AISB Journal **1**(6), © SSAISB, 2005
http://www.aisb.org.uk

A generic negotiation model using XML

Philippe Mathieu and Marie-Hélène Verrons

Equipe SMAC, LIFL, USTL
Cité Scientifique, Bat M3, 59650 Villeneuve d'Ascq, FRANCE
mathieu@lifl.fr ; verrons@lifl.fr

Abstract

In this paper, we present a generic negotiation model for multi-agent systems called GeNCA, built on three levels: a communication level, a negotiation level and a strategic level, which is the only level specific to a particular application. XML files are used to configure both each agent and the global system, freeing the end-user from the need to reconfigure the system each time they want to change a parameter. The aim of this paper is then to show that it is possible to give a precise description of a generic negotiation model that we can use in several real problems. This model has been implemented with a Java API 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.

1 Introduction

With the progress of information technology, multi-agent systems and electronic market places, the need of automatic agents able to negotiate with the others on behalf of the user becomes stronger and stronger. Moreover, the utility of using an agent during negotiations is perfectly justified by the explosion of the number of messages exchanged between agents. In certain cases, specially with cascaded renegotiations, the number of messages can be in $O(m^n)$ if n is the depth of the cascaded process and m the number of agents involved in one negotiation.

Since several years, negotiation has been studied by many researchers ((Rosenschein and Zlotkin, 1994; Sykara, 1989; Schwartz and Kraus, 1997)), and many negotiation systems have been achieved in specific domains like auctions or market places often in the aim of electronic commerce, let's cite Zeus (Nwana et al.,) developed by British Telecommunications, Magnet (Collins et al., 1998b) developed by the university of Minnesota, the SilkRoad project (Ströbel, 2001) developed by IBM, the platform GNP (Benyoucef et al., 2000) developed at the Montreal university and works done at HP Laboratories (Bartolini and Preist, 2001). Of course, negotiation can be used in other domains like meeting scheduling or reservation systems, but it seems that these ways have not been really studied. When studying such negotiation problems, we can see that many used notions are the same in many systems. For example, `contracts, resources, contractors (initiators), participants` have a semantic equivalent in all negotiation systems. Our aim in the software engineering field, is to show that these notions can be reified

in a generic and open negotiation model and to build the corresponding API. The model we propose here is broad enough to allow classical negotiation applications to be covered without an adaptation effort, and has enough parameters to adapt to different negotiation applications, which is a difficult engineering problem.

Although it is difficult to define formally what is negotiation, we will base our arguments on the following consensual definition (Smith, 1980; Jennings et al., 2000; Walton and Krabbe, 1995), which can be applied to many fields such as auctions, appointment taking systems, games or others.

definition : Negotiation is carried out on a *contract* to obtain common *resources* and on the request of an *initiator*. It brings together a set of *participants* and an *initiator* and runs until an agreement satisfying a percentage of participants is reached. Participants equally try to obtain the best possible solution for themselves while giving a minimum set of information to the others.

definition : A contract is the entity which will be negotiated. It contains the *initiator* of the negotiation, the *resources* involved, the *answer delay* and a *default answer* in the case where a participant wouldn't have answered at time.

This definition is of course inspired of the Contract Net Protocol proposed by Smith (Smith, 1980) in 1980, which is a fundamental of many negotiation works (Sandholm, 2000). The main differences with the Contract Net is that negotiation ends with a contract between the initiator and several participants after possible rounds of proposals and counter-proposals. The initiator is the equivalent of the manager of the Contract Net and is in fact the first person who talk in the negotiation process. In the context of our study, we consider that a minimum number of information must be revealed to other agents, because when all information is known, we fall in a problem solver context, where algorithms such as a CSP is more fitted.

To conceive our model and allow a real generality, we have chosen a three-level architecture as a basis. The internal level which contains the management of data structures and speech acts necessary for agents to evolve their knowledge; the communication level allowing agents to send messages in a centralised way if agents are on the same computer, or in a distributed way if they are on different computers; the strategic level allowing agents to reason on the problem and infer on the knowledge obtained from the others. In our work, each level can be changed independently of the others. It is for example possible to use GeNCA in a round robin way with synchronous communication with all agents on the same computer to realise a video game where virtual beings will negotiate turn to turn, and to use it in a distributed way with asynchronous communication for electronic marketplace. In our model, the negotiating agent is composed of reactive micro-agents, where each micro-agent manages a negotiation.

The success of a negotiation depends of course on strategies adapted to the problem processed. We will not discuss here about strategies, which, to be optimal, must be different according to the kind of negotiation done. This is an important field which goes out of this paper. Therefore, we propose simple but generic strategies, which work for several kinds of problems, and that the user can easily refine.

We have identify many criteria to describe a negotiation, where we can find the number of rounds in a negotiation process, the minimum number of agreements needed to confirm the contract, the retraction possibility, or the answer delay. Many of them have been taken into account to build GeNCA.

A human user has two ways to use his agent. Manually, it is then a help-decision tool which shows the state of all the concurrent negotiations. In such case, it is the human user

who agrees a query. Automatically, this time the agent is hidden and proposes or answers queries by itself.

In GeNCA, the general server has an XML configuration file which allows to define the general notions like retraction possibility or the number of rounds in a negotiation process. Each agent can also have his own XML file to define the parameters of his owner (minimum number of agreements needed to confirm the contract, answer delay, etc.). Having XML files to configure the system makes it easier for the user to define a negotiation problem.

In this paper, we will first detail the protocol used (the phases of the protocol, the communication primitives and its properties). Then, we will describe GeNCA and the different ways to use it. After this, we detail two applications realised with GeNCA. Finally, we compare our works to others achieved on the same subject.

2 Proposed protocol

The protocol we propose here aims to define the messages that agents can send to each others with the operational dynamics associated. This negotiation protocol (Figure 1) is characterised by successive messages exchanged between an initiator (the agent who initiates the negotiation) and participants (the agents who participate to the negotiation) as in the Contract Net Protocol framework (Smith, 1980). We first describe the phases that compose our negotiation protocol, and then the communication primitives between agents used in this protocol. Finally we give characteristics of our negotiation protocol.

