Three different kinds of negotiation applications achieved with *GeNCA*

Philippe Mathieu & Marie-Hélène Verrons
Laboratoire d'Informatique Fondamentale Lille – CNRS UMR 8022
59 650 VILLENEUVE D'ASCQ, FRANCE
E-mail : {mathieu,verrons}@lifl.fr

***Abstract*— In this article, we present *GeNCA*: our *Generic Negotiation of Contracts API* and three different negotiation applications achieved with it. The first one is an auction application that runs on a multi-agent platform. The second one is an application for choosing a restaurant for a dinner between friends. This application uses e-mail communications. The third application is a negotiation game inspired of the Civilization™ game of Avalon Hill Game Company. This game is played between centralised agents that speak each after another in a round-robin way. We show in this article that *GeNCA* is well suited for building these negotiation applications. As a matter of fact, different communication ways are provided with *GeNCA*, and the negotiation management and protocol are adapted to these different negotiation applications. Only strategies have to be conceived by the users, as they are specific to the application.**

I. Introduction

With the progress of information technology, multi-agent systems and electronic market places, the need for automatic agents able to negotiate with others on behalf of the user becomes stronger and stronger. Moreover, the usefulness of using an agent during negotiations is perfectly justified by the explosion of the number of messages exchanged between agents. In certain cases, it can be exponential.

For several years, many negotiation systems have been developed in specific domains like auctions or market places often for electronic commerce, let us mention Zeus [1] developed by British Telecommunications, Magnet [2] developed by the university of Minnesota, the SilkRoad project [3] of IBM, the platform GNP [4] of the Montreal university and works done at HP Laboratories [5]. Of course, negotiation can be used in other fields like appointment taking, reservation systems or even video games, but it seems that these areas have not really been studied. When achieving such applications, we can see that a lot of the notions used are the same in many systems. For example, `contracts`, `resources`, `contractors`, `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 show that it is possible to build the corresponding API. The model we propose, called *GeNCA*, 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 always difficult to formally define what negotiation is, we will base our arguments on the following consensual definition, which can be applied to many fields such as auctions, appointment taking systems, games or others.

Definition 1: Negotiation is carried out on a `contract` to obtain common `resources` and at 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.

This definition is of course inspired from the Contract Net Protocol proposed by Smith [6] in 1980, which is a fundamental of many negotiation works.

GeNCA is a general negotiation model that allows a user wishing to develop a negotiation application not to have to do the whole job but to have a model that will facilitate his work.

GeNCA is based on a three-level architecture, that separates the communication part between agents, the negotiation part and the negotiation strategy part of an application. As a matter of fact, the way agents communicate doesn't play a role in the way negotiation is made, and different communication ways can be used in a same application executed on different environments.

It is also important to separate the negotiation strategy from the two other levels, to allow a user to choose which negotiation strategy he will use without disturbing the remaining of the application. Moreover, the negotiation strategy is intrinsically linked to the negotiation application, and it is obvious that negotiating a ton of potatoes is not the same as negotiating a slot-time for an appointment, nor the same as negotiating the exclusive use of a shared resource for an hour.

The negotiation level of *GeNCA* contains a general negotiation protocol and a management of conflicting negotiations that allows to process them either sequentially or simultaneously. Parameters to specialise the protocol are set up in a file. Among these parameters, we can cite the number of agreements needed to confirm the contract, answer delay and default answer, number of rounds in the negotiation process, retraction possibility and number of renegotiations allowed.

GeNCA has been used to achieve different applications such as an auction system, a system to negotiate the choice of a restaurant for a common trip and a negotiation game application.

The auction system involve agents in a multi-agent platform

like Magique or Madkit. A seller agent proposes his article to sell to buyer agents who propose a price for the articles if they interest them in a sealed-bid fashion. The seller compares the highest proposed price with his reservation price and if the proposed price is greater than the reservation price, the buyer wins the auction. Otherwise, the seller proposes again his articles.

The system to negotiate the choice of a restaurant for a common trip involves personal agents that communicate by sending e-mails. The agents must choose in a list of restaurants the one where their users will eat. Users have preferences over the restaurants and their agents must take them into account to negotiate the choice of the restaurant.

