GeNCA : Un modèle général de négociation de contrats

Philippe Mathieu — Marie-Hélène Verrons

LIFL (CNRS - UMR 8022)
Université des Sciences et Technologies de Lille
Cité Scientifique, Bât M3
F-59655 Villeneuve d'Ascq Cedex
{mathieu,verrons}@lifl.fr

RÉSUMÉ. *La modélisation de phénomènes de groupe est au cœur de la problématique multi-agents. Parmi eux se trouvent les systèmes de négociation automatique qui ont été largement étudiés dans le domaine du commerce électronique pour modéliser notamment les enchères. Dans cet article, nous présentons un modèle général de négociation pour les systèmes multi-agents, appelé GeNCA (Generic Negotiation of Contracts API), s'appuyant sur une décomposition à trois niveaux : un niveau de communication, un niveau de négociation et un niveau stratégique, seul niveau spécifique à une application fixée. Le modèle de négociation présenté se veut général et paramétrable à différents types de négociations. Il a été implémenté sous la forme d'une API Java appelée aussi GeNCA, que nous avons utilisée pour construire nos applications. GeNCA est la seule plateforme qui permette d'utiliser différents systèmes de communication et des stratégies de négociation spécifiques aux applications réalisées. Ces recherches sur la négociation s'inscrivent dans le cadre de travaux en génie logiciel pour l'intelligence artificielle et les systèmes multi-agents.*

ABSTRACT. *Modelisation of group phenomena is the core of the multi-agent problematic. Among them are automatic negotiation systems which have been largely studied in the electronic commerce field in order to modelise especially auctions. In this paper, we present a generic negotiation model for multi-agent systems, called GeNCA (Generic Negotiation of Contracts API), built on three levels : a communication level, a negotiation level and a strategic level, which is the only level specific to a particular application. The negotiation model presented here aims to be general and parameterable for different kinds of negotiations. It has been implemented by a Java API also called GeNCA, used to build our applications. GeNCA is the only platform which enables the use of different communication systems and of negotiation strategies specific to the applications achieved. These researches on negotiation take place in software engineering works for artificial intelligence and multi-agent systems.*

Revue d'Intelligence Artificielle. Volume 19 – n°6/2005, pages 837 à -

MOTS-CLÉS : *négociation, multi-agents, intelligence artificielle, génie logiciel.*
KEYWORDS: *negotiation, multi-agents, artificial intelligence, software engineering.*

1. Introduction

Avec les progrès des technologies de l'information, des systèmes multi-agents et avec le développement du commerce électronique et des places de marché sur Internet, le besoin d'agents automatiques capables de négocier avec les autres pour le compte d'un utilisateur devient de plus en plus fort. De plus, l'utilité d'utiliser un agent pendant les négociations est parfaitement justifié par l'explosion du nombre de messages échangés entre les agents. Dans certains cas, il peut être exponentiel.

Depuis plusieurs années, beaucoup de systèmes de négociations ont été réalisés dans des domaines spécifiques comme les enchères ou les places de marché, souvent dans un objectif de commerce électronique, citons notamment Zeus (Nwana *et al.*, 1999), Magnet (Collins *et al.*, 1998), le projet SilkRoad (Ströbel, 2001), la plateforme GNP (Benyoucef *et al.*, 2000) et les travaux réalisés chez les laboratoires HP (Bartolini *et al.*, 2001). Bien sûr, la négociation peut être utilisée dans d'autres domaines comme la prise de rendez-vous, les systèmes de réservation ou même les jeux vidéo, mais il semble que ces voies n'aient pas été réellement étudiées. Lors de la réalisation de telles applications, nous constatons que de nombreuses notions utilisées sont les mêmes dans la plupart des systèmes. Par exemple, les `contrats`, les `ressources`, les `contractants` et les `participants` ont un équivalent sémantique dans tous les systèmes de négociation. Notre but dans le domaine du génie logiciel est de montrer que ces notions peuvent être réifiées dans un modèle de négociation général et ouvert, et de montrer qu'il est possible de construire l'API correspondante. Ce modèle, ainsi que l'API qui l'implémente, est appelé GeNCA. Il est suffisamment large pour permettre de traiter les applications classiques de négociation sans effort d'adaptation, et possède sufisamment de paramètres pour s'adapter aux différentes applications de négociation. Nous parlons de modèle car nous proposons non seulement un nouveau protocole mais aussi un framework pour l'utiliser.

Bien qu'il soit toujours difficile de définir formellement ce qu'est la négociation, nous allons baser nos arguments sur la définition consensuelle suivante (Smith, 1980; Faratin *et al.*, 1999; Jennings *et al.*, 2000b), qui peut être appliquée à de nombreux domaines comme les enchères, les systèmes de prise de rendez-vous, les jeux ou autres.

Définition 1 *La négociation s'effectue sur un* contrat *pour obtenir des* ressources *communes et à la demande d'un* initiateur. *Elle réunit un ensemble de* participants *et un* initiateur *et se déroule jusqu'à ce qu'un accord satisfaisant un pourcentage de participants soit trouvé. Les participants cherchent également à obtenir la meilleure solution possible pour eux tout en donnant le minimum d'informations aux autres.*

Définition 2 *Un contrat est l'entité qui sera négociée. Il comprend* l'initiateur *de la négociation, les* ressources *impliquées, le* délai d'attente *des réponses et une* réponse par défaut *pour le cas où un participant ne répondrait pas à temps.*

Cette définition de la négociation s'appuie bien sûr elle aussi sur le Contract Net Protocol proposé par Smith (Smith, 1980) en 1980, qui est un fondement de tous

les travaux sur la négociation. Dans le cadre de notre étude, nous considérons qu'un minimum d'informations doit être divulgué aux autres agents, car lorsque toutes les informations sont connues, on se retrouve dans un cadre de résolution de problème, et plus dans un cadre de négociation. D'autres algorithmes sont alors mieux adaptés (CSP). Cette contrainte nous a notamment été inspirée par les différentes entreprises avec lesquelles nous collaborons.

Notre proposition s'appuie sur une architecture à trois niveaux pour permettre une réelle généricité.

– **Le niveau de négociation** qui contient la gestion des structures de données et les actes de langage nécessaires aux agents pour faire évoluer leur connaissance ;

– **le niveau de communication** qui permet aux agents d'envoyer des messages de façon centralisée s'ils sont sur le même ordinateur, ou de façon distribuée s'ils sont sur des ordinateurs différents ;

– **le niveau stratégique** qui permet aux agents de raisonner sur le problème et d'inférer leurs décisions en fonction de la connaissance obtenue des autres agents négociateurs.

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

Le succès d'une négociation dépend bien sûr de stratégies adaptées au problème traité. Nous ne parlerons pas ici de ces stratégies, qui, pour être optimales, doivent être différentes selon le type de négociation effectué. Ce domaine important dépasse le cadre de cet article et touche plus à la psychologie et à la connaissance des métiers impliqués qu'à l'informatique. Afin de s'adapter facilement à tout type de problème, nous proposons des stratégies simples, génériques, et prêtes à l'emploi que l'utilisateur peut facilement raffiner.

Afin de faciliter la définition d'une application, nous avons choisi de coder la description de l'application dans des fichiers XML. Cela permet une description aisée d'un problème donné, une vue d'ensemble sur les paramètres nécessaires et évite les recompilations pour chaque problème. Ces fichiers XML s'appuyant sur une DTD précise, il est aussi possible de récupérer facilement ces paramètres dans d'autres applications compatibles avec XML. Dans GeNCA, le serveur général possède un fichier XML de configuration qui permet de définir les notions générales comme la possibilité de se rétracter, le nombre de tours dans la négociation, etc. Chaque agent possède également son propre fichier XML de configuration pour définir les paramètres de son utilisateur (nombre minimal d'accords nécessaires pour confirmer le contrat, délai de réponse, etc).

Dans cet article, nous présentons tout d'abord la notion de négociation et l'intérêt d'un modèle général de négociation. Nous décrivons ensuite le protocole de négociation utilisé dans notre modèle (les phases du protocole, les primitives de communication et les propriétés du protocole). Puis nous détaillons les trois niveaux de notre modèle, et plus particulièrement le niveau de négociation qui contient la modélisation de la négociation que nous avons conçue. Nous présentons enfin différentes applications réalisées avec GeNCA et nous terminons en comparant notre modèle à d'autres, applicatifs ou formels, afin d'en montrer les avantages et inconvénients.

2. L'intérêt d'un système général

Rappelons tout d'abord ce qu'est la négociation. De notre point de vue, il y a négociation lorsqu'il y a une discussion, des propositions entre les protagonistes, et lorsque l'accord final satisfait au mieux tous les participants. Le simple achat d'un article dans un magasin où l'acheteur paie le prix indiqué ne constitue pas, pour nous, de la négociation, car aucun des protagonistes n'a la possibilité de faire des concessions afin d'aboutir à un accord commun. Cette réflexion nous amène à nous interroger au sujet des enchères automatiques (Klemperer, 1999). En effet, lors d'enchères électroniques, il n'y a pas toujours de concessions entre le vendeur et les acheteurs, et seuls le vendeur et le gagnant sont satisfaits de la décision finale. Les autres acheteurs sont insatisfaits car leur avis n'a pas été pris en compte. Bien que la négociation par enchères soit la plus connue, d'autres formes de négociation existent, basées sur le protocole d'interaction du Contract-Net. Bien qu'étant reconnu comme de la négociation, le Contract-Net n'implique pas de discussion ni de contre-proposition, ce qui constitue une négociation pauvre à nos yeux.

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

Tableau 1. *Descriptions des différentes enchères les plus couramment utilisées*

Examinons différentes formes de négociation, en commençant par les *enchères*. Les quatre types d'enchères les plus connues sont présentées dans le tableau 1. Les enchères *anglaises* et *hollandaises* sont des enchères publiques : tout le monde connaît le prix proposé par le vendeur, et se déroulent sur plusieurs tours. Lors d'enchères anglaises (ascendantes), le prix est successivement augmenté jusqu'à ce qu'il ne reste qu'un seul acheteur, qui gagne le bien au prix final. Dans le modèle le plus couramment utilisé, le prix est continuellement augmenté par le vendeur et les acheteurs

quittent les enchères au fur et à mesure que le prix dépasse leur budget. Les autres acheteurs observent les départs et une fois qu'un acheteur a quitté l'enchère, il ne peut plus y revenir. Elle est couramment utilisée pour la vente d'objets d'art ou d'antiquité. Les enchères hollandaises (descendantes) fonctionnent de façon exactement opposée aux précédentes : le vendeur propose un très haut prix pour son bien et le diminue jusqu'à ce qu'un acheteur se manifeste pour acquérir le bien au prix alors mentionné. Ces enchères sont notamment utilisées pour la vente de fleurs aux Pays-Bas, d'où leur nom, et pour la vente de poisson. Comme ces deux types d'enchères se déroulent sur plusieurs tours, on peut y voir une forme de négociation (il y a une sorte de discussion entre les protagonistes) bien qu'à la fin, seuls le vendeur et le gagnant soient les seuls satisfaits.

Lors d'enchères à *offres scellées*, qui se déroulent sur un seul tour, chaque acheteur propose un prix unique, sans connaître le prix proposé par les autres (d'où le terme offres scellées). Le vainqueur d'une enchère à offre scellée *au meilleur prix* est celui qui a proposé le prix le plus élevé et paie son prix, d'où leur nom. Ces enchères sont utilisées pour l'acquisition de droits sur les minéraux dans les terres possédées par l'Etat. Le vainqueur d'une enchère à offre scellée *au second meilleur prix* est celui qui a proposé le prix le plus élevé et paie le second meilleur prix proposé. Ces deux types d'enchères se déroulent sur un seul tour de parole, il n'y a donc pas, à notre sens, de réelle négociation entre les protagonistes.

Passons maintenant aux négociations basées sur le *Contrat-Net*. Comme nous l'avons mentionné auparavant, le Contract-Net (Smith, 1980) est un mécanisme d'allocation de tâches où les contractants ne formulent qu'une seule proposition avant la sélection par le contracteur. Ce protocole sans possibilité de contre-propositions ne s'apparente pas pour nous à de la négociation. D'autres protocoles se sont basés sur le Contract-Net, et proposent un mécanisme de contre-propositions. C'est notamment le cas du système proposé par Sandholm dans l'article (Sandholm *et al.*, 1995). Une autre forme de négociation très primaire consiste à formuler une proposition qui est *à prendre ou à laisser* par le ou les participants. Cette négociation se déroule sur un seul tour, sans contre-proposition ni renégociation. C'est celle que l'on rencontre tous les jours pour acheter son pain, par exemple. Là encore, nous ne considérons pas que ce soit de la négociation. Les *négociations combinées* sont utilisées lorsqu'une personne a besoin d'un ensemble d'objets non disponibles auprès d'un unique vendeur. Il faut alors négocier chaque objet (ou sous-ensemble d'objets) séparément, et avoir un mécanisme de liaison entre les négociations, car si l'ensemble des objets ne peut être acquis, aucun objet ne doit l'être. T. Sandholm (Sandholm, 2002) et S. Aknine (Aknine, 2002) étudient cette forme de négociation et plus particulièrement les stratégies à utiliser pour mener à bien cette négociation avec les meilleurs résultats possibles. Un exemple type de négociation combinée est la composition d'un voyage. Pour obtenir un voyage, il faut un moyen de transport (par exemple, un vol aller/retour vers la destination choisie), et les nuits d'hôtel pour la période du voyage. Le vol aller/retour se négocie avec une compagnie aérienne, tandis que les nuits d'hôtel se négocient avec une compagnie hôtelière. La *négociation multi-niveaux* (Meyer *et al.*, 1988) est un type de négociation où le contrat se décompose en plusieurs "sous-contrats", dépendants les

uns des autres. La négociation s'effectue séquentiellement pour chaque sous-contrat, la réussite globale est atteinte lorsque tous les sous-contrats ont été négociés avec succès. Lorsqu'un sous-contrat est en cours de négociation et qu'aucune solution n'est possible, on effectue un backtrack pour la négociation du sous-contrat précédent. La *négociation à base d'argumentation* est utilisée chez les agents logiques qui possèdent une base de connaissances avec des prédicats et des règles d'inférence. L'argumentation a alors pour but de modifier les croyances des autres agents afin qu'ils adoptent le même point de vue, les mêmes croyances et intentions que l'agent argumentant. N.R. Jennings (Parsons *et al.*, 1996) a présenté un rapport préliminaire sur cette forme de négociation en 1996, dans lequel il utilise des agents BDI (Belief, Desire, Intention) et des arguments et des règles d'inférences utilisant la notation Prolog et une extension des arguments pour indiquer le soutien ou le doute dans les propositions et pour les prouver ; tandis que S. Kraus et K. Sycara (Kraus *et al.*, 1998) ont proposé un modèle logique et son implémentation.