2.1 Protocol phases

We distinguish three phases for a negotiation process : the first one is the proposal phase which begins the negotiation process. Then, there is an optional phase named conversation phase. This phase consists of rounds of proposals and counter-proposals in order to converge to an acceptable contract for everyone. Finally, there is the final decision phase where the contract is either confirmed, either cancelled.

Proposal phase In this phase, the initiator proposes a contract to participants and waits for their answer. In response to the proposal, each participant answers if he agrees or rejects it.

Conversation phase This phase is necessary if there was not enough participants who agreed the contract proposal. A conversation is then started between the initiator and participants during which modification proposals are exchanged. Following these proposals, the initiator proposes a new contract to participants, and a new proposal phase is entered.

Final decision phase This final decision phase comes to either a confirmation or a cancellation of the contract. This decision is taken by the initiator in response to participants' answers.

2.2 Negotiation primitives

To carry out a negotiation process between agents, it is necessary to define several negotiation primitives between agents. We thus need specific primitives for initiators and

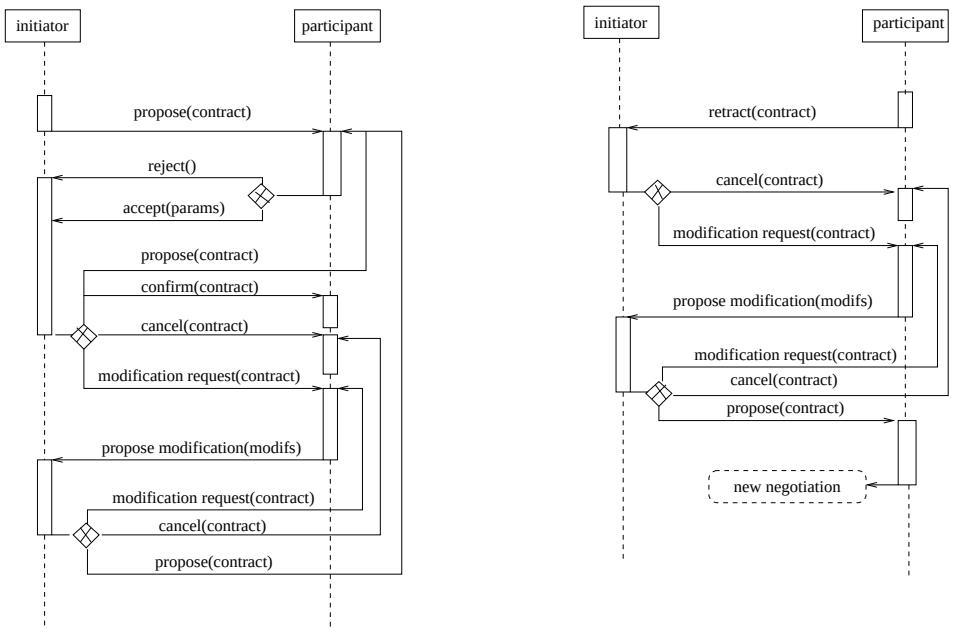

Figure 1: Negotiation protocol of GeNCA. Left : the sequence of messages between the initiator and the participants during a negotiation. For clarity, only one participant is shown here. Right : the sequence of messages for the renegotiation of a contract for which a participant has retracted.

specific primitives for participants. Our aim here is not to ensure communication between one of our agents and any other agent from another different platform (which would require a "FIPA-compliant" platform or more simply agents communicating via ACL), but to facilitate the development of an application with our agents. We don't use FIPA ACL or KQML for our negotiation primitives because they are not adapted to our protocol. The primitives defined by FIPA ACL deal with actions to perform or believes to assert. The specifications of the FIPA ACL primitives include conditions that can't be met with our model, so we can't use them with the meaning we want to give them. For example, the *propose* primitive denotes the intention to perform an action under certain conditions, whereas our meaning of *propose* is a contract offer from the initiator to participants, which will be accepted, rejected or discussed. We are not concerned here with believes. Moreover, FIPA ACL messages seem to be only textual messages, and the negotiation primitives we need for our model can't be used only with textual messages. Because of the content they use, our messages need to contain objects. The sequencing of these primitives is shown in Figure 1. Let us examine these primitives more deeply.

Initiator primitives The initiator begins and leads his negotiation process. He thus has specific primitives to do so. The initiator can send four negotiation primitives to a set of participants :

- *propose(contract)* : this is the first message sent by the initiator. He sends a contract proposal to the participants. The contract contains different resources to negotiate.
- *modification request(contract)* : this message indicates to participants that the con-

tract can't be taken like this and it has to be modified. The initiator asks participants to send him one or several possible modifications of the contract in order to propose a new one, better fitting everyone. This can also be a way to refine the contract.

- *confirm(contract)* : this message indicates participants that the contract is confirmed. The negotiation has been a success.

- *cancel(contract)* : this message indicates participants that the contract is cancelled. The negotiation failed.

Participant primitives Messages sent by a participant are only received by the initiator. It's a choice we made so that other participants don't know about these messages. Moreover, participants don't know the set of participants in the negotiation, they thus cannot form a coalition during negotiation. It is for example useful in Vickrey auctions where bids are private, or in other commercial negotiations where buyers could join their offers in order to have an interesting price as the quantity of goods asked is greater than if each buyer makes an offer for a lower quantity of goods.
Participants have three communication primitives which are answers to the initiator queries.

- *accept(parameters)* : this message replies to a contract proposal from the initiator. By this message, the participant indicates the initiator that he accepts the contract as it is. Parameters can be used in case of a partially instantiated contract. For example, it is the case in Vickrey auctions where participants have to propose a price for the article sold.

- *reject* : this message replies to a contract proposal from the initiator. By this message, the participant indicates the initiator that he refuses the contract.

- *propose modification(modification list)* : this message replies to a modification request from the initiator. The participant sends to the initiator a list of possible modifications for the contract. The number of modifications contained in the list is a negotiation parameter. This list can be empty if there is no possible modification for the contract.