The negotiation game application involves centralised agents with talk one after another. This game is called JNego and is inspired from the Civilization™ game of Avalon Hill Game Company. The objective of the game for a player is to negotiate the exchange of resources in order to maximize the total value of the resources he possesses. There are six different resources that have a different value. The total value of the resources a player possesses depends on the number of resources of one type by square multiplied with the value of this resource.

In this article, we first give an overview of *GeNCA*. Then we describe these three negotiation applications that we easily achieved thanks to *GeNCA*.

II. *GeNCA*: AN OVERVIEW

GeNCA is a generic contract-based negotiation model and API that allows the user to easily develop negotiation applications. *GeNCA* is based on a three-level architecture, that separates the communication part between agents, the negotiation part and the negotiation strategy part of an application. We only present in this section the negotiation protocol used in *GeNCA*. More details on the communication and on the strategy levels can be found in [7]. Then, we present *GeNCA* features and finally the way to use *GeNCA* to achieve a negotiation application.

A. The negotiation protocol and its properties

Here we present the negotiation protocol used in our model. The aim of the protocol is to define the messages that agents can send to each other with the associated operational dynamics. This negotiation protocol is characterised by successive messages exchanged between an initiator (the agent who initiates the negotiation) and participants (the agents who participate to the negotiation) like in the Contract Net Protocol framework.

1) 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 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. The sequencing of these primitives is shown in Figure 1. Let us examine these primitives more deeply.

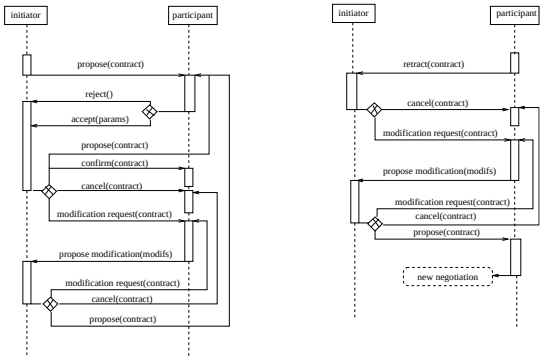

Fig. 1. Interaction graph between one initiator and one participant

Initiator primitives: The initiator can send four communication primitives to a set of participants :

- *propose(contract)* : this is the first message sent by the initiator to participants in order to propose a contract to them. The contract contains different resources to negotiate.
- *modification request(contract)* : this message indicates to participants that the contract cannot be taken as it is 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, which suits everyone. This can also be a way to refine the contract.
- *confirm(contract)* : this message indicates to participants that the contract is confirmed. The negotiation has been a success.
- *cancel(contract)* : this message indicates to participants that the contract is cancelled. The negotiation has failed.

Participant primitives: Messages sent by a participant are only received by the initiator. Other participants do not know about these messages. Moreover, participants do not know about the set of participants in the negotiation, they thus cannot form a coalition.

Participants have three communication primitives which are answers to the initiator's queries :

- *accept(parameters)* : this message replies to a contract proposal from the initiator. With this message, the participant indicates to 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. With this message, the participants indicate to the initiator that they refuse the contract.
- *propose modification(modification list)* : this message replies to a modification request from the initiator. The participant sends 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)* : the contract has been confirmed but a participant or the initiator can't honour it anymore. The agent then decides to retract from the initiator.

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

Fig. 2. DTD file to configure the negotiation protocol.

2) Protocol parameters: We have presented a general protocol to model different types of negotiation. Here we detail the parameters needed to specify this general protocol in order to obtain specific negotiation protocols. We have chosen to configure these parameters into an XML file, so we have conceived a DTD file (Figure 2) in order to validate it.

During distributed negotiations as it is the case when agents act for their user, a participant may not answer to the initiator proposal, either because he is not there, or because there was a failure. Negotiation then must not be blocked. In order to continue the negotiation, an answer delay mechanism is used, and when this delay is over, the initiator considers a default answer for the participant who has not answered. This default answer will often be a rejection of the proposal, as a matter of fact, in commercial negotiations, one can't oblige someone to buy the good. This default answer is given to the participants, so if their answer is the same, they don't have to answer and communications are limited. Even if it seems strange, an agreement as default answer can be useful, especially when negotiation takes place in an appointment taking system, for example. If the initiator wants to receive answers before 10 minutes and considers a rejection by default, the parameters will be : `<answer-delay>10</answer-delay>` and `<default-answer value=''refuse''/>`.