D'après l'analyse de ces différentes formes de négociation, on constate qu'un certain nombre de notions fondamentales sont communes. Une négociation met en jeu des ressources, qui seront rassemblées dans un contrat afin d'être négociées, et un ensemble de personnes qui participent à la négociation. Nous appelons la personne qui formule une proposition l'*initiateur* de la négociation et les personnes à qui la proposition est faite les *participants* à la négociation. Il y a toujours un ou plusieurs initiateurs (vendeur ou autre) et un ou plusieurs participants (acheteurs ou autre).

Nous constatons également qu'en utilisant des paramètres, il est possible de formuler un modèle de négociation qui a la faculté d'être général et paramètrable et qui convient parmi ces différents types de négociation notamment au CNP, à la négociation à prendre ou à laisser, et aux enchères. Nous ne prendrons pas en compte ici les négociations combinées, multi-niveaux et à base d'argumentation, qui sont plus complexes à mettre en œuvre de façon générale.

Ce florilège de types de négociation différents et la mise en lumière de leurs points communs montrent l'intérêt d'un système général de négociation.

3. Le protocole de négociation et ses propriétés

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

3.1. *Les différentes phases du protocole*

Le protocole peut être décomposé en trois phases :

La phase de proposition Cette phase est la première phase de notre protocole, elle initie la négociation. Elle comprend la proposition du contrat par l'initiateur aux participants et la collecte des réponses de chacun des participants. Chaque participant peut soit accepter, soit refuser la proposition. Nous avons choisi de ne pas inclure de contre-proposition dans cette phase car cette fonctionalité n'est pas commune à toutes les formes de négociation étudiées. Les négociations permettant les contre-propositions utilisent la phase suivante du protocole.

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

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

3.2. *Primitives de négociation*

Pour mener à bien un processus de négociation entre agents, il est nécessaire de définir des primitives spécifiques à l'initiateur et des primitives spécifiques aux participants. Notre objectif ici n'est pas de faire communiquer un de nos agents avec n'importe quel autre agent issu d'une plateforme différente (ce qui nécessiterait une plateforme "FIPA compliant" ou plus simplement des agents communiquant via ACL), mais de faciliter la réalisation d'une application avec nos agents. Le séquencement de ces primitives est représenté à la figure 1. Examinons plus en détail ces primitives.

3.2.1. *Primitives de l'initiateur*

L'initiateur possède quatre primitives de communication :

– *propose (contrat)* : c'est la première primitive que l'initiateur envoie aux participants pour leur proposer un contrat. Le contrat contient les différentes ressources à négocier.

– *demande modification (contrat)* : ce message indique aux participants que le contrat ne peut être conclu sous sa forme actuelle et qu'il faut le modifier. L'initiateur demande aux participants de lui indiquer une ou plusieurs modifications possibles du contrat afin d'en proposer un nouveau convenant mieux à l'ensemble des participants. Ce peut également être un moyen de raffinement du contrat.

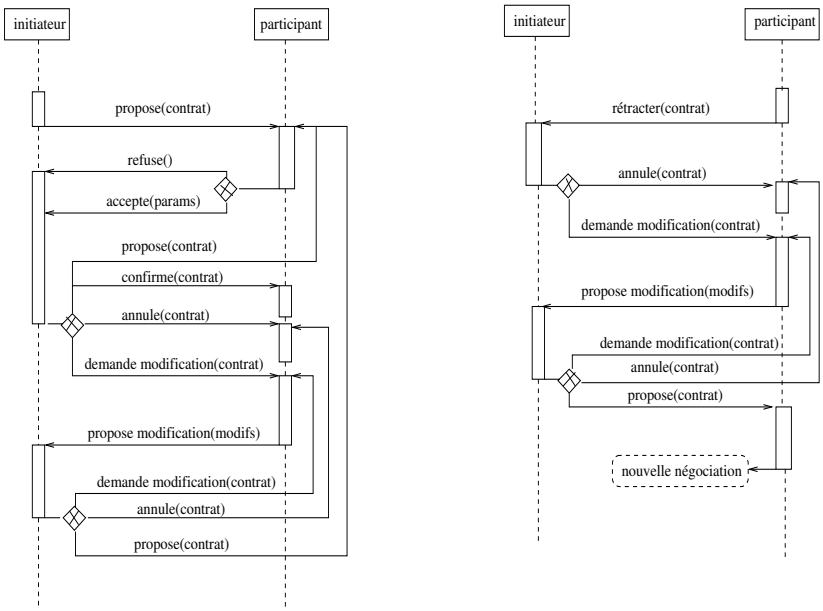

Figure 1. *Graphe d'interaction entre un initiateur et un participant : à gauche, le séquencement des messages pour la négociation d'un contrat entre un initiateur et des participants. Par souci de clarté, un seul participant est représenté ici. A droite, le séquencement des messages pour la renégociation d'un contrat pour lequel un participant s'est rétracté*

– *confirme (contrat)* : ce message indique aux participants que le contrat est confirmé. La négociation a été un succès.

– *annule (contrat)* : ce message indique aux participants que le contrat est annulé. La négociation a échoué.

3.2.2. *Primitives du participant*

Les messages envoyés par les participants sont uniquement destinés à l'initiateur. C'est un choix que nous avont fait afin que les autres participants n'aient pas connaissance de ces messages. De plus, les participants ne connaissent pas la liste des participants conviés à la négociation, ils ne peuvent donc pas former de coalition au cours de la négociation.
Le participant possède trois primitives de communication :

– *accepte (paramètres)* : ce message répond à la proposition de contrat faite par l'initiateur. Le participant indique par ce message à l'initiateur qu'il accepte le contrat tel qu'il est. Il peut y avoir des paramètres dans le cas où le contrat est partiellement instancié. C'est le cas dans les enchères Vickrey où les participants doivent proposer

un prix pour l'article mis en vente.

– *refuse* : ce message répond à la proposition de contrat faite par l'initiateur. Le participant indique à l'initiateur qu'il refuse le contrat.

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

Une primitive de communication est commune aux initiateurs et aux participants :

– *rétractation (contrat)* : le contrat avait été confirmé mais le participant ou l'initiateur ne peut (ou ne veut) plus l'honorer. Il décide donc de se rétracter.

3.3. *Dynamique opérationnelle*

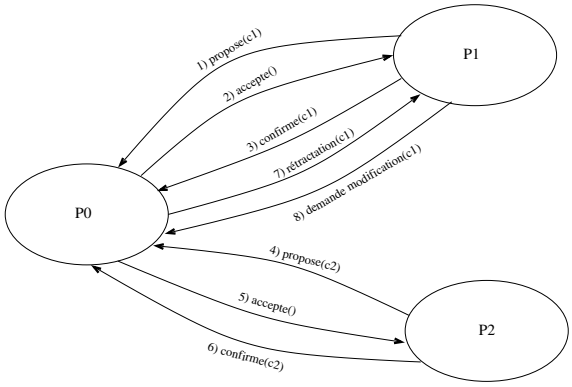

Figure 2. *Exemple de négociation avec renégociation : P_1 propose un contrat à P_0 qui l'accepte. Puis P_2 propose un contrat entrant en conflit avec le précédent à P_0. Celui-ci trouve ce nouveau contrat plus avantageux et l'accepte. Comme P_2 confirme le contrat, P_0 se rétracte pour celui qu'il avait pris avec P_1. P_1 entame donc une renégociation du contrat avec P_0*

Examinons un exemple de négociation faisant intervenir 3 personnes : P_0, P_1 et P_2 (figure 2). Pour cet exemple, P_0 considère que P_2 est plus prioritaire que P_1. Dans un premier temps, P_1 propose un contrat c_1 à P_0 qui accepte. P_1 joue le rôle de l'agent initiateur et P_0 celui de l'agent participant. P_1 crée le contrat et envoie à P_0 le message *propose(c_1)*. P_0 reçoit le contrat, l'étudie et envoie le message *accepte()* à P_1 pour lui signaler qu'il accepte les termes du contrat. P_1 confirme donc le contrat auprès de P_0 (message *confirme(c_1)*). Puis P_2 propose un contrat c_2 à P_0 pour les mêmes ressources. P_0, considérant P_2 plus prioritaire, envoie un message *accepte()* à

P_2. Celui-ci confirme le contrat auprès de P_0 (message *confirme(c_2)*). P_0 prend alors le contrat avec P_2 et annule le précédent contrat pris avec P_1 (message *rétractation(c_1)*). Lorsque P_1 reçoit ce message, il décide de demander une modification pour ce contrat à P_0 (message *modifier contrat(c_1)*).

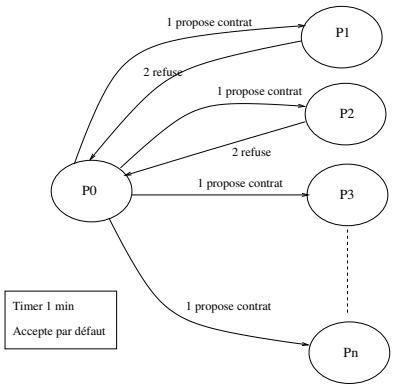

Figure 3. *Exemple de négociation avec timer : P_0 propose un contrat à n participants. Seuls deux d'entre eux répondent (ils refusent la proposition). Comme la réponse par défaut est l'acceptation de la proposition, P_0 va confirmer son contrat auprès des participants n'ayant pas répondu*

Afin de voir l'utilité d'un délai de réponse, prenons l'exemple décrit en figure 3. P_0 est l'initiateur d'un contrat avec n participants : P_1 à P_n ($n > 5$). P_0 veut qu'au moins 50 % des participants acceptent le contrat pour le confirmer. La réponse par défaut est d'accepter et le temps d'attente est fixé à 1 minute. P_1 et P_2 refusent le contrat. Après une minute, les autres participants n'ont pas répondu. P_0 considère donc la réponse par défaut pour ces $n-2$ participants, c'est-à-dire qu'ils acceptent le contrat. Au total, il y a donc $n-2$ acceptations et 2 refus, le contrat est donc confirmé.

3.4. *Paramètrage du protocole*

Nous avons vu différents types de négociation et une proposition d'un protocole de négociation général au sens où il est facilement adaptable. Nous décrivons ici différents paramètres permettant, à partir du noyau de négociation, de formaliser différents types de négociation. Nous avons choisi de configurer ces paramètres dans un fichier au format XML pour l'ouverture du système à l'extérieur. Nous avons donc conçu un fichier DTD (figure 4) afin de pouvoir valider les fichiers de configuration. Il y a un fichier pour configurer l'application et chaque agent peut redéfinir les valeurs des paramètres dans un fichier qui lui est propre.

Lors de négociations distribuées comme c'est le cas lorsqu'il s'agit d'agents qui travaillent pour le compte d'un utilisateur, il se peut qu'un participant ne réponde pas

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

Figure 4. *Fichier DTD pour la configuration du protocole de négociation. On y retrouve le délai de réponse des participants associé à une réponse par défaut, le nombre minimal d'accords nécessaires à la conclusion du contrat, le nombre de tours de parole des participants et le nombre de modifications qu'ils peuvent proposer par tour de parole. Il est également indiqué si la rétractation est autorisée, et dans ce cas, le nombre maximal de renégociations qui seront effectuées*

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

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

Afin de ne pas avoir une phase de conversation infinie, nous avons défini le nombre de tours de négociation, c'est-à-dire le nombre de fois où les participants pourront

proposer une modification du contrat, faire une contre-proposition. Nous avons choisi de borner la durée de la négociation par un temps d'attente des réponses et un nombre de tours de parole plutôt que par une durée maximale comme c'est souvent le cas, car ainsi, nous pensons que la négociation sera plus efficace. On peut effectivement supposer que le nombre de contre-propositions effectuées sera supérieur si les agents doivent répondre dans des délais plus brefs que lorsqu'ils ne connaissent qu'une date limite de fin de négociation et dans ce cas répondent moins rapidement aux initiateurs. Mais cette utilisation provient plus spécifiquement du fait qu'une date limite de négociation pose de nombreux problèmes, et plus principalement se pose le problème d'une référence de temps universelle. En effet, la synchronisation des différents ordinateurs sur lesquels tournent les agents est un réel problème que nous n'avons pu résoudre.