A communication primitive is common to initiators and participants :

- *retract(contract)* : this primitive can be used only for a contract that has been confirmed earlier (after a *confirm* message has been sent for this contract). Both participants and initiators can use it. The agent sends this message to the initiator when he can't meet the contract taken anymore. The initiator can't prevent the agent to retract itself. Whether retraction is allowed or not depends on the application. Typically, retraction is not allowed in auctions, but is for appointment taking. That's why this possibility is a parameter of our negotiation model that is set up by the application designer, and the number of retractions allowed for the same negotiation is also a parameter.

2.3 Protocol characteristics

In this subsection, we present the type of applications achievable with this protocol, as it is aimed to be general, and then we give the complexity in number of messages exchanged during a negotiation process.

2.3.1 Applications achievable with this protocol

As we mentioned before, this protocol is inspired of the Contract-Net, and it adds an optional phase of conversation. As the protocol describes messages exchanged between agents but especially the order of messages and agents' turn to talk, and not what is the content of the message (for example, always a price ...), it allows many different applications to use it, which is not the case of many protocols such as the one used in ZEUS which is dedicated to marketplaces.

For example, you can use it in a "take it or leave it offer" form if you don't use the conversation phase. If you want to make auctions applications, you can implement English auctions as well as Dutch auctions. For English auctions, the initiator proposes his articles and participants answer giving a price as argument of the accept message if they are interested in the article, or rejecting the proposal otherwise. If no participant has proposed a satisfying price for the initiator, a conversation phase is entered where each modification consists of a new bid. The process finishes when a satisfying price has been proposed or when no one rebids or the maximum number of turns predefined by the initiator has been reached.

For Dutch auctions, the initiator proposes an article with a high price, and if no participant accepts the proposal, the initiator proposes again the article with a lower price without asking for a modification from participants. The process finishes when a participant accepts the contract, or when the price reaches the minimum price wished by the initiator, or when the maximum number of rounds defined by the initiator is reached.

This protocol is not adapted to negotiations that have to be processed on several levels, for example, for negotiating to buy a car, you can first negotiate the colour, and then the price .... This protocol is not adapted to combined negotiations (Aknine, 2002), where contracts need to be linked. For example, you can't create two contracts and say both must be taken or none. If you want several resources from the same person, you put them in a single contract, but if you want several resources from several persons, you'll need one contract per person/resource but you can't specify that all contracts must be taken or none. Despite the protocol could fit it, negotiation with argumentation (Parsons et al., 1998) is not included in GeNCA. The protocol could be adapted since the parameters of acceptance or modifications could be arguments.

2.3.2 Complexity

Complexity is an important feature in negotiation. Negotiation complexity is the reason why you can't do without negotiating agents. Let's examine here complexity in number of messages induced by our protocol.

In the worth case, for m participants at a negotiation process, the number of messages to be sent is m^n if n is the depth of cascaded renegotiation process. You imagine easily what could happen to your secretary in such case to organise a meeting with fifty people.

To prove this result, let us look at the different cases that can happen.

Linear order Assume that m persons want to take a contract. Let's call *initiator* the person who wishes to take the contract and *participants* the others. Figure 2 shows five persons, before and after that the contract has been taken (each dot represents one person).

Firstly, let us consider that all participants agree with the proposal. The initiator *proposes* the contract, the participants *agree* and the initiator *confirms* : $3 * (n - 1)$ messages are exchanged.

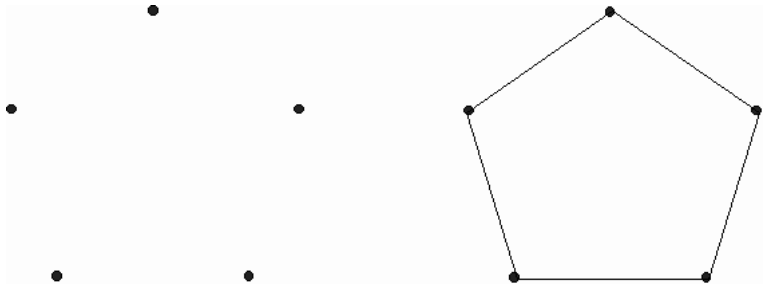

Figure 2: Complexity in linear order

As soon as one participant disagrees, the initiator *requests a modification* from participants who send one to the initiator (*propose modification* message). $2 * (n-1)$ messages are then exchanged. The initiator sends a new *proposal* which will be accepted, adding $3 * (n-1)$ messages. In total, $7 * (n-1)$ are exchanged, taking into account those of the first proposal and answers of participants with at least a negative one. The initiator sends $4 * (n-1)$ messages and receives $3 * (n-1)$. Each participant receives 4 messages and sends 3.

Taking a contract, with or without modification request, without renegotiation of other contracts, has a global complexity in O(n), is linear for the initiator and in O(1) for participants.

Quadratic order

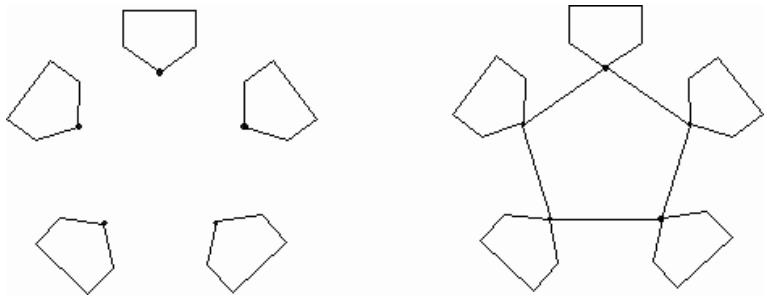

Figure 3: Complexity in quadratic order - first case

First case Let us now assume that taking a contract calls previous contracts already taken with other persons into question (Figure 3).

To simplify, all contracts will involve n persons and will have the same priority.

Participants will modify the contract, $7 * (n-1)$ messages will then be sent. But, at time to confirm the contract, each participant will have to request a modification for the contract he has already taken. Let us assume that modifications are accepted without any problem. The number of exchanged messages in this renegotiation is $5 * (n-1)$. Participants of the first contract, considered as initiators of the second ones, send $3*(n-1)$ and receive $2 * (n-1)$ messages. If all renegotiations are independent, there are $(n-1)$

renegotiations and thus $5 * (n-1)^2$ messages. The total number of exchanged messages for taking the contract is thus $5 * (n-1)^2 + 7 * (n-1)$. The initiator sends $4 * (n-1)$ and receives $3 * (n-1)$ messages. Each participant receives $4 + 2 * (n-1)$ messages and sends $3 + 3 * (n-1)$.