For the initiator to decide whether to confirm or annul the contract, given the answers of participants, a parameter indicating the minimum number of agreements needed to confirm the contract is set up. This number can be a percentage. For example, for an auction, only one participant must accept the contract, whereas in other applications, everybody might be agree : `<minAgreements>100%</minAgreements>`.

In order not to have an infinite conversation phase, we introduce a number of rounds in negotiation, that is to say the number of times a participant can propose a modification for the contract, make a counter-proposal. We have chosen to limit negotiation duration by an answer delay and a number of speech rounds rather than with a maximal duration for the whole negotiation process as used in general, because we think negotiation will be more efficient this way. We can effectively assume that the number of counter-proposals done will be greater if agents must answer in quicker delays than if they only know a limit date for the end of negotiation and in this case answer less rapidly to initiators. But this usage comes more specifically from the fact that a limit date for negotiation poses many problems, and the major is the one of a universal reference time. As a matter of fact, synchronisation of the different computers where the agents are running is a real problem we couldn't solve.

This number of rounds will then be affected to zero if negotiation is a *take it or leave it offer* one, or if it is first or second-price sealed-bids auctions : `<nbRounds> 0 </nbRounds>`. At every round, each participant can propose some modifications to the contract. This number of modifications is set up by the parameter : `<nb-modifications-by-round> 0 </nb-modifications-by-round>`.

We told about the necessity to be able to retract oneself from a contract previously taken, for some kinds of negotiation. For this reason, a boolean parameter indicates whether retraction is authorised or not, and another parameter sets up the maximum number of renegotiations allowed : `<retraction-possible value=''true''/>` and `<nbRenegotiations>5</nbRenegotiations>`.

Thanks to all these parameters, it is possible to specify the general protocol in order to fit a negotiation. Removing one parameter would lower the protocol generality and strength. Let us take the example of the number of rounds in the negotiation, if we remove it, that is to say we only do one-proposal negotiations, it is no longer possible to implement Dutch or English auctions. Our proposal consists in offering a system providing this general protocol and taking into accounts these parameters in order to instantiate different negotiations.

B. *GeNCA features*

GeNCA features are developed in [8], we only mentioned them here. A first feature is the *XML parameterisation*. In *GeNCA*, the parameters that are needed to configure a negotiation application are set up in XML files: one for the system parameterisation and one for each agent which is optional.

The *management of conflicting negotiations* that can be done in *GeNCA*, is either sequential or parallel management. The user opts for the management he prefers.

Many times, during negotiations, some contracts can't be met any longer and have to be negotiated again. For this purpose, we propose to *renegotiate automatically* contracts that have to be moved. 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 to know how many times a contract can be negotiated again.

The success of a negotiation depends of course on strategies adapted to the problem processed. 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.

C. Using the package to create an application

The package we provide implements the whole negotiation level and gives default implementations for the interfaces of the communication and strategic levels.

Implementations of the communication level we give, allows the use of the Magique and Madkit platforms, the use of threaded agents acting in a round-robin way, and the use of e-mails. We also provide the server agent to which the users' agents subscribe and which is responsible for message sending. For these four kinds of uses (Magique, Madkit, round-robin and e-mail), a main class launching the application is given. For any other kind of communication mode (sockets,...), it is necessary to implement the *Communicator* interface and to have an agent that integrates the name server implemented in the package.

Default strategies provided with the package are quite simple but can be easily refined. They take into account priorities given to resources and to persons in order to choose which contract to accept in case of conflict, and which resources to propose in case of modification request.

The package also provides a graphic interface for negotiation, which allows the user to create a contract, to visualise the messages sent and received by the agent, to answer a contract proposal if the manual mode is chosen, to visualise contracts taken by the agent, to have a view on the negotiations being conducted on resources and to retract a previously chosen contract.