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

Nous avons évoqué la nécessité de pouvoir se rétracter d'un contrat pris dans certains types de négociation. Pour cela, un paramètre booléen indique si la rétractation est possible ou non, et si elle l'est, un autre paramètre fixe le nombre maximal de fois où il est possible de renégocier le contrat : `<retraction-possible value=''true''/>` et `<nbRenegotiations>5</nbRenegotiations>`.

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

3.5. *Propriétés de notre protocole*

3.5.1. *Renégociation automatique*

Souvent, lors de négociations, certains contrats ne peuvent plus être honorés et doivent être renégociés. C'est par exemple le cas lors de la prise de rendez-vous. Un participant peut être amené à annuler un rendez-vous à cause d'un imprévu mais voudrait en obtenir un autre. Pour cette raison, nous proposons de renégocier automatiquement les contrats qui doivent l'être. Lorsque la renégociation est autorisée, l'initiateur d'un contrat peut automatiquement le renégocier pourvu que le nombre maximum de renégociations ne soit pas atteint pour ce contrat. Dès qu'un initiateur reçoit

une rétractation pour un contrat, il peut choisir de l'annuler pour tous les participants et éventuellement d'entrer dans une nouvelle phase conversationnelle afin de trouver un nouvel accord. L'initiateur peut également ne rien faire si le nombre d'accords reste supérieur au nombre minimal d'accords nécessaires pour le succès de la négociation.

3.5.2. *Cardinalité de la négociation*

La cardinalité de la négociation est une notion importante pour les SMA. Il s'agit de savoir combien d'agents négocient entre eux. Différents types de cardinalité de négociation existent (Guttman *et al.*, 1998), depuis la négociation de 1 vers 1 jusqu'à n vers m. Kasbah (Chavez *et al.*, 1996) est un exemple de négociation de 1 vers 1 : un acheteur négocie un article avec un vendeur à la fois. Cette forme de négociation n'est utile que lorsque seulement deux personnes sont impliquées dans la négociation. Mais lorsque la négociation implique plusieurs participants avec un initiateur, il s'agit de négociation de 1 vers n. Notre protocole permet à plusieurs initiateurs de proposer un contrat à un ensemble de participants simultanément, il s'agit donc de négociation de n vers m agents, contrairement à Kasbah ou Zeus.

3.6. *Evaluation de notre protocole*

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

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

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

Ce protocole peut, par exemple, convenir pour une négociation du type "à prendre ou à laisser" si la phase de conversation n'est pas utilisée, c'est-à-dire si le paramètre `nbRounds` est fixé à 0. Ce protocole permet également d'implémenter des enchères (*cf.* tableau 1), que ce soient des enchères à offres scellées ou des enchères hollandaises. Pour les enchères à offres scellées, l'initiateur propose un article et les participants répondent en donnant un prix en argument de la méthode *accepte* s'ils veulent enchérir

pour l'article, sinon ils rejettent la proposition. Si aucun participant n'a proposé de prix satisfaisant pour l'initiateur, on entre dans la phase de conversation où les modifications proposées sont un nouveau prix. Ce processus se termine lorsqu'un prix satisfaisant a été proposé, ou lorsque personne ne surenchérit ou encore lorsque le nombre de tours de parole des participants défini par l'initiateur est atteint.

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

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

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

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

3.7. *Complexité du système*

La complexité est une caractéristique importante pour la négociation. C'est l'une des principales raisons pour laquelle il est nécessaire d'utiliser des agents négocia-

teurs. Examinons la complexité en nombre de messages induite par notre protocole. Nous supposons ici qu'un message est envoyé par destinataire.

Supposons que m personnes soient impliquées dans la négociation d'un contrat (l'initiateur et $m-1$ participants).

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

Dès qu'un participant n'est pas d'accord, l'initiateur *demande une modification* du contrat aux participants qui en envoient alors une à l'initiateur (message *propose modification*). $2*(m-1)$ messages sont alors échangés. L'initiateur envoie une nouvelle proposition qui sera acceptée, ce qui ajoute $3*(m-1)$ messages. Au total, $7*(m-1)$ messages sont échangés, en tenant compte de la première proposition de l'initiateur et les réponses des participants dont au moins une est négative. L'initiateur envoie $4*(m-1)$ messages et en reçoit $3*(m-1)$. Chaque participant reçoit 4 messages et en envoie 3.

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

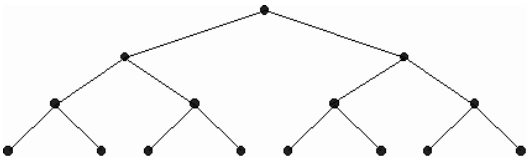

Figure 5. *Complexité de la négociation d'un contrat avec deux participants nécessitant des renégociations en cascade. Chaque point représente un agent. La complexité globale est exponentielle (en O(2^n))*

Supposons que tous les agents doivent remettre en question un contrat jusqu'à une profondeur n (figure 5), que les contrats soient pris avec l'initiateur et deux participants et que tous ces déplacements soient indépendants.

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

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

Si nous supposons maintenant que les contrats ne sont plus indépendants mais que les agents situés à la profondeur n demandent à l'initiateur du contrat principal

d'en déplacer un autre, le nombre de demandes de renégociations sera de 2^n pour l'initiateur.

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

4. Les 3 niveaux de notre modèle

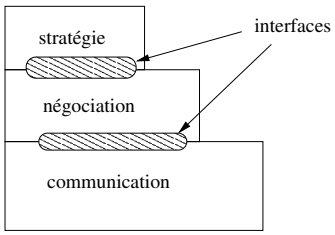

Figure 6. *Les trois niveaux de notre modèle général. Le niveau de négociation est au cœur du modèle. Il possède une interface avec le niveau de communication qui se charge de faire communiquer les agents entre eux selon le mécanisme fournit par l'application. Il possède également une interface avec le niveau stratégique qui définit la façon dont l'agent négociera ses contrats, ce qui est également spécifique à l'application réalisée*

Comme nous l'avons indiqué précédemment, notre modèle général de négociation s'appuie sur trois niveaux (communication, négociation, stratégique), indépendants les uns des autres (figure 6), afin de faciliter l'adaptation de GeNCA à une application donnée. Par construction, il est possible de modifier un niveau sans que cela ait de conséquence sur les autres. Dans cette section, nous présentons les niveaux de communication et de stratégie ainsi que leurs interfaces avec le niveau de négociation, qui sera l'objet de la prochaine section.

4.1. *Le niveau communication*

Ce niveau est responsable de la communication entre les agents. Il définit les primitives de base que tous les agents doivent implémenter pour négocier avec GeNCA.

Notre modèle utilise un service d'abonnement à un serveur qui gère les participants et les ressources qui seront négociées. Ce serveur permet de conserver, selon le type de ressources négociées (articles pour une vente aux enchères, tranches horaires pour une prise de rendez-vous), les différents participants et les ressources en jeu. Lorsqu'un agent s'inscrit, le serveur lui communique la liste des participants inscrits pour la même application que lui, ainsi que la liste des ressources qui seront négociées. Son arrivée est elle aussi communiquée aux participants déjà inscrits à la même

application, ainsi que les ressources apportées. Les participants connaissent uniquement les noms (pseudos) des autres participants, mais pas leur identifiant physique. Ils ne peuvent donc communiquer entre eux que par l'intermédiaire du serveur qui est le seul à connaître les identifiants physiques.

Pour interagir avec le serveur, quatre primitives de base sont utilisées :

– *souscrire(nom,identifiant,type de négociation,resources)* : cette méthode permet à un agent négociateur de s'inscrire auprès du serveur. L'agent donne son nom logique (pseudo), son identifiant physique (adresse IP, mail, etc), le type d'application de négociation dans laquelle il veut participer (enchères, prise de rendez-vous, etc) et ses ressources personnelles qu'il va négocier.

– *se connecter(nom,identifiant)* : cette méthode permet à un agent déjà inscrit de se connecter au serveur.

– *se déconnecter(nom)* : cette méthode permet à un agent connecté de se déconnecter du serveur, ce qui le rend indisponible pour négocier.

– *envoyer un message(destinataires, message)* : cette méthode est invoquée par un agent négociateur pour envoyer un message à un ensemble de participants. Si un destinataire n'est pas connecté, le message est placé dans une boîte aux lettres et sera envoyé au destinataire lors de sa prochaine connection.

Le serveur a lui aussi une primitive de base lui permettant d'envoyer un message à un destinaire : *envoyer(destinataire, message)*.

L'interface Java entre la couche Communication et la couche négociation est décrite par le code suivant.

```
public interface Communicator {

    /** to ask the NameServer to send a message to a list of participants
        @param to the list of receivers (their names)
        @param msg the message to send
    **/
    void sendMessage(Vector to,AntsMessage msg);

    /** to subscribe to the negotiation system
        @param name the user/agent name
        @param address the user/agent address
        @param negotiationType the type of negotiation done
        @param agentResources the own resources of the agent
    **/
    void subscribe(String name,Object address, String negotiationType,
                           Vector agentResources);

    /** to connect to the negotiation system
        @param name the user/agent name
        @param address the user/agent address
    **/
    void connect(String name,Object address);
```

```
    /** to disconnect to the negotiation system
        @param name the user/agent name
    **/
    void disconnect(String name);

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

La plateforme Magique (, *http ://www.lifl.fr/SMAC/projects/magique*, n.d.), proposée par le LIFL (Laboratoire d'Informatique Fondamentale de Lille), permet de distribuer physiquement des agents et de les faire communiquer. Cette plateforme offre une gestion des compétences des agents, ce qui leur permet d'en acquérir de nouvelles au cours de leur vie ou de transmettre les leurs. Elle propose également une organisation hiérarchique des agents afin de pouvoir facilement déléguer une tâche ou transmettre une compétence. Pour communiquer entre eux, les agents possèdent trois primitives : *perform*, *ask* et *askNow*. Ces primitives prennent en paramètre le nom de l'agent avec qui l'on communique si l'on effectue une connexion directe, sinon l'un des agents possèdant la compétence requise traitera la requête. Le second argument est la compétence invoquée, qui représente un nom de méthode à invoquer chez l'agent destinataire. Puis les arguments de la compétence sont indiqués soit dans un tableau d'objets, soit directement. La primitive *perform* est un appel non bloquant sans résultat attendu, la primitive *ask* est un appel non bloquant avec résultat attendu, et la primitive *askNow* est un appel bloquant. A titre d'exemple, l'implémentation de l'interface Communicator avec la plateforme Magique est présentée juste après. Dans cette implémentation, seules les primitives de communication *perform* et *askNow* sont utilisées. On aurait tout aussi bien pu prendre n'importe quelle autre plateforme distribuée comme Madkit (, *http ://www.madkit.org*, n.d.), Jade (, *http ://sharon.cselt.it/projects/jade/*, n.d.), etc. qui offre à notre niveau les mêmes possibilités (distribution des agents et communication entre eux).

```
public class MagiqueCommunicator extends MagiqueDefaultSkill
implements Communicator {

    private Agent myAgent;

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

    public void sendMessage(Vector to,AntsMessage msg){
        perform("sendMyMessage",to,msg);
    }
```

```
    public void subscribe(String name,Object address,
           String negotiationType, Vector agentResources){
        perform("subscribe",new Object[]{name,(String)address,
                    negotiationType,agentResources});
    }

    public void connect(String name,Object address){
        perform("connectMe",new Object[]{name,(String)address});
    }

    public void disconnect(String name){
        perform("disconnectMe",new Object[]{name});
    }

    public void sendNow(Object to,AntsMessage msg){
        askNow((String)to,"receiveMessage",msg);
    }
}
```

4.2. *Le niveau stratégique*

Les stratégies utilisées pendant la négociation sont spécifiques à l'application réalisée. En effet, négocier l'obtention de ressources partagées et négocier les prix dans une place de marché ne se fait pas de la même façon. C'est pourquoi nous avons choisi de séparer ce niveau de conception et de laisser aux utilisateurs la responsabilité de la création des stratégies adaptées à leur domaine d'application. Elaborer une bonne stratégie nécessite en effet une expertise du domaine que nous ne pouvons fournir a priori. Nous proposons néanmoins deux stratégies par défaut (l'une pour l'initiateur et l'autre pour le participant) qui sont générales et adaptables à de nombreuses applications.

Nous avons vu que notre protocole distingue deux rôles : initiateur et participant. La stratégie de négociation n'est pas la même selon le rôle de l'agent, il y a donc deux sortes de stratégies. La stratégie de l'initateur lui permet de décider de confirmer ou d'annuler le contrat, et de synthétiser les différentes propositions de modifications des participants afin de proposer un nouveau contrat. L'autre stratégie est celle du participant qui l'utilise pour décider d'accepter ou de refuser la proposition de contrat et de trouver une modification pour le contrat lorsque l'initiateur en demande une. Chaque agent possède donc deux stratégies : l'une utilisée lorsqu'il tient le rôle d'initiateur, et l'autre utilisée lorsqu'il tient le rôle de participant. Pour chacune de ces deux stratégies, une interface Java correspondant aux différentes décisions à prendre a été définie. Ces interfaces sont décrites par le code suivant.