Taking a contract with renegotiation of another one by participant has a global complexity of O(n^2) and is linear for the initiator and participants.

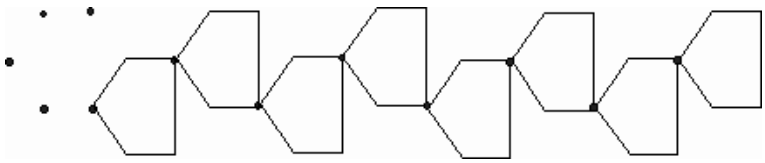

Figure 4: Complexity in quadratic order - second case

Second case Let us now assume that only one participant has to modify a contract already taken with another person (Figure 4). During renegotiation, this person also has to modify another contract and recursively on m persons. The principal negotiation needs $7 * (n-1)$ messages, the others $5 * (n-1)$. The total number of messages is $(2 + 7 * m) * (n-1)$ messages.

Taking a contract with renegotiation of another one by one participant and this recursively at a depth of m, has a global complexity of O($n * m$) and is linear for the initiator and participants.

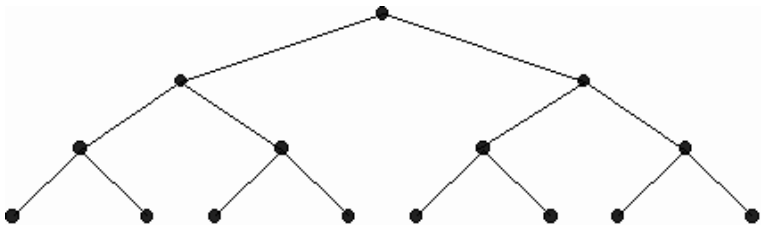

Figure 5: Complexity in exponential order

Exponential order To prove the result given at the beginning of subsection, let us take a formal example. For this example, a contract will always be negotiated between one initiator and two participants. Figure 5 shows a binary tree representing the cascaded renegotiation process. The root of this tree is the initiator of the first contract. He has got two children : the two participants. Each participant is in his turn the initiator of another contract, having also two children etc. We suppose here that there are no other relations between all these agents, ie. they are all different, all nodes represent a different agent.

Having this, we can now compute the number of messages that will be exchanged.

The number of exchanged messages for a modification of a contract which will be immediately accepted is equal to five : modification request, modification from participants, proposal of a new contract (the old one modified), agreement from participants and then confirmation of this new contract. The number of agents at level i equals 2^i and the number of messages exchanged at that level is $5 * 2^i$.

Global complexity is thus O(2^n) and is linear for the initiator and the participants.

If we now suppose that contracts are not independent anymore but that agents at level n ask the initiator of the principal contract to modify another one, the number of asks for renegotiations will be 2^n for the initiator.

Global complexity is still O(2^n) and keeps linear for participants, on the other hand, it becomes O(2^n) for the initiator.

In this section, we presented the negotiation protocol used in GeNCA, let's now see the different use modes of GeNCA.

3 GeNCA

GeNCA is a Java API for negotiation between agents. It is aimed to provide a generic software architecture for contract-based negotiations to applications developers in order to facilitate their work. The internal objects needed to the implementation of GeNCA are described in (Mathieu and Verrons, 2002). The novelty in GeNCA is that the parameters that are needed to configure a negotiation application are set up in XML files, thus avoiding recompilations at each change of a parameter value and facilitating the writing of a new application. Two kinds of files are defined : one for the system parameterisation, one for each agent which is optional. The system file contain common characteristics for all users of the negotiation system. We define them in a DTD file called genca.dtd available at http://www.lifl.fr/SMAC/projects/genca. Common resources, agents initially present in the system, retraction ability are found in it, plus default values for users parameters. Each agent can have its own file to set up its individual resources, its communicator, its strategies and negotiation parameters like default answer and answer delay. Figure 6 shows the system XML file for an appointment taking application.

We discuss here about the different ways to use GeNCA, and its major features.

3.1 GeNCA features

GeNCA major features are its conception in three levels, its negotiation cardinality and the management of deadlocks.

Conception in three levels The first feature of GeNCA is his conception in three levels, in order to separate the implementation of communications between agents, the implementation of negotiations management and the implementation of negotiations strategies. These three levels are presented more deeply in (Mathieu and Verrons, 2003a; Mathieu and Verrons, 2003b). We decided to separate these three levels in order to provide more facilities to adapt the negotiation system to applications as their common need is the negotiation level. As a matter of fact, each application has its own communication system and needs specific strategies of negotiation. For example, communications between distributed agents can be done via e-mail or a MAS platform, while communications between centralised agents can be done in a round-robin way. It is easy to define which communicator or which strategy an agent will use as it is set up in an XML file. This separation of these three levels is a difficult software engineering problem, and from our knowledge, no other platform than GeNCA separates them.

Negotiation cardinality Negotiation cardinality is an important feature for MAS. Its purpose is to know how many agents negotiate together. Different kinds of negotiation cardinality exist (Guttman and Maes, 1998), from one-to-one to many-to-many. Kasbah is an example of one-to-one negotiation : one buyer negotiates an article with one seller at a time. This form of negotiation is useful when only two persons are involved in the negotiation. But when a negotiation involves many participants with an initiator, it is a one-to-many negotiation. Our protocol enables contract-based negotiation between one initiator and several participants. Our implementation of this protocol in GeNCA allows several negotiations to take place simultaneously between one initiator and several participants, that is to say many-to-many negotiation, or more precisely many (one-to-many) negotiation. The advantage provided by many-to-many negotiation is that it enables one-to-many and one-to-one negotiation.