In our package, the human user has two ways to use its agent. Manually, it is then a decision-helping tool which shows the state of all current negotiations, and, in this case, it is the user who answers a contract proposal. Automatically, this time, the agent is hidden and answers proposals by itself without human interventions.

To write an application with the package, one only needs to implement the interfaces of the communication and the strategic levels (if the ones provided don't suit the application), and to define the XML configuration file where the resources and the negotiation parameters are indicated, and of course to write his own agents including the *Negotiator* of the package. When the application concerns the negotiation of contracts that only contain resources, there is nothing else to do. The whole management of the negotiations is automatically done. In return, if the contract needs other parameters, such as a quantity or a price, the *Contract* class must then be extended and the graphic interface for creating a contract must be updated to include these new parameters.

III. An auction application

Auction applications are more and more used over the internet, websites such as *eBay*, *onSale*, etc. know a growing interest from people. We propose here to achieve an auction application involving agents on a multi-agent platform, where bids are sealed. 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,...).

A. Description

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 or not 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 predefined number of rounds is reached.

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

B. Analysis and implementation with GeNCA

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

In this auction application, a new parameter is involved in the negotiation: a price. Default strategies provided in the package thus don't fit the application, others must be conceived. The price is not a parameter of the contract, so this class hasn't to be modified, but proposed by interested buyers in the *accept* message. In return, the reservation price is a *property of the contract*, this class must then be extended in order to include it.

The graphical interface for creating a contract must also be modified in order to get this reservation price and also the graphical interface for contract proposal in manual mode in order to allow the user to enter a price for the article if he is interested in it.

The *negotiator* must also be extended in order to manage the selling and buying of resources and the wallet of the user. In return, if we use the Magique platform, no work is needed for the communication level.

C. Negotiation strategies

1) Initiator strategy: The initiator first creates a contract through the graphic interface, by indicating which articles he wants to sell, participants to whom the articles will be proposed, the answer delay for participants answers, the default answer he will consider (a rejection) and the number of rounds

in the negotiation. In the auction case, an additional parameter must be indicated : the reservation price.

When an agreement is received, the strategy 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.

2) Participant strategy: 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 *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.

D. Configuration files

Figure 3 shows the common configuration file of the auction application. No resource is common to everyone as participants only sell articles they possess. As we mentioned it before, retraction is not authorised and only one acceptance is needed to sell the article. Obviously, the default answer is a rejection. Agents use the Magique communicator and have by default no money.

An agent who wishes to sell articles must have his own configuration file where his articles are defined. Each agent also has to define his amount of money in his configuration file. Figure 4 represents the configuration file of an agent named Paul who wants to sell a table, a fridge and a cook book. He has 50 euros.

Figure 5 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.

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

Fig. 3. XML configuration file for the auction application

E. Advantages of using GeNCA

Many auction applications exist, among which we can cite Kasbah [9], AuctionBot [10] and Fishmarket [11], but in most of them, a third person collects offers to sale and offers to buy and match them.

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, etc. And thirdly, this application is portable, as a matter of fact, agents can be placed on PDAs, over a heterogeneous network, etc.

IV. A RESTAURANT CHOICE SYSTEM

When coming out with friends to have dinner, the choice of the restaurant is many times a problem. The application we present here aims to solve this problem before the day of the dinner by negotiating the choice via e-mails. Users have