```
public interface InitiatorStrategy {

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

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

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

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

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

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

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

public interface ParticipantStrategy {

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

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

Afin de pouvoir élaborer des stratégies, nous avons mis en place dans le modèle deux listes de priorités, une pour les ressources et une pour les personnes. Ainsi chaque agent peut définir un ordre sur les ressources et les personnes grâce auquel il pourra choisir le contrat qu'il souhaite prendre en cas de conflits. Par exemple, au sein d'une équipe, le chef aura une plus grande priorité que les collègues. Si deux collègues ont pris un contrat sur une ressource et que le chef leur propose un contrat sur la même ressource, ils prendront le contrat avec le chef et annuleront le leur. Ces listes de priorités permettent également aux initiateurs de pondérer les modifications proposées par les participants selon leur priorité et les priorités des ressources qu'ils proposent.

Nous fournissons également des méthodes permettant de consulter les négociations précédantes en fonction de la personne et/ou des ressources impliquées afin de prendre en compte les résultats des négociations passées dans la prise de décision pour une nouvelle négociation.

4.2.1. *Stratégie par défaut de l'initiateur*

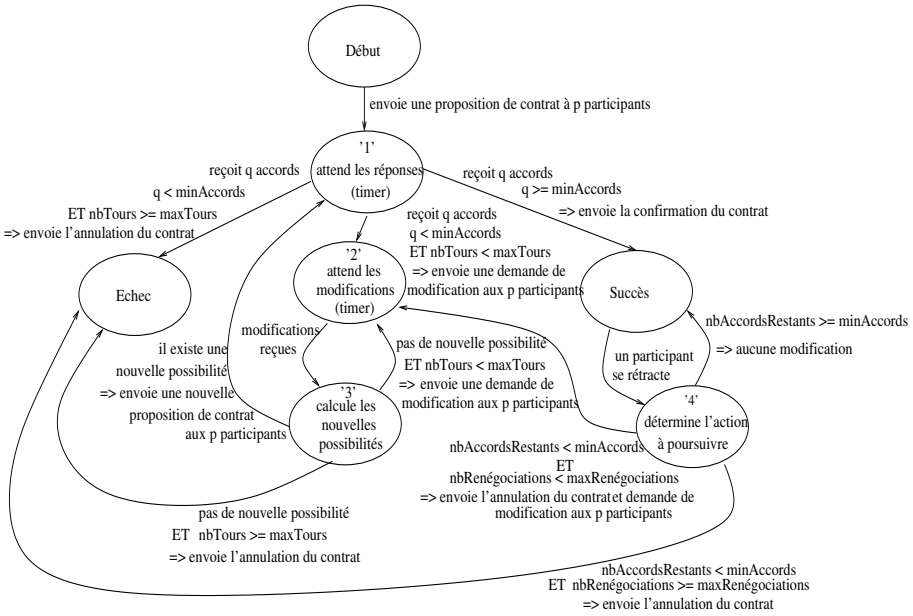

Figure 7. *Le graphe de la stratégie par défaut de l'initiateur. Cet automate à états décrit le comportement de l'initiateur face aux messages reçus des participants et aux paramètres de négociation*

La stratégie par défaut de l'initiateur peut être formalisée par un graphe (figure 7). Tout commence par la proposition d'un contrat à un ensemble de `p` participants. L'initiateur passe dans l'`état 1` où il attend les réponses des participants. Lorsque l'initiateur envoie la proposition de contrat, il définit un délai d'attente des réponses afin

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

1) Si `q` est supérieur ou égal au nombre minimal d'accords nécessaires à la prise du contrat, alors il confirme le contrat et passe à l'`état de succès`.

2) Si `q` est inférieur au nombre minimal d'accords nécessaires à la prise du contrat, et si le nombre de tours de négociation déjà effectué est supérieur ou égal au nombre maximal de tours de négociation choisi par l'initiateur à la création du contrat, il annule le contrat et passe dans un `état d'échec`.

3) Si `q` est inférieur au nombre minimal d'accords nécessaires à la prise du contrat, et si le nombre de tours de négociation déjà effectué est inférieur au nombre maximal de tours de négociation choisi par l'initiateur à la création du contrat, il demande aux `p` participants de lui proposer une modification pour le contrat et passe à l'`état 2`.

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

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

où ***priorité(***r_i***,init/***$part_j$***)*** représente la priorité de la ressource r_i pour l'initiateur ou le participant $part_j$ (si la resource n'a pas été proposée, on lui donne une priorité égale à zéro), ***priorité(init,init)*** est la priorité que l'initiateur s'est donnée et ***priorité(***$part_j$***,init)*** est la priorité du participant $part_j$ pour l'initiateur.

Remarque : Les priorités sont comprises entre 1 et 10, 10 étant la meilleure note.

Ces notes permettent à l'initiateur de trouver une nouvelle possibilité de ressource selon ses préférences et celles des participants. Bien sûr, d'autres formules peuvent être utilisées, mais nous avons choisi celle-ci car nous pensons qu'elle prend mieux en compte l'opinion de tous et l'importance donnée à cette opinion par l'initiateur.

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

2. Si le nombre maximal de tours de négociation est atteint, alors l'initiateur envoie une annulation pour le contrat et passe dans un `état d'échec`.

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

4.2.2. *Stratégie par défaut du participant*

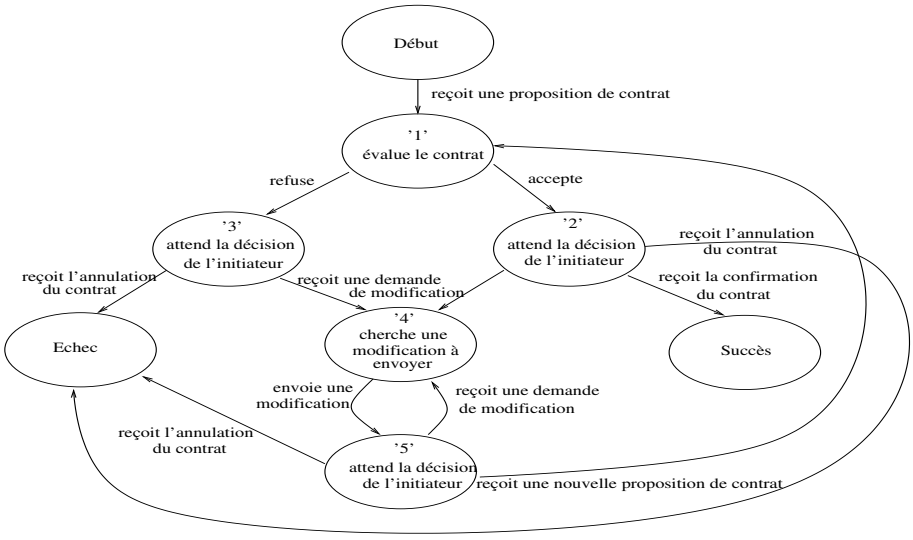

Figure 8. *Le graphe de la stratégie par défaut du participant. Cet automate à états décrit le comportement du participant face aux messages reçus de l'initiateur*

La stratégie par défaut du participant peut elle aussi être formalisée par un graphe (figure 8). Tout commence lors de la réception d'une proposition de contrat. Le participant passe dans l'`état 1`. Cet état est celui où l'agent examine le contrat, l'évalue et choisit soit de l'accepter, et dans ce cas il passe à l'`état 2`, soit de le refuser et passe à l'`état 3`. Lorsqu'il est dans l'`état 2`, le participant attend la réponse de l'initiateur qui sera soit la confirmation du contrat, soit une demande de modification du contrat,

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

5. Le niveau de négociation

Dans cette section, nous décrivons le niveau de négociation de notre modèle. Ce niveau intègre le protocole de négociation (cf. section 3) et a la charge de gérer les différentes négociations de l'agent. Afin de modéliser la négociation, deux aspects doivent être traités : l'aspect statique qui contient les structures de données nécessaires à la description d'une négociation, et l'aspect dynamique qui permet de décrire l'avancement des négociations.

5.1. *Les structures de données de description de la négociation*

Les ressources

Les ressources sont les objets que chaque agent essaie d'obtenir. Les ressources sont abstraites et peuvent aussi bien représenter des articles que des tranches horaires ou encore des cageots de fruits. N'importe quel objet peut être une ressource, incluant un contrat comme type de donnée, mais pas le contrat en train d'être négocié lui-même.

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, le *délai de réponse* pour éviter les deadlocks et la *réponse par défaut* en cas de non-réponse du participant. C'est lui qui sera proposé aux participants par l'initiateur.

Un contrat peut contenir une ou plusieurs ressources. Si un contrat porte sur plusieurs ressources, cela implique que l'obtention de toutes les ressources est nécessaire pour confirmer le contrat, les ressources forment un tout. Si toutes les ressources

ne peuvent pas être obtenues, la négociation échouera. Tandis que plusieurs contrats sur une seule ressource signifie que l'agent veut obtenir les ressources séparément, c'est-à-dire que si l'agent n'arrive pas à obtenir l'une des ressources, ce n'est pas grave, toutes les négociations (une par contrat) sont indépendantes, l'échec de l'une n'a pas d'importance sur la réussite ou l'échec d'une autre.

Le contrat ne contient pas l'ensemble des participants. C'est un choix que nous avons fait de sorte à préserver leur "vie privée", et donc un participant ne sait pas s'il y a d'autres participants ni leur réponse aux propositions ou leur proposition de modification. Ceci évite également la formation de coalitions entre les participants.

Les *propriétés* du contrat sont constituées de la liste des *participants* à la négociation, du *nombre minimal d'accords* pour confirmer le contrat, du *nombre de tours* dans la négociation et du *nombre de renégociations autorisées*. Elles caractérisent la façon de négocier le contrat et sont privées pour l'initiateur. Elles mémorisent également les participants qui ont accepté le contrat, le nombre d'accords reçus et le nombre de tours et le nombre de renégociations effectuées.

Le contrat et ses propriétés sont des structures de données "mémoire" de base qui contiennent les propriétés communes à tout type de négociation.

5.2. *Les structures de données pour la dynamique de la négociation*

Les micro-agents

Chaque agent négociateur (initiateur ou participant) est constitué de micro-agents [1] appelés buts et engagements. Les buts sont créés chez les initiateurs de contrat tandis que les engagements sont créés chez les participants à une négociation. Les micro-agents sont des agents purement réactifs à l'intérieur de l'agent négociateur. Chaque micro-agent est responsable de la négociation du contrat pour lequel il a été créé. Il peut y avoir simultanément plusieurs buts, comme il peut y avoir plusieurs engagements au sein d'un agent négociateur. En fait, il y a autant de buts qu'il y a de contrats créés par l'agent et qui sont en cours de négociation, et il y a autant d'engagements qu'il y a de négociations en cours où l'agent intervient en tant que participant. Chaque micro-agent interagit avec la stratégie (cf. section 4.2) qui lui a été attribuée. A chaque arrivée de message concernant la négociation de son contrat, le micro-agent informe la stratégie qui prend en compte le message. Celle-ci indique par la suite au micro-agent le message qu'elle souhaite envoyer en réponse. Prenons l'exemple d'une proposition de contrat arrivant à un micro-agent engagement, celui-ci envoie la proposition à la stratégie, via la méthode *whatDoIanswer* de l'interface *ParticipantStrategy*, qui lui demande ensuite d'envoyer une acceptation ou un refus.

1. Nous parlons de micro-agents car chaque micro-agent a sa propre forme d'autonomie. En effet, chaque micro-agent réagit aux messages de négociation qui lui arrive et envoie une réponse à l'initiateur s'il participe à la négociation (par exemple l'acceptation de la proposition) ou à l'ensemble des participants s'il initie la négociation (par exemple la confirmation du contrat).