Deadlocks Deadlocks are an important problem in negotiation applications. It can cause many damages if it is not resolved. Deadlocks can appear when two agents propose a contract on the same resource one to the other, and when they chose to negotiate sequentially contracts on same resources. Both are then waiting to the other's answer and the deadlock appears. Deadlocks are avoided in GeNCA thanks to our mechanism of answer delay. As a matter of fact, each initiator defines the delay that have participants to answer. If a participant doesn't answer before this delay, the initiator takes into account a default answer for him and so, negotiation is not blocked.

3.2 GeNCA use modes

GeNCA can be used in different modes, which gives its genericity. Among these ways to use it, we find the kind of resources negotiated, simultaneous management, automatic renegotiation, tools for strategies and agents use modes.

Resources Resources that will be negotiated can be common to all agents or individual. If we take the example of meeting scheduling, each agent has the same agenda, and so the same time slots. Thus, resources (time slots) are common to all agents and any of them can make a proposal on the time slots he wants. On the contrary, auctions applications are typically those where we find individual resources. Agents wishing to sell articles will sell only their own articles, and not the one of its neighbours. So, for this kind of applications, resources are individual, visible to all agents but only the agents that possess them can make a contract proposal. Resources are described in XML files. If they are common to all agents, they are set up in the system file, but if they are individual, they are set up in the agent file.

Simultaneous management The management of negotiations is an important criterion in a negotiation application. Negotiations can be processed sequentially, or in parallel, depending on the constraints of the application. Two managements are possible in GeNCA, immediately or deferred simultaneous management. The user opts for the one he prefers. When he chooses to negotiate immediately all contracts, no restriction is made on the resources, they can already being negotiated for another contract. But if the user chooses to negotiate in a deferred way, the only negotiations that will take place simultaneously are the ones which involves disjoint sets of resources. The other negotiations will wait for their turn. This management of simultaneous negotiations is possible because we have designed a structure to check if all resources needed for a negotiation are free or

yet under negotiation, and so to know if the negotiation process can begin or not. This structure is a Tetris like matrix, which is described in (Mathieu and Verrons, 2002). Simultaneous negotiations are possible because we've chosen to entrust micro-agents with the management of one negotiation. In fact, each time an agent creates or receives a proposal, a micro-agent is created (a goal if the agent is the initiator, an engagement if the agent is a participant) which is responsible for the whole negotiation process of this proposal. It is thus possible to negotiate simultaneously several contracts, and being initiator as well as participant in the same time.

Automatic renegotiation Many times, during negotiations, some contracts can't be met any longer and has to be negotiated again. It is the case when appointments are negotiated. For this purpose, we propose to renegotiate automatically contracts that have to be moved. But you can't always question a contract that has been taken. For example in auctions, when an article is sold, it is definitely sold, you can't retract yourself. That's why we define a parameter called retraction allowed, used to know whether it is possible or not to retract yourself from a contract previously taken. This is a common parameter to all agents which is defined in the system XML file. If retraction is allowed, when an agent retracts itself, the initiator of the contract can automatically renegotiate the contract, and a number of renegotiations is defined by the initiator (in the agent XML file) to know how many times a contract can be negotiated again.

Tools for strategies The success of a negotiation depends of course on strategies adapted to the problem processed. We will not discuss here about strategies, which, to be optimal, must be different according to the kind of negotiation done. This is an important field which goes out of this paper. Therefore, we propose simple but generic strategies, which work for all kinds of problems, and that the user can easily refine. In order to give basis to develop strategies, two priority lists are defined in GeNCA. Each person defines a priority list for resources and a priority list for persons. Thus, each person will be able to give a priority to a contract according to priorities of resources included in the contract, and according to the initiator's priority. For example, if I took an appointment with a colleague and my boss asks me for an appointment at the same time, I will take the appointment with my boss (who has a greater priority) and I will move the appointment with my colleague. These lists can also be used in case that I am initiator of a contract and I requested modifications from participants, I can weight their answer according to the priority I gave them.

GeNCA also provides rates of success or retraction of negotiations that have been done in the past, given a participant and a set of resources. It is thus possible to know if a participant globally accepts proposals he receives, and if he keeps his engagements.

Agents use modes As we mentioned before, a human user has several ways to use its agent. He can use it with a graphical interface to interact with it, in this case, the agent is a help decision tool for the user. The agent manages the negotiations and it is the user who answers contract proposal, and creates contract to negotiate. Through the interface, the user views messages received and sent, contracts taken and being negotiated, and he can create a new contract, cancel a contract he has previously taken and reply to a contract proposal.

Another way to use the agent is the automatic way, in this case, the agent manages the whole negotiation and replies itself to proposals, the graphical interface is not used, and the agent runs like a background task.

GeNCA features and use modes have been applied to several negotiation applications like appointment taking, Dutch and English auctions and timetable creation. These applications can be downloaded at http://www.lifl.fr/SMAC/projects/genca.

In the next section, we present two applications realised with GeNCA.

4 Applications

Our aim is to propose a generic model to negotiate contracts whatever they are. The model, we called GeNCA, has been implemented in the Java language in order to provide an API for the creation of contract negotiation applications. Here we present two applications among those we have developed with GeNCA. One of them is a classical one, it uses participant individual resources, it is an auction application. The other is much less classical, it uses resources common for all participants, it is an appointment-taking system.

4.1 Application with common resources

The first application we describe here is the one which involves common resources for all participants in the negotiation. It's an appointment taking application where resources are time slots. Each agent must be able to negotiate appointments for the user. Each user defines a schedule with time slots which are free or not. In addition, he gives preferences on slots and on persons with whom he prefers to take appointments. As resources are common for all participants, each one is able to create a contract for one or several resources and to propose it to a set of participants. There is no essential need for each user to have his own XML file since resources are defined once for all in the system XML file. We obviously don't let the agents share their schedules in order to find a suitable time slot for an appointment.

This problem is a full-featured one because it needs preferences over persons, for example, the boss has a greater priority than the colleague, but also priorities over resources (here time slots), e.g. if I don't want to have appointments at lunch time or before 8 am, I'll give the corresponding time slots a lower priority. Moreover, appointment taking is an application where there are typically many renegotiations and retractions, because it is difficult to find time slots that fit everyone.