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

Fig. 4. Configuration file of an agent

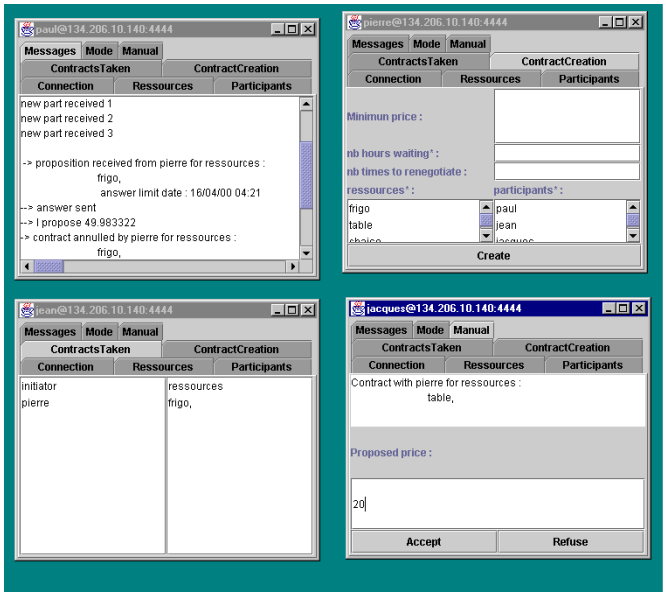

Fig. 5. Four agents participating in the auction application

preferences over the restaurants and their agents must take them into account to negotiate the choice of the restaurant.

A. Description

A group of friends wants to choose the restaurant where they will eat together for their next meeting. They know a list of restaurants and they choose their next restaurant in this list. Moreover, they rank the restaurant from the most to the least preferred. One of the friends will propose a restaurant to the others, who reply if they like or not the restaurant. If 75% of the friends like this restaurant, they have found where they will eat. Otherwise, another restaurant has to be proposed. To choose the next one, the proposer requests proposals to the friends and computes the one which is most popular to propose this one to everyone. The process ends when the restaurant satisfies at least 75% of the friends.

B. Analysis and implementation with GeNCA

Resources to be negotiated in this application are the restaurants and are common to all friends. These resources thus appear in the common configuration file.

The contract only has to contain the proposed restaurant, so nothing has to be added to this class. The preferences between the restaurants can be stored in the priority list defined in the model, so no new data structure has to be created. There is no parameter that hasn't been defined in the model to be added. So there is nothing to do in the negotiation level.

The package provides a mail communication system, so there is nothing to do for this level too.

The only level that has to be improved is the strategy level.

C. Negotiation strategies

Strategies for choosing a restaurant can be based only on preferences of the user or also on the previous restaurants where the friends have eaten.

1) Initiator strategy: The default strategy provided in the package can be used for this application. As a matter of fact, this strategy confirms the contract if at least the minimum number of agreements is reached, here, this number is 75%. If this number is not reached, the initiator requests a modification from the participants. Once the modifications received, the initiator gives a mark to each resource according to the number of times it appears in a modification proposal from participants. Then, the initiator proposes the resource which has the best (greatest) mark.

As this strategy fits the application, no work has to be made for the initiator strategy.

2) Participant strategy: Several strategies can be defined. A simple one consists of accepting a restaurant if it is in the top half of the preference list. If a modification request is received, the participant proposes the restaurant at the top of his list, and goes through this list if other modifications are requested.

Another strategy is to accept the proposal if it is not the restaurant where they eat last time.

D. Configuration file

Figure 6 shows the common configuration file for this application. We can observe the list of restaurants, the strategies used to negotiate, and the protocol parameters. The minimum number of agreement is set to 75%, participants have 20 minutes to reply to a proposal and if they don't, it is as if they have accepted. Each participant proposes a single restaurant if a modification is requested, and they can propose one up to 5 times.

E. Advantages of using GeNCA

The only work needed to achieve this application with *GeNCA* is to conceive participant strategies and define the configuration file. There's nothing else to do ! This ease of achievement of a negotiation application is a great advantage. Moreover, this application can be easily extended to fit voting methods like Borda count or Hare system.

```
<?xml version="1.0"?>
<!DOCTYPE genca SYSTEM "genca.dtd" >
<genca>
  <application-name>restaurant
  </application-name>
  <resources-list>
    <resource>little</resource>
    <resource>cheap</resource>
    <resource>expensive</resource>
    <resource>self</resource>
    <resource>fastfood</resource>
  </resources-list>
  <communicator>
    fr.lifl.genca.mail.MailCommunicator
  </communicator>
  <default-initiator-strategy>
  fr.lifl.genca.strategy.DefaultInitiatorStrategy
  </default-initiator-strategy>
  <default-participant-strategy>
    resto.RestoParticipantStrategy
  </default-participant-strategy>
  <protocol>
    <answer-delay>20</answer-delay>
    <default-answer value="accept"/>
    <minAgreements>75%</minAgreements>
    <nbRounds>5</nbRounds>
    <nb-modifications-by-round>1
    </nb-modifications-by-round>
    <retraction-allowed value="true"/>
    <nbRenegotiations>3</nbRenegotiations>
  </protocol>
  <default-priority value="5"/>
  <management value="sequential"/>
  <window value="true"/>
</genca>
```