Représentation graphique des négociations

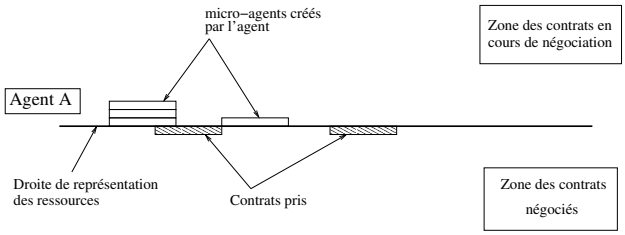

Figure 9. *Représentation des ressources. Une droite segmentée représente les ressources (un segment = une ressource). En dessous de la droite, on trouve les contrats pris par l'agent (ceux qui ont été confirmés) face aux ressources concernées. Au dessus de la droite, les contrats en cours de négociation sont empilés sur les ressources qu'ils nécessitent*

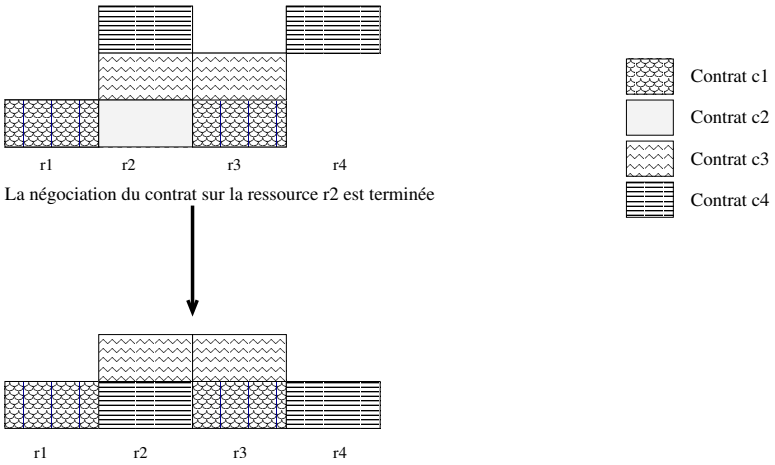

Figure 10. *Gestion des négociations entrant en conflit pour une ou plusieurs ressources. Les contrats c1 et c2 sont en cours de négociation pour les ressources {r1,r3} et {r2}. Le contrat c3 attend la libération des ressources {r2, r3}. Le contrat c4 attend la libération de la ressource {r2}. Supposons que le contrat c2 soit annulé. La ressource r2 est donc libérée. Le contrat c3 ne peut toujours pas être négocié car la ressource r3 n'est pas libre. En revanche, le contrat c4 qui nécessite les ressources {r2, r4} peut maintenant être négocié*

Afin de représenter les ressources, nous avons défini un modèle graphique (figure 9) dans lequel une droite segmentée représente l'ensemble des ressources disponibles et chaque ressource correspond à un segment. La partie inférieure contient les contrats négociés sur ces ressources, c'est-à-dire les contrats pris par l'agent,

les ressources correspondantes ne sont alors plus libres. La partie supérieure contient les micro-agents qui sont en cours de négociation sur les ressources correspondantes. Cette partie supérieure représente une matrice où les colonnes sont les ressources et où les micro-agents arrivent dans une liste de type premier arrivé, premier sorti. En fait, 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. Le micro-agent actif (en cours de négociation) 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 micro-agent 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. Pour fixer les idées, prenons l'exemple de la figure 10. Il y a quatre contrats : c_1, c_2, c_3 et c_4, et quatre ressources : r_1, r_2, r_3 et r_4. Le premier contrat c_1 qui arrive porte sur les ressources r_1 et r_3. Comme aucun contrat n'a encore été proposé, toutes les ressources sont libres et donc le contrat c_1 est placé à la surface de la matrice dans les colonnes correspondant aux ressources r_1 et r_3. Le processus de négociation du contrat c_1 est donc entamé. Puis, un second contrat c_2 est proposé pour la ressource r_2. Comme aucun contrat n'est en cours de négociation pour cette ressource, le contrat c_2 est placé à la surface de la matrice, dans la colonne correspondant à la ressource r_2. La négociation du contrat c_2 est entamée. Un troisième contrat c_3 est proposé pour les ressources r_2 et r_3. Malheureusement, ces deux ressources sont en cours de négociation. Le contrat c_3 est donc placé dans la matrice dans les colonnes correspondant aux resources r_2 et r_3 au dessus des contrats c_1 et c_2. La négociation du contrat c_3 est mise en attente. Un quatrième contrat c_4 arrive ensuite. Ce contrat porte sur les ressources r_2 et r_4. Comme la ressource r_2 est en cours de négociation pour le contrat c_2, le contrat est placé au-dessus des autres. Comme c_3, le contrat c_4 est mis en attente. Supposons que la négociation du contrat c_2 soit terminée. Le contrat c_2 est donc retiré de la matrice, libérant ainsi la ressource r_2. Le contrat c_3 ne peut descendre à la surface de la matrice car la ressource r_3 n'est pas libre. Donc, c'est le contrat c_4, qui nécessite les ressources r_2 et r_4, qui tombe à la surface de la matrice. L'engagement correspondant au contrat c_4 est alors réveillé et entame son processus de négociation.

Pour résumer, c'est cette matrice qui indique aux micro-agents (but ou engagement) si leur processus de négociation peut commencer ou non (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 la négociation du contrat arrivant, son micro-agent est alors mis en attente. La matrice le réveillera lorsque les ressources qu'il doit négocier seront libérées.

Notre modèle introduit des listes de priorité pour les participants, ce qui permet de choisir parmi différents contrats portant sur les mêmes ressources celui qui sera accepté. C'est pourquoi il se peut que des négociations soient en cours sur des ressources déjà prises car il est possible que l'initiateur du contrat en cours soit plus prioritaire pour l'agent que celui dont le contrat a été pris auparavant, et donc l'agent peut prendre ce nouveau contrat et se dédire du précédent.

Gestion simultanée

Lorsqu'une personne doit négocier de nombreux contrats, il peut être préférable d'en négocier plusieurs simultanément. Cela ne pose pas de problèmes lorsque les ressources mises en jeu dans les contrats sont différentes. En revanche, il peut être souhaitable de négocier séquentiellement les contrats portant sur des ressources communes. Notre modèle propose deux façons de gérer les négociations de contrats portant sur des ressources communes : la gestion séquentielle et la gestion en parallèle. L'utilisateur choisit celle qu'il préfère grâce au paramètre <*simultaneity value="deferred"*/> de son fichier XML. Lorsqu'il choisit la gestion en parallèle, aucune restriction n'est faite sur la disponibilité des ressources, elles peuvent déjà être en cours de négociation pour un autre contrat. Mais si l'utilisateur choisit la gestion séquentielle, le système utilisera le comportement de la matrice décrit précédemment.

La gestion des négociations de contrats portant sur des ressources différentes est toujours effectuée de façon parallèle. Ceci est rendu possible grâce à notre structure de micro-agents. Le fait d'avoir des micro-agents 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.

La figure 11 représente les différents cas qui peuvent se présenter pour la négociation de deux contrats ayant une ressource commune $r1$. Dans le cas a), on négocie les contrats en parallèle, tandis que dans le cas b), on les négocie séquentiellement. Le cas c) consiste à négocier le premier contrat qui porte sur la ressource $r1$ et à négocier en même temps la ressource $r2$ du deuxième contrat. Cette possibilité n'est pas offerte dans notre modèle, le contrat ne pouvant pas être divisé en sous-contrats contenant chacun une ressource.

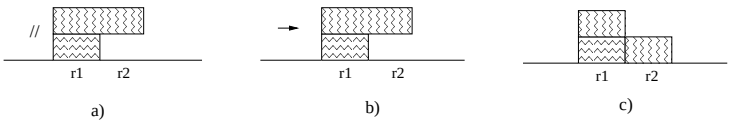

Figure 11. *Les différents cas de gestion des négociations portant sur des ressources communes. a) Tous les contrats sont négociés en parallèle. Aucune restriction n'est faite quant à la disponibilité des ressources. b) Les contrats sont négociés séquentiellement. c) Les contrats sont scindés en plusieurs parties (une par ressource). Chaque partie portant sur une ressource libre est négociée de suite. Les autres attendent la libération des ressources*

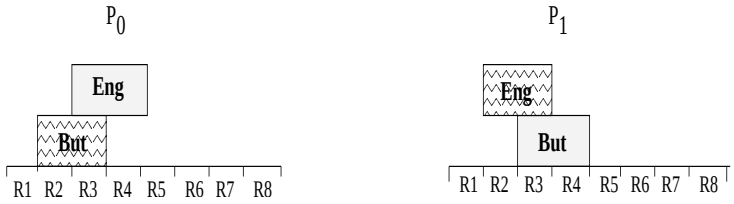

Figure 12. *Une situation de deadlock. Deux agents (P_0 et P_1) créent un contrat sur des ressources communes et le propose à l'autre. Lorsque la gestion séquentielle des contrats conflictuels est utilisée, chacun attend la réponse de l'autre avant de donner la sienne. Le deadlock est évité grâce au mécanisme de délai d'attente des réponses. Le contrat ayant le plus petit délai de réponse sera annulé, ce qui déclenchera la négociation de l'autre contrat*

Deadlocks

Un participant à une négociation peut être l'initiateur d'une autre négociation simultanée alors que les ressources sont partagées sans qu'il ne se produise un deadlock. En effet, les paramètres <*answer-delay*> et <*default-answer value="refuse"/*> du fichier de configuration XML permettent de limiter l'attente des réponses des participants et donc de débloquer une situation où un agent ne répondrait pas par exemple pour cause de crash.

Prenons l'exemple de deux agents : P_0 et P_1, qui créent simultanément un contrat sur une ressource identique et le proposent à l'autre (figure 12). Ceci constitue typiquement une situation de deadlock : chaque initiateur attend la réponse de l'autre. Cependant, dans notre système, le deadlock ne se produit pas. En effet, si l'on suppose que P_0 a défini un délai d'attente de 2 minutes et un refus par défaut, et que P_1 a défini un délai d'attente de 5 minutes et un refus par défaut, alors après deux minutes, P_0 considérera que P_1 refuse le contrat et l'annulera ce qui libérera la ressource $r1$. La ressource $r1$ étant à nouveau libre, P_0 entamera la négociation du contrat sur les ressources $r1$ et $r2$ avec P_1. Aucun blocage ne se sera donc produit.

6. Différentes applications réalisées avec GeNCA

Notre but est de proposer un modèle général pour négocier des contrats quels qu'ils soient. Ce modèle, que nous avons appelé GeNCA, a été implémenté en langage Java afin de fournir une API pour la création d'applications de négociation de contrats. Les diagrammes de classes de cette API sont présentés en annexe en figures 16,17,18,19, et 20. Nous présentons ici deux applications parmi celles que nous avons réalisées avec GeNCA. L'une est tout à fait classique, elle utilise des ressources individuelles aux participants, il s'agit d'une application de ventes aux enchères. L'autre est beaucoup moins classique, elle utilise des ressources communes

à tous les participants, il s'agit d'un système de prise de rendez-vous. D'autres applications ont également été réalisées, dont notamment une application de vente aux enchères hollandaises, un système de création d'emplois du temps ainsi qu'une application permettant aux agents de négocier le choix d'un restaurant pour une sortie commune. Ces applications sont décrites et peuvent être téléchargées à l'adresse suivante : *http ://www.lifl.fr/SMAC/projects/genca*.

6.1. *Utilisation du paquetage pour une application*

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

Les implémentations de la couche de communication que nous fournissons permettent d'utiliser les plateformes Magique et Madkit, ainsi que des agents centralisés parlant à tour de rôle. Nous fournissons également l'agent système auprès duquel les agents des utilisateurs s'enregistrent et qui est responsable de l'envoi des messages. Pour ces trois modes d'utilisation (Magique, Madkit et round-robin), une classe permettant de lancer l'application est fournie. Pour toute autre mode de communication (mail, etc.), il est nécessaire d'implémenter l'interface *Communicator* et d'avoir un agent intégrant le serveur de noms implémenté dans le paquetage. Les stratégies implémentées sont celles décrites dans la sous-section 4.2.

Le paquetage fournit également une interface graphique pour la négociation, qui permet de créer un contrat, de visualiser les différents messages envoyés et reçus par l'agent, de répondre à une proposition de contrat si l'utilisateur a choisi le mode manuel, de visualiser les contrats pris par l'agent, d'avoir une vue sur les négociations en cours sur les ressources et de se rétracter d'un contrat pris précédemment.

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

Afin de configurer l'application de négociation, et plus particulièrement les agents, deux types de fichiers XML sont utilisés : l'un est commun à tous les agents, il fournit la liste des ressources communes pour l'application, les paramètres du protocole, la gestion par défaut des négociations et les stratégies utilisées par défaut. Ce fichier est défini par le concepteur de l'application. L'autre est propre à chaque agent et optionnel. Il permet un utilisateur de redéfinir les valeurs par défaut des paramètres du fichier commun aux agents.

Pour écrire une application avec ce paquetage, il suffit donc d'implémenter les interfaces de communication et de stratégies qui sont spécifiques à l'application (si les implémentations par défaut ne conviennent pas), et de définir le fichier commun

de configuration en XML des agents et bien sûr écrire ses agents en leur incorporant le *négociateur* du paquetage. Lorsque l'application concerne la négociation de contrats portant uniquement sur les ressources, il n'y a rien d'autre à faire. Toute la gestion des négociations se fait automatiquement. En revanche, si le contrat nécessite d'autres paramètres (quantité, prix, etc), il faut alors étendre la classe *Contrat* et modifier l'interface graphique de création de contrat afin d'y faire apparaître les nouveaux paramètres.

6.2. *Un système de prise de rendez-vous*

Pour illustrer notre travail, nous décrivons ici une des applications nécessitant l'utilisation de la plupart de nos paramètres. Ce problème est une application de négociation pour la prise de rendez-vous. Chaque agent doit être capable de négocier des rendez-vous pour le compte de l'utilisateur. Chaque utilisateur possède un agenda avec des plages horaires libres ou non. Il possède de plus des préférences sur les heures et les personnes avec qui il peut prendre rendez-vous. Chaque utilisateur peut initier une demande de rendez-vous avec 1 ou plusieurs participants sur 1 ou plusieurs plages horaires. On s'interdit bien sûr de voir tous les agendas de chaque participant en transparence.