This appointment taking application involves resources of one hour timeslot in one day, and four agents running on the same computer. The system file (Figure 6) contains thus these resources and agents, and defines that retraction is possible, ie an appointment can be moved if it can't be maintained at the time defined. For this application, we used the Magique platform to run our agents, so the Magique communicator is used. Specific strategies have been implemented to fit the application, particularly to group consecutive hours if one hour was too short for the appointment.

Default values for users' parameters are set up like this : each participant has 10 minutes to answer the proposal, and would be considered as rejecting the proposal if he doesn't answer. Everyone must agree for the appointment to be taken. The initiator can request 20 times modifications from participants who can propose 5 modifications at a time. The appointment can be moved 3 times and all negotiations that take place simultaneously must involve different time slots.

This single file is sufficient to launch the application with these four agents. They all have their own GUI to create contracts, answer to proposals, view their messages sent and received and the contracts they've taken.

```
<?xml version="1.0"?>
<!DOCTYPE genca SYSTEM "genca.dtd" >
<genca>
<negotiation-type>rdv</negotiation-type>
<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>
<agents-list>
<agent><name>Paul</name>
       <address>localhost</address></agent>
<agent><name>Pierre</name>
       <address>localhost</address></agent>
<agent><name>Jean</name>
       <address>localhost</address></agent>
<agent><name>Jacques</name>
       <address>localhost</address></agent>
</agents-list>
<default-communicator>
fr.lifl.genca.magique.MagiqueCommunicator
</default-communicator>
<default-initiator-strategy>
rdv.RdvInitiatorStrategy
</default-initiator-strategy>
<default-participant-strategy>
rdv.RdvParticipantStrategy
</default-participant-strategy>
<nbRounds>20</nbRounds>
<nbRenegotiations>3</nbRenegotiations>
<minAgreements>100%</minAgreements>
<answer-delay>10</answer-delay>
<default-answer value="refuse"/>
<simultaneity value="deferred"/>
<retraction-allowed value="true"/>
<nb-modifications-by-round>5
</nb-modifications-by-round>
<magique><skill><class>
fr.lifl.genca.magique.NegotiationSkill
</class></skill></magique>
</genca>
```

Figure 6: System XML file for appointment taking application

4.1.1 Initiator behaviour

The initiator first chooses the participants he wants to meet and a time slot for the meeting. He also checks the parameters of the negotiation, such as the default answer, the minimum number of agreements to take the appointment, etc. All this define the contract and its properties. The contract is then proposed to the set of participants. The initiator thus uses the *propose* message of the protocol. Then, he waits for participants answers during the answer delay he has defined.

When the delay is over, the initiator checks participants answers. If there are more agreements than the minimum number of agreements he has chosen, he then *confirms* the contract for the participants who have agreed, and *cancels* the contract for the others. Otherwise, he *requests a modification* to all participants if the maximum number of rounds of negotiation is not reached. In the other case, he *cancels* the contract for everyone.

If the initiator requests a modification, he then waits for propositions from participants during the same answer delay. After this delay, he takes one of the following decisions :

- He *proposes* a new contract based on the propositions of the participants.
- He can't find a new contract proposal, so he *requests* again a modification from participants.
- He *cancels* the contract.

If the initiator receives a *retraction* message, he checks if there are enough participants left. In this case, he only removes the retracting participant from the list of agreed participants. In the other case, he *cancels* the contract for everyone and *requests a modification* from all participants in order to find a new contract that satisfies the participants.

4.1.2 Participant behaviour

When a participant receives a contract proposal, he first checks if the time slots proposed are free in his agenda. If they are, he accepts the proposal, thus sending the *accept* message. If the slots aren't free, he compares the priority of the initiator of the contract taken previously for these slots with the priority of the initiator of the new contract. If the older initiator has a greater priority, he then *rejects* the proposal. Otherwise, he *accepts* it.

When the participant receives a modification requests, he sends to the initiator a list of free time slots in order of preferences according to the priority he has given to the slots via the *propose modification* message.

When a contract is confirmed, the participant adds it in his agenda and *retracts* itself from previous contracts he has taken on the same time slots if they exist.

This application allows agents to negotiate appointments for its user. Contrary to other systems that can be found in shops, users' agendas are private and the problem isn't to find a suitable time slot free and common to all participants, knowing their agendas, but to negotiate the hour of the appointment, taking into account the preferences of the users on hours and persons. Moreover, this system renegotiate automatically an appointment that has to be moved due to participants retractions.

4.2 Application with individual resources

Auction applications are typically applications where resources are individual for participants. The only participants who will create contracts are the ones who possess goods to

sell. We describe here an auction application where some participants want to sell goods they have defined in their own XML file.

In this auction application, each agent must be able to negotiate auctions for the user. For this purpose, each user defines an amount of money (his credit), and a bidding strategy (linear, quadratic,...).

Auctions are defined like this : a seller proposes an article for which he wants to obtain a minimal price that he keeps secret (reservation price). Then, buyers tell him if they are interested (accept) or not (reject) in it, and if they are they propose a price for it. The seller keeps the highest price proposed and the buyer who proposed it. If this price is greater than or equals the reservation price, the buyer wins the auction. Else, the seller proposes again his article to the interested buyers for them to propose a higher price. This process is repeated until a buyer wins the auction or the number of rounds is reached.

For this application, retraction is not allowed, once an article is sold, it is definitely sold.

For this application, there are no common resources in the XML system configuration file and we launch four agents on the same computer. For this application, these agents run on a Magique platform and so they use the Magique communicator to exchange messages. Two strategies have been written to evaluate and propose bids, which are the default strategies set up in the system file. Only one person can buy the article, so the parameter *minAgreements* is set up to 1. *Three minutes* are granted for participants to bid, if they don't, the initiator considers that they *reject* the proposition. If no bid fits the initiator, he can ask a new bid *20 times* to participants, who propose a *single bid* by round. *Retraction* is not allowed. Auctions on same goods are processed sequentially, that's why the parameter simultaneity has the value *deferred*.