Fig. 6. XML configuration file for the restaurant application

V. JNEGO : A NEGOTIATION GAME

We have conceived this game by inspiring us of the Civilization™ game of Avalon Hill Game Company. This game allows us to illustrate many concepts from negotiation as well as game theory, which makes its richness. This game aims to incite competition between players that have to maximize the value of the resources they possess. This game runs in a synchronous way, players speak each one in his turn.

A. Rules of the game

There are 6 different resources which worth, in this order and by convention, from 1 to 6 points, and that are present in the game in limited quantities.

1) wheat
2) wood
3) stone
4) bronze
5) silver
6) gold

Each player possesses N resources. The value of the *ith* resource is denoted *value(r_i)* and *nbCards(r_i)* denotes the number of cards representing the *ith* resource that the player possesses. The value of a player's hand, denoted *value(hand)*, is computed according to the following formula:

$$\text{value(hand)} = \sum_{j=1}^{6} \text{value}(r_i) * \text{nbCards}(r_i)^2 \qquad (1)$$

This formula shows that the more resources you get in the same family, the more points you get. As a matter of fact, the increase is quadratic to the number of identical resources possessed. Let us take an example of one hand containing 3 wood cards and a hand containing 1 wood, 1 wheat and 1 stone. The first one is worth $2 * 3^2 = 18$ points, whereas the second one is worth $2 * 1^2 + 1 * 1^2 + 3 * 1^2 = 6$ points.

To succeed in obtaining a hand that has a maximal value, players exchange cards. The only obligation is to exchange 2 cards for 2 cards. This sometimes constrains players to broke a group of identical resources they have formed. Assume that a player possesses 5 cards of stone and 1 of wood. If he wants to exchange his card of wood, he must also give 1 card of stone.

Three kinds of exchanges can be proposed : the first one indicates what is given and what is asked, the second one indicates only what is given, and the third one only what is asked.

B. Analysis and implementation with GeNCA

The rules indicate that the resources are commonly known by all players, and each of them possesses a total of N resources. These resources thus are defined in the common configuration file.

As the contract provided by the package contains resources to negotiate but provides no way to distinguish resources given or asked for an exchange in this game, the contract must be extended to include a second set of resources to indicate given resources. The set of resources contained by the basic contract represents asked resources.

For this game, one needs to know what hand he has, what is its value, and to be able to update his hand when he makes an exchange. He also needs to determine which resources he'd like to ask or to give. These functionalities are added to the Negotiator class, so that they can be accessible by the strategies.

Two classes of our negotiation level have to be extended for this application. Concerning the communication level, no work will be needed because our API provides a CentralisedCommunicator which is designed for synchronous communication in a round-robin way.

The most important work to be done is to conceive strategies for this game.

C. Negotiation strategies

We present in this subsection quite simple strategies because our aim here is not to provide the best strategies for this game but to show that *GeNCA* is well-suited for designing this application. Better strategies can be obtained by adding a knowledge base which stores the number of times a player wanted to obtain or to give each resource. This can help

a player to determine which resources interest which other player, and who is likely to give resources that interest him.

1) Initiator strategy: When the exchange specifies what is asked and what is given, it is the simplest case. If a player has accepted the exchange, it is confirmed. If no player has accepted it, it is cancelled. If several players have accepted the exchange, one is chosen (randomly or the first who had answered) to achieve the exchange.

When the initiator has only specified given resources, he must evaluate each player's proposal concerning the resources he would get. The initiator chooses the exchange e_i such as :

$$\begin{cases} \text{gain}(e_i) > \text{gain}(e_j) \forall j \neq i \\ \text{gain}(e_i) > 0 \end{cases} \quad (2)$$

with

$\text{gain}(e_k)$ = value(hand after the exchange e_k) − value(hand before the exchange e_k)

The initiator cancels the contract for every other player. If no exchange satisfies these conditions, then the initiator request a modification from participants so that they propose other cards. When all modifications are received, the initiator chooses the exchange by the same way. If no player wants to give the initiator the resources, the contract is cancelled.