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

Pour cette application, les stratégies fournies dans le paquetage conviennent et il n'est donc pas nécessaire d'en écrire d'autres. Il n'y a donc que la communication à implémenter. Nous utilisons ici la plateforme Madkit pour faire tourner nos agents. Nous n'avons donc aucun code à écrire car le paquetage fournit une implémentation pour cette plateforme. Il ne reste à définir que le fichier de configuration système contenant les ressources, les stratégies des agents utilisées par défaut, et les valeurs des paramètres de la négociation. Ce fichier est présenté en figure 13. On y retrouve donc les tranches horaires d'une heure entre 8h et 18h et les stratégies fournies par GeNCA. Viennent ensuite les paramètres du protocole de négociation : le nombre de tours de parole des participants (20), le délai de réponse des participants (10 minutes), la réponse par défaut (refus), le nombre minimal d'accords nécessaires pour confirmer le contrat (100 %). La renégociation des contrats est autorisée, c'est d'ailleurs un besoin incontesté d'une application de prise de rendez-vous, le nombre de renégociations est fixé à 3. Pour cette application, on a choisi de négocier séquentiellement les contrats pour des rendez-vous au même moment.

```
<?xml version="1.0"?>
<!DOCTYPE genca SYSTEM "genca.dtd" >
<genca>
<application-name>rdv</application-name>
<resources-list>
    <resource>8h-9h</resource>
    <resource>9h-10h</resource>
    <resource>10h-11h</resource>
    <resource>11h-12h</resource>
    <resource>14h-15h</resource>
    <resource>15h-16h</resource>
    <resource>16h-17h</resource>
    <resource>17h-18h</resource>
</resources-list>
<default-initiator-strategy>
    fr.lifl.genca.strategy.DefaultInitiatorStrategy
</default-initiator-strategy>
<default-participant-strategy>
    fr.lifl.genca.strategy.DefaultParticipantStrategy
</default-participant-strategy>
<simultaneity value="deferred"/>
<protocol>
    <answer-delay>10</answer-delay>
    <default-answer value="refuse"/>
    <minAgreements>100%</minAgreements>
    <nbRounds>20</nbRounds>
    <nb-modifications-by-round>5</nb-modifications-by-round>
    <retraction-allowed value="true"/>
    <nbRenegotiations>3</nbRenegotiations>
</protocol>
</genca>
```

Figure 13. *Fichier XML commun de configuration des agents pour la prise de rendez-vous On y retrouve le type d'application réalisée (rdv), la liste des ressources qui seront négociées (les tranches horaires), les stratégies utilisées par l'agent et les valeurs des différents paramètres du protocole de négociation.*

La figure 14 montre l'interface de création d'un contrat pour la prise de rendez-vous. L'initiateur y indique les tranches horaires souhaitées, les participants conviés au rendez-vous, le délai d'attente des réponses des participants, la réponse qu'il considère par défaut, le nombre de tours dans la négociation et le nombre de renégociations autorisées.

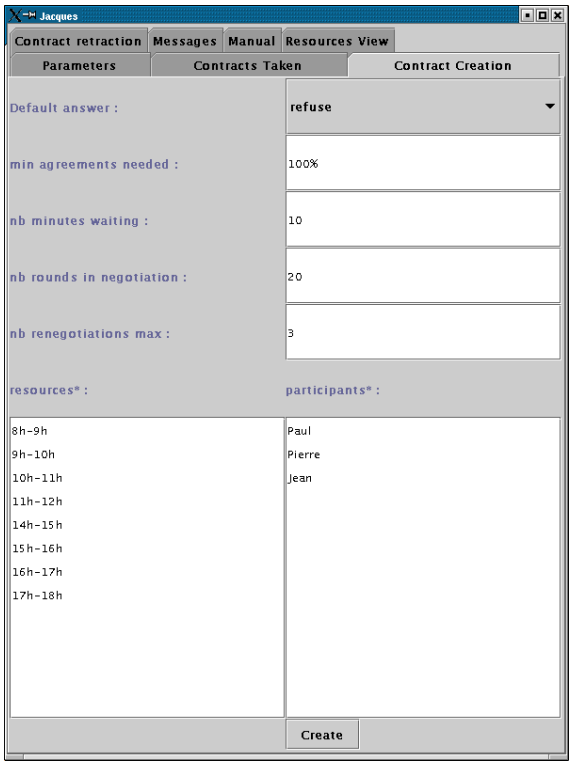

Figure 14. *Interface de création d'un contrat pour la prise de rendez-vous avec GeNCA. L'initiateur choisit les ressources qu'il veut négocier, les participants avec qui il souhaite prendre rendez-vous ainsi que les valeurs des paramètres de la négociation : nombre minimal de personnes pour prendre le rendez-vous, délai d'attente des réponses, réponse par défaut, nombre de tours dans la négociation et nombre de déplacements du rendez-vous possibles*

Cette application permet aux agents de négocier des rendez-vous pour le compte de leur utilisateur. Contrairement à d'autres systèmes que l'on trouve dans le commerce, les agendas des utilisateurs sont privés et le problème ne consite pas simplement à trouver un créneau horaire libre et commun à tous les participants, connaissant leurs agendas, mais à négocier l'heure du rendez-vous en tenant compte des préférences des utilisateurs sur les horaires et les personnes. De plus, ce système renégocie automatiquement un rendez-vous qui doit être déplacé suite aux désistements des participants.

6.3. *Un système de ventes aux enchères*

Dans cette application de vente aux enchères, chaque agent doit être capable négocier des enchères pour l'utilisateur. Pour cela, chaque utilisateur définit une somme d'argent (son crédit), et une stratégie d'enchère (linéaire, quadratique, etc).

L'enchère se déroule de la façon suivante : un vendeur propose un article pour lequel il décide d'obtenir un prix minimal qu'il garde secret. Ensuite, les acheteurs lui font savoir s'ils sont intérressés ou non par cet article, et s'ils le sont lui proposent un prix pour celui-ci. L'acheteur retient le prix le plus élevé proposé et si ce prix est supérieur au prix minimal qu'il avait défini, l'acheteur ayant proposé ce prix gagne l'enchère. Si le meilleur prix proposé n'atteint pas le prix minimal requis par le vendeur, celui-ci propose à nouveau son article aux acheteurs intéressés afin qu'ils proposent un prix supérieur. Ceci est répété jusqu'à ce qu'un acheteur gagne l'enchère ou que le nombre de tours défini est atteint. Ce type d'enchères s'approche des offres scellées au meilleur prix qui se déroulent sur plusieurs tours à la façon des enchères anglaises.

Pour cette application, la rétractation n'est pas autorisée, une fois l'article vendu il l'est définitivement.

Cette application fait intervenir un paramètre supplémentaire dans la négociation : un prix. Les stratégies par défaut ne sont donc plus utilisables, il faut en concevoir d'autres. Le prix n'est pas un paramètre du contrat, il ne faut donc pas modifier cette classe. En revanche, il faut rajouter dans les *propriétés du contrat* le prix de réservation du vendeur. Il faut également adapter l'interface de saisie de contrat afin de pouvoir entrer ce prix et l'interface de proposition en mode manuel afin que le participant puisse entrer une enchère. Il faut également étendre le *négociateur* afin de gérer les achats et ventes de ressources ainsi que le porte-monnaie de l'utilisateur. Par contre, si l'on utilise la plateforme Magique, aucun travail n'est nécessaire pour la couche de communication. En effet, comme nous l'avons mentionné en sous-section 6.1, notre API fournit différentes instanciations de la couche communication dont une pour l'utilisation de la plateforme Magique.

6.3.1. *Stratégie de l'initiateur*

L'initiateur commence par créer un contrat via l'interface graphique comme pour l'application de prise de rendez-vous. Dans le cas des enchères, un paramètre supplémentaire doit être indiqué : le prix de réservation.

Lorsqu'un accord est reçu, la stratégie met à jour le prix le plus élevé proposé jusqu'alors. Si le nouveau prix est plus élevé, ce prix devient le prix maximal proposé et l'acheteur l'ayant proposé le vainqueur courant de l'enchère. Une fois toutes les réponses reçues, l'initiateur décide de *confirmer* l'enchère auprès du participant ayant proposé le prix maximal si ce prix dépasse le prix de réservation du vendeur, et donc d'*annuler* l'enchère auprès des autres participants. Si le prix de réservation n'est pas

atteint ni le nombre de tours, l'initiateur *demande une modification* aux participants intéressés. Dans les autres cas, il *annule* l'enchère.

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

6.3.2. *Stratégie du participant*

Lorsqu'un participant reçoit une *proposition* d'enchère, il regarde si l'article proposé l'intéresse ou non. Si c'est le cas, il *accepte* le contrat et propose un prix. Sinon, il *refuse*.

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

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

La figure 15 montre les interfaces de 4 agents négociant des enchères avec notre API.

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

Les avantages de cette application sont nombreuses, nous allons donc citer ici les plus importantes. Premièrement, cette application aide l'utilisateur à enchérir, et enchérit à sa place quand il n'est pas là, selon la stratégie qu'il a défini. Deuxièmement, cette application peut facilement être étendue à un autre type d'enchères, comme les enchères anglaises, hollandaises, vickrey, etc. Et troisièmement, cette application est portable, en effet, les agents peuvent être placés sur un réseau hétérogène, sur des PDAs, etc.

Ces deux exemples montrent que notre protocole s'applique à différents types d'applications de négociation comme les enchères ou la prise de rendez-vous. Cela illustre notre propos de protocole général. Dans la section suivante, nous allons comparer notre protocole avec différentes applications réalisées par d'autres chercheurs pour montrer les différences qui existent entre eux.

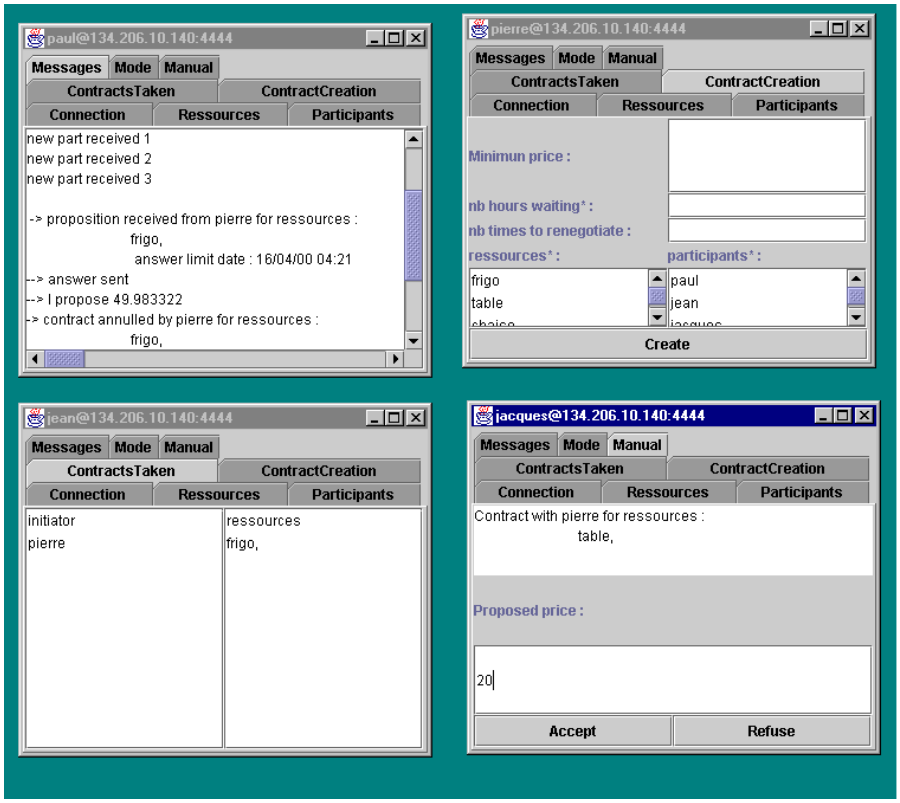

Figure 15. *Quatre agents participant à l'application de vente aux enchères. En haut et à gauche, l'agent Paul visualisant ses messages. En bas et à gauche, l'agent Jean visualisant ses contrats pris, ici, les articles qu'il a achetés. En haut et à droite, la fenêtre de création de contrat de l'agent Pierre. En bas et à droite, l'agent Jacques répondant manuellement à la proposition de Pierre pour vendre une table. Jacques propose ici 20 euros*

7. Comparaison avec d'autres modèles

Nous ne sommes évidemment pas les seuls ayant pour objectif la négociation entre agents et la proposition d'une architecture générale pour l'accomplir.

7.1. *Les travaux sur la négociation aux laboratoires HP*

Claudio Bartolini et ses collègues((Bartolini *et al.*, 2001; Bartolini *et al.*, 2002b; Bartolini *et al.*, 2002a)) veulent créer un framework général pour la négociation au-

tomatique dédié aux mécanismes de marché. Ce framework comporte un *protocole général* de négociation qui se paramètre par des *règles de négociation*. En choisissant différents ensembles de règles, différents mécanismes de négociation peuvent être implémentés. Il y a deux rôles principaux dans la négociation : *participant* et *hôte*. Un participant est un agent qui veut atteindre un accord, tandis que l'hôte de la négociation est responsable de faire respecter le protocole et les règles de négociation. Chaque participant peut envoyer des propositions par un message destiné à l'hôte de négociation. Celui-ci détermine quels participants doivent voir le message et le multicaste de façon appropriée. Cela permet de modéliser la négociation de 1 vers 1 comme un cas particulier de la négociation de n vers m. Afin de négocier, les participants doivent initialement partager un *template* de négociation. Il spécifie les différents paramètres de négociation qui peuvent être soit contraints, soit totalement ouverts. Le processus de négociation consiste à parvenir à un accord acceptable depuis le template de négociation. Au cours de la négociation, les participants échangent des *propositions* représentant des accords actuellement acceptables pour eux. Chaque proposition contient des contraintes sur tout ou partie des paramètres du template de négociation. Ces propositions sont envoyées à l'hôte de négociation.