If users are satisfied with these parameters and do not have goods to sell, they do not need to have their own XML files. Let us take the example of our agent named Jean who wants to sell goods . Thus, he has his XML file Jean.xml where his goods (a fridge, a table and a chair, for example) are listed in the resources list. The other parameters this agent will use are those defined in the system file.

4.2.1 Initiator behaviour

The initiator first creates a contract containing the article to sell, the reservation price, the other negotiation parameters and the set of participants. The initiator then sends this *proposal* to the participants.

When an agreement is received, the initiator updates the highest bid proposed so far. If the new price tops the highest bid proposed, this new bid becomes the highest and the buyer who proposed it the current winner of the auction. Once all replies have been received, the initiator decides to *confirm* the auction for the current winner if the highest bid tops the reservation price, and thus to *cancel* the auction for the other participants. If neither the reservation price nor the maximum number of rounds are reached, then the initiator *requests a modification* from the interested buyers, in other cases, he *cancels* the auction.

When a modification proposal is received, the initiator proceeds exactly as for an agreement, as a modification proposal is a new price for the article.

4.2.2 Participant behaviour

When a participant receives an auction proposal, he first checks if the article interests him or not. If he is interested in it, he *accepts* the contract and proposes a price. Otherwise, he

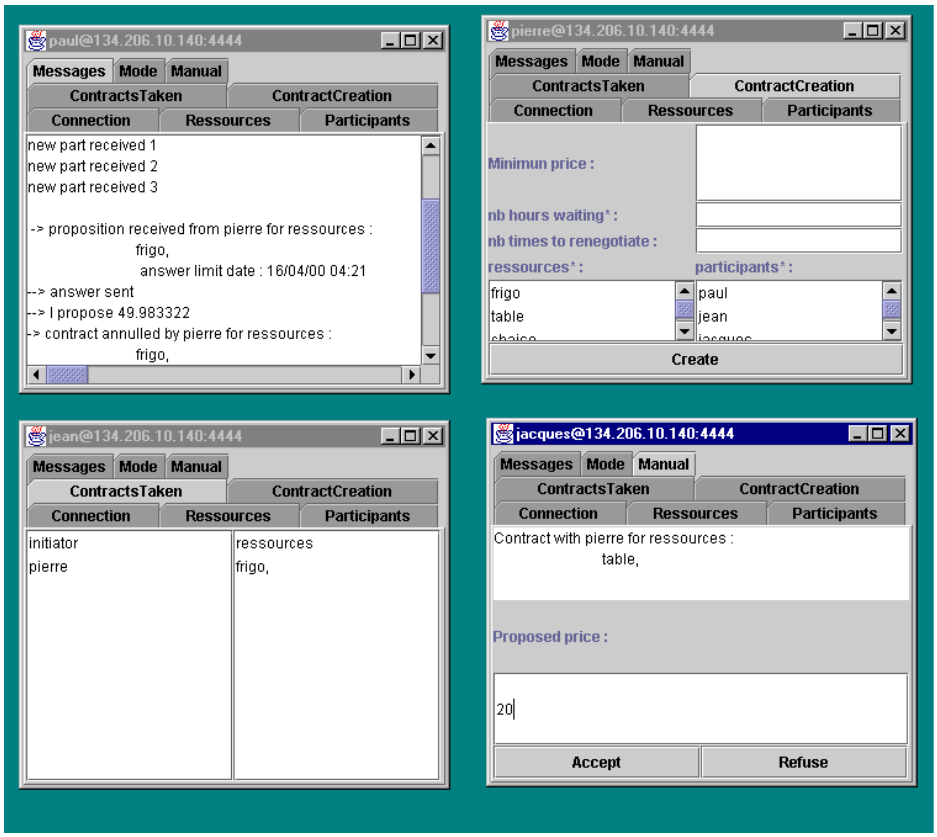

Figure 7: Four agents participating in the auction application

rejects the proposal.

When an auction confirmation is received, the participant adds the article in his bag and virtually pays the price to the seller.

When a modification request is received, the participant checks the amount of money he has and proposes a higher price than in the previous round if he has enough money or a price equal to 0 if he doesn't want to participate further in the auction.

Figure 7 shows the graphic interfaces of four agents negotiating auctions with our API.

The top left-hand screen is an agent showing his window for visualising messages sent and received by him. It permits to see the different proposals received and the proceedings of the negotiation (answer sent, confirm, cancel, modification request,...). The top right-hand screen is an agent showing the new contract input interface, the bottom left-hand one displays contracts chosen with the name of the initiator and the negotiated resources. The last one shows the display of a contract proposal for manual mode.

The advantages of this application are numerous, the most important ones are mentioned here. First, this application helps the user to bid, and bids in his place when he's not there, according to the strategy he has defined. Secondly, this application can easily be extended to other kinds of auctions, like English, Dutch, Vickrey auctions... And thirdly, this application is portable, as a matter of fact, agents can be placed on PDAs or over a

heterogeneous network.

These two examples show that our protocol can be applied to different kinds of negotiation applications such as auctions or appointment-taking. This illustrates our purpose of a generic protocol. In the next section, we compare our protocol with different applications developed by other researchers to show the differences between them.

5 Comparison with other works

We are obviously not the only ones who are interested in negotiation between agents and in proposing a generic architecture to accomplish it. Let's cite the works realised at HP Laboratories by Claudio Bartolini et al. (Bartolini and Preist, 2001; Bartolini et al., 2002b; Bartolini et al., 2002a) who want to create a general framework for automated negotiation dedicated to market mechanisms. In this paper, they define two roles : participant and negotiation host. A participant is an agent who wants to reach an agreement, while the negotiation host is responsible for enforcing the protocol and rules of negotiation. Rules of negotiation include posting rule, visibility rule, termination rule .... It is the negotiation host who is responsible for making agreements. This framework proposes a general negotiation protocol parameterised with rules to implement a variety of negotiation mechanisms. It has common properties with our, like enabling one-to-one, one-to-many and many-to-many negotiations, or like parameterisation.

Another formal work we can cite is the one done by Morad Benyoussef et al. (Benyoucef et al., 2000) who want to create a Generic Negotiation Platform for marketplaces.