When the initiator proposes an exchange giving only asked resources, he first check if he has the resources wanted in exchange by the other players. If he has the resources for several exchanges, then he chooses the one which satisfies conditions 2 and he cancels the contract for the other players. If the initiator has no one of the resources asked by the players or if he doesn't want to give them, then he request a modification from the players. He then evaluate the modifications by the same way.

2) Participant strategy: When an exchange specifying the resources asked and given by the initiator is proposed to a player, he first checks if he has the asked resources. If he hasn't the resources, he refuses the contract. Else, he computes if the exchange would provide him with a gain. If it is the case, he accepts the contract, otherwise he refuses it.

When the player receives a proposal mentioning only given resources, he checks if these resources interest him. It could be the case if he already has such resources in his hand. If the resources don't interest him, he refuses the contract. Else, he accepts the contract and specifies which resources he would give in exchange. For example, he would give resources he has in a single item, or that value less than the cards he would get. If the initiator requests a modification, the players proposes other cards that he has and that don't interest him, or nothing if he has no more proposals to make.

When the player receives a proposal mentioning only asked resources, he checks if he possesses these resources. If he doesn't possess them, he refuses the contract. Otherwise, he computes if an exchange could raise his hand's value, and in this case he proposes this exchange. In the other case, he refuses the contract. If the initiator requests a modification, he proposes another exchange that raises his hand's value if one exists. Else, he proposes nothing.

D. Advantages of using GeNCA

The advantages of using *GeNCA* for this application are that only two classes have to be extended for the negotiation level and that the user only has to define strategies for this game. All the remaining of the application is already done.

VI. Conclusion

In this paper, we gave an overview of *GeNCA*, our general model for contract-based negotiation between agents, and presented three different applications that we achieved thanks to our model. The first one is an auction application that runs on a multi-agent platform. The second one is an application for choosing a restaurant for a dinner between friends. This application uses e-mail communications. The third application is a negotiation game inspired of the Civilization™ game of Avalon Hill Game Company. This game is played between centralised agents that speak each after another in a round-robin way. We showed that these applications don't need a lot of work to be achieved with *GeNCA*, and that the principal work was to define negotiation strategies for each application. Other applications have been achieved thanks to *GeNCA*, they are available on http://www.lifl.fr/SMAC/projects/genca.

References

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

[2] J. Collins, B. Youngdahl, S. Jamison, B. Mobasher, and M. Gini, "A Market Architecture for Multi-Agent Contracting," in *2nd Int'l Conf on Autonomous Agents*, Minneapolis, May 1998, pp. 285–292.

[3] M. Ströbel, "Design of Roles and Protocols for Electronic Negotiations," *Electronic Commerce Research, Special Issue on Market Design*, vol. 1, no. 3, pp. 335–353, 2001.

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

[5] C. Bartolini, C. Preist, and N. R. Jennings, "Architecting for reuse: A software framework for automated negotiation," in *Proc. 3rd Int Workshop on Agent-Oriented Software Engineering*, Bologna, Italy, 2002, pp. 87–98.

[6] R. G. Smith, "The Contract Net Protocol : high-level communication and control in a distributed problem solver," *IEEE Transactions on computers*, vol. C-29, no. 12, pp. 1104–1113, December 1980.

[7] P. Mathieu and M.-H. Verrons, "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*, Madeira Island, Portugal, july 26-30 2003, pp. 169–176.

[8] ——, "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, October, 5-8 2003.

[9] A. Chavez and P. Maes, "Kasbah : An Agent Marketplace for Buying and Selling Goods," 1996.

[10] P. Wurman, M. Wellman, and W. Walsh, "The Michigan Internet AuctionBot : A Configurable Auction Server for Human and Software Agents," in *Proceedings of the Second International Conference on Autonomous Agents*, Minneapolis, MN, USA, May 1998.

[11] P. Noriega, "Agent mediated auctions: The Fishmarket Metaphor," Ph.D. dissertation, University of Barcelona, 1998.