Ce framework a été implémenté en utilisant la plateforme Jade, les agents communiquent par envoi de messages au format FIPA ACL, et Jess a été utilisé pour le traitement des règles.Ce framework est proche du notre par son protocole général qui est paramétrable ici par des règles, et qui fournit un mécanisme de contre-propositions. Cependant, notre protocole permet en plus de se rétracter d'un contrat et de le renégocier automatiquement. De plus, nous ne sommes pas liés à une plateforme multi-agents car nous avons séparé notre couche de communication afin de pouvoir s'adapter à différents moyens de communication comme par exemple l'envoi d'e-mails.

7.2. *La plateforme de négociation générique pour les places de marché : GNP*

La Generic Negotiation Platform (GNP) est réalisée par Morad Benyoucef, Rudolf Keller et leurs collègues ((Benyoucef *et al.*, 2000)) au département IRO de l'université de Montréal. C'est une plateforme générique de négociation d'entreprises à entreprises orientée enchères pour les places de marché. La négociation entre les agents s'effectue via le service de négociation qui s'occupe de l'appariement entre la demande des acheteurs, l'offre en bien et services des vendeurs et la fixation des prix. Cette plateforme reçoit les annonces des acheteurs ou des vendeurs et les mises qui répondent à ces annonces. Les règles de marché sont définies dans des fichiers XML. L'initialisation d'une négociation sur GNP se fait par l'envoi d'un document XML initial : l'annonce, qui contient tous les éléments d'entrée nécessaires au démarrage d'une négociation. L'annonce apparaît à l'utilisateur dans un format HTML. Elle contient toujours une partie *règles* (rules) et, selon le mécanisme choisi, une partie *produit* (productReference), et une partie *ordre* (order). La partie *règles* établit précisément les paramètres fixes de la négociation, telles que les heures d'ouvertures et de fermetures,

l'incrément minimum, les conditions de fin de rondes et les conditions d'équilibrage du marché. Si ceux-ci doivent varier, la négociation est divisée en plusieurs phases. La partie *produit* définit techniquement la nature du produit à négocier, renvoit à sa description précise. La partie *ordre* représente les données de la mise initiale faite par le participant, telles que son prix de départ, son prix de réserve, la quantité de produit mise en négociation et ses fractions permises. Lorsque l'annonce est remplie, elle est envoyée au système qui va déclencher une session d'annonce. Les utilisateurs interagissent avec GNP uniquement via un navigateur web. Aucune installation de logiciel ne doit être effectuée.

D'un point de vue formel, les auteurs

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

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

7.3. *Le projet SilkRoad*

Un troisième travail est le projet SilkRoad (Ströbel, 2001) réalisé au laboratoire de recherche d'IBM à Zurich. Ce projet a pour but de faciliter la conception et l'implémentation de systèmes de support de négociation pour des domaines d'applications spécifiques. SilkRoad facilite les négociations multi-attributs dans les scénarios d'e-business grâce à une méthodologie de conception spécifique et à une architecture

de système générique avec des composants de support de négociation réutilisables. Un système de support de négociation construit sur la base du modèle d'architecture SilkRoad agit comme un intermédiaire entre les agents négociant réellement (qui peuvent être des agents humains ou logiciels) et ainsi fournit une communication basée sur des règles et un support de décision.

Les deux éléments au coeur de SilkRoad sont le `ROADMAP` et le `SKELETON`. Le `SKELETON` fournit plusieurs composants de service de négociation modulables et configurables, qui peuvent être utilisés pour implémenter un média de négociation électronique. Ces composants de service sont : *match* (appariement), *score* (classement), *mediate* (médiation), *bid* (gestion des offres), *contract* (finalisation de l'accord) et *bundle* (cumulation d'offres).

Le projet SilkRoad se concentre sur les négociations commerciales où sont échangées des offres de vente et des offres d'achat. Bien que le scénario d'une négociation puisse être spécifié par l'utilisateur, celui-ci reste relatif aux offres et il n'est pas possible de définir ses propres actions, seules celles fournies par le `SKELETON` peuvent être utilisées. La généricité de cette plateforme est donc relative, puisqu'elle est cantonnée aux fonctionalités fournies par celle-ci. Notre aproche de la négociation et plus particulièrement d'une plateforme générique de négociation est plus ouverte. Pour nous, il doit être possible d'effectuer des négociation commerciales sous différentes formes (enchères, etc) aussi bien que des négociations non-commerciales (prise de rendez-vous, etc). Dans SilkRoad, il n'y a pas, a priori, de communication entre les agents : un serveur regroupe les offres et s'occupe de les apparier. Notre aproche est de fournir aux agents un cadre de négociation au sein duquel ils peuvent s'échanger des propositions de contrat. Chaque agent gère donc ses négociations, et il est possible de définir de nouvelles stratégies de négociation.

Ces trois travaux sont proches des nôtres, mais ils sont plus dirigés vers le commerce électronique tandis que notre modèle a pour but de convenir aussi à d'autres types de négociation automatiques.

7.4. *La plateforme de négociation Magnet*

Examinons maintenant la plateforme de négociation Magnet (Multi AGent NEgotiation Testbed). Magnet (Collins *et al.*, 1998) est un banc de tests pour la négociation multi-agents, implémenté sous la forme d'une architecture généralisée de marché et développé à l'université du Minnesotta. Il fournit un support pour une variété de types de transactions, du simple achat/vente de biens à la complexe négociation de contrats entre agents. Un mécanisme de session permet à un consommateur de lancer un appel à propositions et de conduire une autre affaire en même temps. Le protocole de négociation pour la planification par la prise de contrats consiste en trois phases : un appel à propositions, une récolte des propositions et un choix de propositions. Dans Magnet, il y a un intermédaire explicite dans le processus de négociation et les agents

interagissent entre eux par son intermédiaire. Magnet s'intéresse plus particulièrement aux mécanismes de marché, à l'exécution de plans et aux chaînes de productions. Elle offre des outils de mesure des performances de recherche et de qualité des solutions trouvées, et s'investit dans la proposition de statégies de détermination du gagnant dans des enchères combinées. Son principal but est de fournir une plateforme de simulation de gestion de chaines de production qui permet d'évaluer différentes stratégies.

Contrairement à celui proposé dans Magnet, notre protocole permet à l'initiateur de l'appel à propositions de faire des contre-propositions jusqu'à ce qu'un accord soit atteint. Ceci permet d'aboutir plus rapidement à un accord et de prendre en compte les souhaits des participants. Un autre point fort de GeNCA est que les agents interagissent directement entre eux dans notre processus de négociation, il n'y a pas d'intermédiaire explicite comme dans Magnet. Les agents initiateurs sont donc maîtres de leur négociation. La rétractation est possible moyennant une pénalité à payer, mais la renégociation ne se fait automatiquement, alors que c'est le cas dans notre modèle.

7.5. *La plateforme Zeus*

Zeus (Nwana *et al.*, 1999) est une API Java générique réalisée par British Telecom dans le but de concevoir facilement des applications de négociation entre agents automones basées sur la notion de coût. Zeus propose un protocole de négociation entre deux agents uniquement (un initiateur et un participant) et sur une unique ressource par contrat. Le protocole est constitué d'un appel d'offres et aucun mécanisme de contre-proposition n'est fourni. De plus, il est possible de négocier simultanément différents contrats sur des ressources identiques. La rétractation n'est pas possible avec Zeus. Une fois un contrat pris, il est impossible de se rétracter. De plus, Zeus fournit uniquement des stratégies basées sur les coûts.

GeNCA permet à plusieurs agents (1 initiateur et n participants) de négocier plusieurs ressources au sein d'un contrat, ce qu'empêche Zeus, alors que de nombreux types de négociation en ont besoin. De plus, Zeus se limite au contract-net, et donc aucun mécanisme de contre-proposition n'est proposé aux agents qui ne peuvent réellement participer à la négociation, puisqu'ils ne peuvent pas donner leur avis. Ce mécanisme de contre-proposition est inclus dans notre protocole de négociation. Notre protocole permet également aux agents de se rétracter d'un contrat qu'ils avaient précédemment pris. Les agents peuvent donc changer d'avis au cours du temps, ce qui n'est pas le cas dans Zeus.

Plateforme	cardinalité	généricité	rétraction	renégociation automatique	simultanéité
GeNCA	$n \rightarrow m$	oui	oui	oui	oui
GNP	$1 \rightarrow 1$	en partie	non	non	oui
SilkRoad	$1 \rightarrow 1$	en partie	?	?	oui
Magnet	$1 \rightarrow m$	oui	oui	non	oui
HP	$n \rightarrow m$	oui	non	non	oui
Zeus	$1 \rightarrow 1$	non	non	non	oui

Plateforme	séparation de la communication	contre-propositions	réponse par défaut	paramètrage possible
GeNCA	oui	oui	au choix	oui
GNP	non	oui	refus implicite	oui
SilkRoad	non	oui	refus implicite	non
Magnet	non	non	refus implicite	non
HP	non	oui	non	oui
Zeus	non	non	non	non

Tableau 2. *Tableau comparatif des plateformes de négociation. Le* ? *signifie qu'aucune information ne nous permet d'affirmer ou d'infirmer la propriété énoncée*

Il en existe encore bien d'autres comme Kasbah, AuctionBot, Fishmarket ou Jasa (Java Auction Simulator API) (, *http ://www.csc.liv.ac.uk/ sphelps/jasa/*, n.d.), qui permettent de créer des applications de ventes aux enchères uniquement. Ces précédents travaux, comme les nôtres, se basent sur le modèle général du Contract Net Protocol (Smith, 1980) qui fonctionne sur le modèle d'un appel à propositions d'un agent manageur et des agents contractants. De tous ces travaux, Magnet (pour la plateforme) et les travaux de Morad Benyoucef (pour l'aspect formel) sont probablement les plus proches de ce que nous présentons. Néanmoins, aucun d'eux ne prend en compte à la fois les aspects génériques, la renégociation automatique et un mécanisme de gestion des conflits entre des négociations simultanées, que nous proposons dans GeNCA. De plus, GeNCA est la seule plateforme qui sépare les niveaux de communication, de négociation et de stratégie. Le tableau 2 reprend les différentes plateformes présentées ainsi que GeNCA et apporte une comparaison sur différents critères qui nous paraissent importants pour une plateforme de négociation générique.