A third work is the SilkRoad project (Ströbel, 2001). This project aims to facilitate the design and implementation of negotiation support systems for specific application domains. SilkRoad facilitates multi-attribute negotiations in e-business scenarios through a specific design methodology and a generic system architecture with reusable negotiation support components. A negotiation support system built on the basis of the SilkRoad architecture model acts as an intermediary between the actual negotiating agents (which might be software agents or humans) and thereby provides rule-driven communication and decision support. This project has common points with ours, like the possibility to have either software or human agents and the genericity of the system.

These three works are close to our, but they are more directed to electronic commerce whereas our model aims to fit also other types of automated negotiations.

Let's now examine two platform for negotiation : magnet and zeus. **Multi AGent NEgotiation Testbed** (Collins et al., 1998a) is a testbed for multi-agent negotiation, implemented as a generalised market architecture and developed at the university of Minnesota. It provides a support for a variety of types of transaction, from simple buying and selling of goods to complex multi-agent contract negotiation. A session mechanism enables a customer to issue a call-for-bids and conduct other business. The negotiation protocol for planning by contracting consists of three phases : a call-for-bids, bidding and bid acceptance. In contrast, our protocol enables the initiator of the call-for-bids to make counter-proposals until an agreement is reached. In MAGNET, there is an explicit intermediary into the negotiation process and agents interact with each other through it, whereas all agents directly interact with each other in our negotiation process.

ZEUS (Nwana et al.,) is a generic Java API realised by British Telecom in order to easily conceive cost-based negotiation applications between autonomous agents. Zeus proposes a negotiation protocol between two agents (an initiator and a participant) and on a single resource per contract. The protocol consists of a call-for-bids, and no mechanism

of counter-proposal is provided. Moreover, it is possible to negotiate simultaneously different contracts on the same resource, that we don't allow. Another difference with our protocol is that retraction is not possible with Zeus. Once a contract is taken you can't retract yourself. Moreover, Zeus provides only cost-based strategies, and so is less generic than our protocol which is not dedicated to cost-based contracts. Although it is possible to add an interaction protocol in Zeus, it is a difficult thing to do, as says S. Thompson in the mailing list of Zeus in April 2002. On the other hand, parameters of GeNCA negotiation protocol can be set up in XML files, which simplifies modifications.

These previous works, like our, are based on the general **Contract Net Protocol** model (Smith, 1980) which works on bids invitation between a Manager agent and Contractor agents. From all these works, Magnet is probably the one which is closest to what we present. Nevertheless, none of them takes into account at the same time generic aspects, automatic renegotiations and a mechanism to manage conflicts between simultaneous negotiations, that we propose in GeNCA. Moreover, GeNCA is the only platform which separates the communication level, the negotiation level and the strategic level.

6 Conclusion

In this paper, we have presented a generic protocol for contract-based negotiation and a Java API called GeNCA, which enables many-to-many negotiations, simultaneous negotiation of several contracts, and the management of deadlocks in conversation. Three distinct levels were defined : the knowledge representation level allowing the agent viewing the advancement of his/her negotiations, the communication level which we realised with a multi-agent platform allowing physical distribution, and the strategic level for which we propose generic strategies adaptable to any kind of problem. Each level can be easily extended by the developer as he wants to map with his application, which is a feature that only GeNCA proposes. Moreover, XML files are used to set up parameters and define an application, which facilitates the end-user work, and avoid useless recompilations. These works are a part of software engineering and distributed artificial intelligence works. Many implementation perspectives of these works on different software supports are possible (distributed, centralised, WEB) and strategic level enhancement for different specific problems is considered. This API will now be applied to different problems like distance teaching, network games, workflow systems.

References

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

Bartolini, C. and Preist, C. (2001). A framework for automated negotiation. Technical Report HPL-2001-90, HP Laboratories Bristol.

Bartolini, C., Preist, C., and Jennings, N. R. (2002a). Architecting for reuse: A software framework for automated negotiation. In *Proc. 3rd Int Workshop on Agent-Oriented Software Engineering*, pages 87–98, Bologna, Italy.

Bartolini, C., Preist, C., and Jennings, N. R. (2002b). A generic software framework for automated negotiation. Technical Report HPL-2002-2, HP Laboratories Bristol.

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

Collins, J., Tsvetovatyy, M., Mobasher, B., and Gini, M. (1998a). MAGNET : A Multi-Agent Contracting System for Plan Execution. In *Workshop on Artificial Intelligence and Manufacturing: State of the Art and State of Practice*, pages 63–68, Albuquerque, NM. AAAI Press.

Collins, J., Youngdahl, B., Jamison, S., Mobasher, B., and Gini, M. (1998b). A Market Architecture for Multi-Agent Contracting. In *2nd Int'l Conf on Autonomous Agents*, pages 285–292, Minneapolis.

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

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

Mathieu, P. and Verrons, M.-H. (2002). A generic model for contract negotiation. In *Proceedings of the AISB'02 Convention*, London, UK.

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

Mathieu, P. and Verrons, M.-H. (2003b). A Generic Negotiation Model for MAS using XML. In *Proceedings of the ABA workshop Agent-based Systems for Autonomous Processing, held by the IEEE International Conference on Systems, Man and Cybernetics.*, Washington, USA. IEEE Press.

Nwana, H., D.T. Ndumu, L. L., and Collis, J. ZEUS : A Toolkit for Building Distributed Multi-Agent Systems.

Parsons, S., Sierra, C., and Jennings, N. R. (1998). Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292.

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

Sandholm, T. (2000). *Automated Negotiation*. MIT Press.

Schwartz, R. and Kraus, S. (1997). Negotiation on Data Allocation in Multi-Agent Environments. In *Proc. of the AAAI-97*, pages 29–35.

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

Ströbel, M. (2001). Design of Roles and Protocols for Electronic Negotiations. *Electronic Commerce Research, Special Issue on Market Design*, 1(3):335–353.

Sykara, K. (1989). Multiagent compromise via negotiation. In Gasser, L. and Huhns, M., editors, *Distributed Artificial Intelligence*, volume 2, pages 119–137, Los Altos, CA. Morgan Kaufmann Publishers.

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