Jennings. *Negotiation through argumentation - a preliminary report.*

[Cabri] Giacomo Cabri, Letizia Leonardi, Franco Zambonelli. *Auction-based Agent Negotiation via Programmable Tuple Spaces.*

[Koning -2] Jean-Luc Koning. *Designing and Testing Negotiation Protocols for Electronic Commerce Application.*

[Zeus] H.S. Nwana, D.T. Ndumu, L.C. Lee & J.C. Collis. *ZEUS : A Toolkit for Building Distributed Multi-Agent Systems.*

[Jennings 99-3] A. Lomuscio, M. Wooldridge & N. Jennings. *Automated Negotiation in Agent-Mediated Electronic Commerce.*

[Mathieu] Philippe Mathieu & Alain Taquet. *Une forme de négociation pour les systèmes multi-agents.* JFIADSMA'00.

[AuctionBot] P.R. Wurman, M.P. Wellman & W.E. Walsh. *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, May 1998.

[Jennings 97-2] Peyman Faratin, Carles Sierra & Nick R. Jennings. *Negotiation Decision Functions for Autonomous Agents.* 1997.

[Smith] Reid G. Smith. *The Contract Net Protocol : high-level communication and control in a distributed problem solver.* IEEE Transactions on computers, C-29(12) :1104-1113, December 1980.

[Adept] J.L. Alty, D. Griffiths, N.R. Jennings, E.H. Mamdani, A. Struthers & M.E. Wiegand. *ADEPT - Advanced Decision Environment for Process Tasks : Overview and Architecture.* 1996.

[Bensaid 99] NourEddine Bensaid. Thèse de doctorat. Université de Lille 1. *MAGIQUE, une architecture multi-agents hiérarchique.* Soutenue le Mardi 11 mai 1999.

[Bensaid 97] Nourredine Bensaid and Philippe Mathieu. *A Hybrid and Hierarchical Multi-Agent Architecture Model.* In Proceedings of the Second International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology, 21st-23rd April 1997, PAAM'97, pages 145–155, London-United-Kingdom, 1997. The Practical Application Company Ltd.

[Taquet 96] A. Taquet. *La négociation dans les systèmes multi-agents. Une application à la secrétaire virtuelle.* Mémoire de DEA session de juillet 1996.

[Fau 00] J. Fau. *Modélisation de la négociation dans les systèmes multi-agents, étude et réalisation d'une API Java réalisant la négociation.* Rapport de stage de DEA 2000.