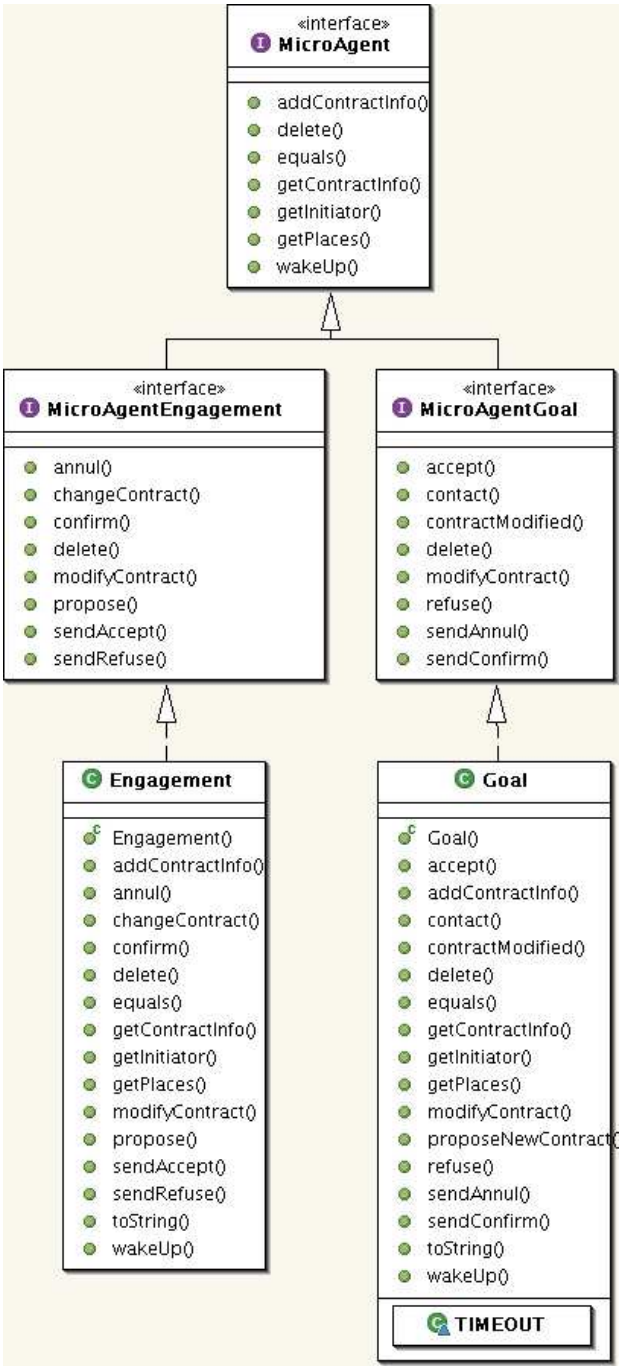

Figure 16. *Diagramme de classes de la couche négociation - Partie 1*

Figure 17. *Diagramme de classes de la couche négociation - Partie 2*

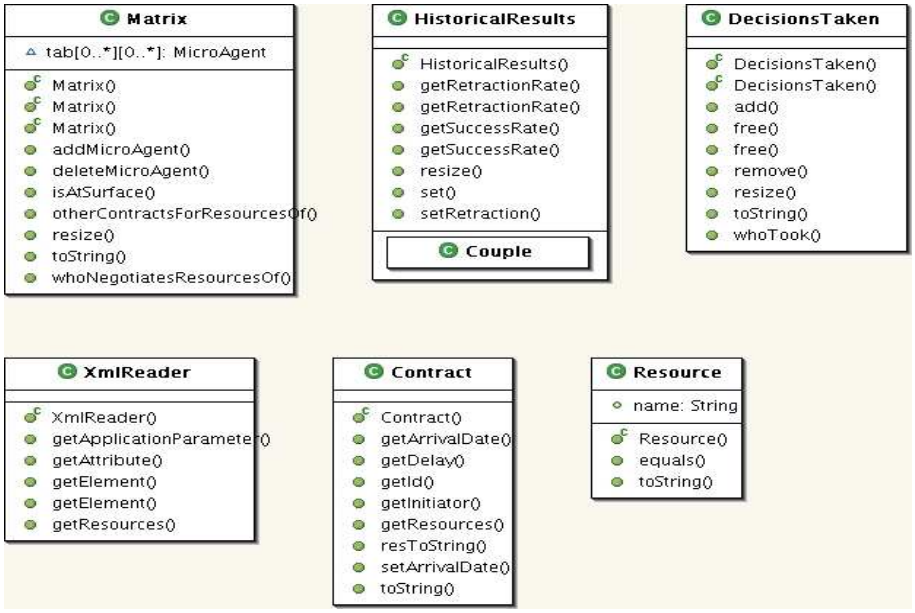

Figure 18. *Diagramme de classes de la couche négociation - Partie 3*

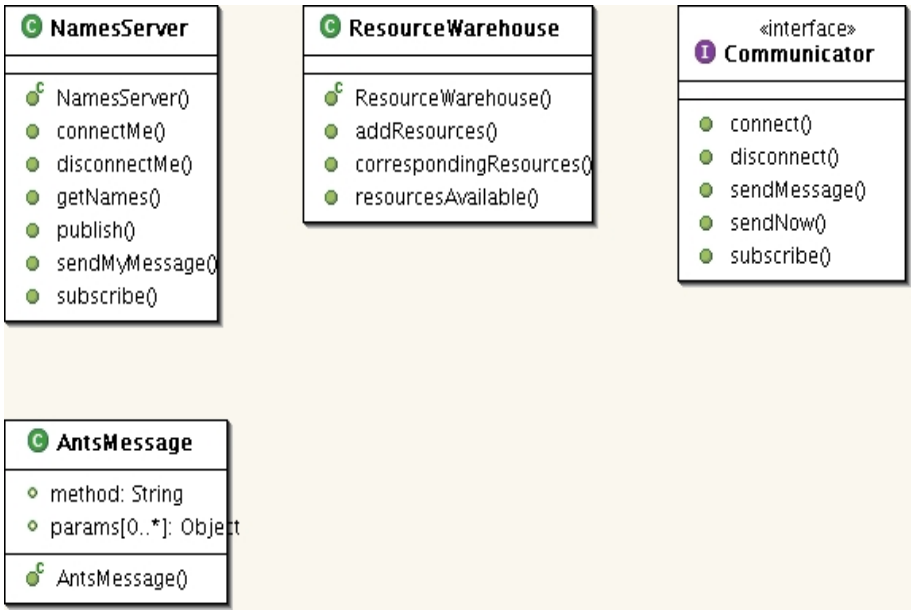

Figure 19. *Diagramme de classes de la couche communication*

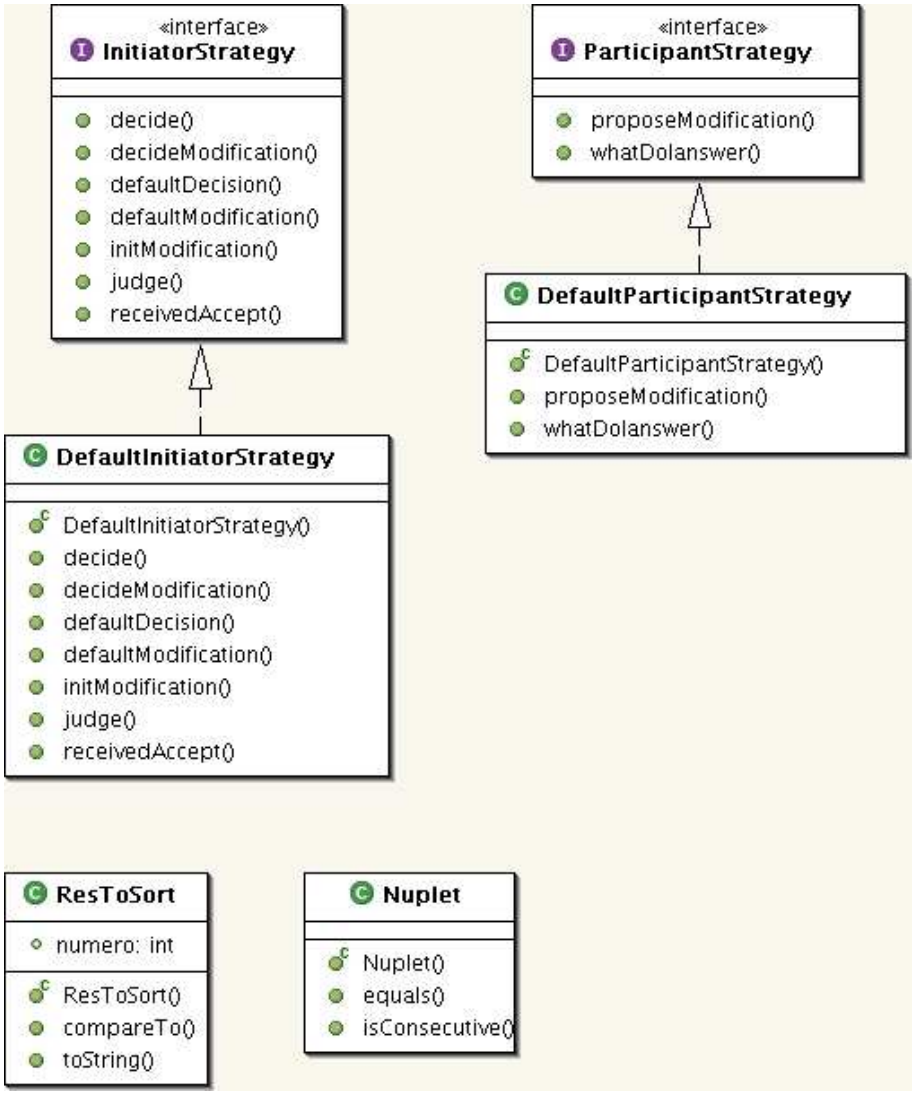

Figure 20. *Diagramme de classes de la couche stratégie*

8. Conclusion et perspectives

Dans cet article, nous avons défini un modèle général pour la négociation de contrats (GeNCA), et son implémentation par une API Java. Notre modèle s'appuie sur trois niveaux indépendants les uns des autres : le niveau de négociation, le niveau de communication et le niveau de stratégie. Chaque niveau peut facilement être étendu par le développeur de façon à s'adapter à son application. Le niveau de négociation est le coeur de notre modèle, il inclut le protocole de négociation et les structures de données nécessaires à la réalisation d'une négociation. Le protocole se paramètre via un fichier XML afin de s'adapter à de nombreuses applications de négociations. Les deux autres niveaux qui sont spécifiques à une application fixée possèdent des implémentations par défaut afin de pouvoir utiliser le système immédiatement. Cette API permet la négociation de n vers m agents, la négociation simultanée de plusieurs contrats, et la gestion des deadlocks dans la conversation. Ces travaux font partie du génie logiciel et des travaux sur l'intelligence artificielle distribuée, avec les mêmes objetifs que la plateforme GNP de Montréal. La première perspective que nous avons est d'extraire le protocole du modèle afin de pouvoir facilement en changer, en le spécifiant dans un fichier XML, par exemple. De nombreuses perspectives d'implémentation de ces travaux sur différents supports software sont possibles (distribué, centralisé, WEB) et l'amélioration du niveau stratégique pour différents types de problèmes spécifiques est considéré. Nous tentons maintenant d'appliquer cette API à différents problèmes comme l'enseignement à distance, les jeux en réseau, et les systèmes de workflow.

9. Bibliographie

, *http ://sharon.cselt.it/projects/jade/*n.d.

, *http ://www.csc.liv.ac.uk/ sphelps/jasa/*n.d.

, *http ://www.fipa.org*n.d.

, *http ://www.lifl.fr/SMAC/projects/magique*n.d.

, *http ://www.madkit.org*n.d.

Aknine S., « New Multi-Agent Protocols for M-N-P Negotiations in Electronic Commerce », *National Conference on Artificial Intelligence, AAAI, Agent-Based Technologies for B2B Workshop*, Edmonton, Canada, July, 2002.

Bartolini C., Preist C., A Framework for Automated Negotiation, Technical Report n° HPL-2001-90, HP Laboratories Bristol, 2001.

Bartolini C., Preist C., Jennings N. R., « Architecting for reuse : A software framework for automated negotiation », *Proc. 3rd Int Workshop on Agent-Oriented Software Engineering*, Bologna, Italy, p. 87-98, 2002a.

Bartolini C., Preist C., Jennings N. R., A Generic software framework for automated negotiation, Technical Report n° HPL-2002-2, HP Laboratories Bristol, 2002b.

Bensaid N., MAGIQUE, une architecture multi-agents hiérarchique, Thèse de Doctorat, Université de Lille 1, Mai, 1999.

Benyoucef M., Keller R. K., Lamouroux S., Robert J., Trussart V., « Towards a Generic E-Negotiation Platform », *Proceedings of the Sixth International Conference on Re-Technologies for Information Systems*, Zurich, Switzerland, p. 95-109, 2000.

Chavez A., Maes P., « Kasbah : An Agent Marketplace for Buying and Selling Goods », *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK, April, 1996.

Collins J., Youngdahl B., Jamison S., Mobasher B., Gini M., « A Market Architecture for Multi-Agent Contracting », *2nd Int'l Conf on Autonomous Agents*, Minneapolis, p. 285-292, May, 1998.

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

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

Guttman R. H., Maes P., « Cooperative vs. Competitive Multi-Agent Negotiations in Retail Electronic Commerce », *Proceedings of the Second International Workshop on Cooperative Information Agents (CIA'98)*, Paris, France, July, 1998.

Jennings N. R., Faratin P., Lomuscio A. R., Parsons S., Sierra C., Wooldridge M., « Automated Negotiation : Prospects, Methods and Challenges », *Int. Journal of Group Decision and Negotiation*, vol. 10, n° 2, p. 199-215, 2000a. (to appear).

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

Klemperer P., « Auction Theory : A Guide to the Literature », *Journal of Economics Surveys*, 1999.

Kraus S., *Strategic Negotiation in Multiagent Environments*, MIT Press, 2001.

Kraus S., Sycara K., Evenchik A., « Reaching agreements through argumentation : a logical model and implementation », *Artificial Intelligence*, vol. 104, p. 1-69, 1998.

Mathieu P., Verrons M.-H., « A generic model for contract negotiation », *Proceedings of the AISB'02 Convention*, London, UK, p. 1-8, 2002.

Mathieu P., Verrons M.-H., « ANTS : an API for creating negotiation applications », *Proceedings of the 10th ISPE International Conference on Concurrent Engineering : Research and Applications (CE2003), track on Agents and Multi-agent systems*, Madeira Island, Portugal, p. 169-176, july 26-30, 2003a.

Mathieu P., Verrons M.-H., « A Generic Negotiation Model for MAS using XML », *Proceedings of the ABA workshop Agent-based Systems for Autonomous Processing, held by the IEEE International Conference on Systems, Man and Cybernetics.*, IEEE Press, Washington, USA, p. 4262-4267, October, 5-8, 2003b.

Meyer R., Conry S., Lesser V., « Multistage negotiation in distributed planning », *in* , A. Bond, , L. Gasser (eds), *Reading In Distributed Artificial Intelligence*, Morgan Kaufmann Publishers, Los Altos, CA, p. 367-386, 1988.

Nwana H. S., Ndumu D. T., Lee L. C., Collis J. C., « ZEUS : a toolkit and approach for building distributed multi-agent systems », *in* , O. Etzioni, , J. P. Müller, , J. M. Bradshaw (eds), *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, ACM Press, Seattle, WA, USA, p. 360-361, 1999.

Parsons S., Jennings N., « Negotiation through argumentation - a preliminary report », *Proc. Second Int. Conf. on Multi-Agent Systems*, Kyoto, Japan, p. 267-274, 1996.

Rosenschein J., Zlotkin G., *Rules of encounter : designing conventions for automated negotiation among computers*, MIT Press, Cambridge, Mass., 1994.

Sandholm T., « Algorithm for Optimal Winner Determination in Combinatorial Auctions », *Artificial Intelligence*, vol. 135, p. 1-54, 2002.

Sandholm T., Lesser V., « Issues in Automated Negotiation and Electronic Commerce : Extending the Contract Net Framework », *First International Conference on Multiagent Systems (ICMAS-95)*, San Francisco, p. 328-335, 1995. (Acceptance rate 33%).

Smith R. G., « The Contract Net Protocol : high-level communication and control in a distributed problem solver », *IEEE Transactions on computers*, vol. C-29, n° 12, p. 1104-1113, December, 1980.

Ströbel M., « Design of Roles and Protocols for Electronic Negotiations », *Electronic Commerce Research, Special Issue on Market Design*, vol. 1, n° 3, p. 335-353, 2001.

Walton D., Krabbe E., *Commitment in Dialogue*, SUNY Press, 1995.