In International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce (MAMA'2000)

A model of virtual competition between remote agents

F.Andriamasinoro, R.Courdier

IREMIA, University of La Reunion
15, Avenue René Cassin
97715 Saint-Denis
La Réunion-FRANCE
E-Mail: {fandriam, courdier}@univ-reunion.fr

Abstract

When multi-agent research is used in electronic commerce, most of time, the study relates to the negotiation. The general idea is to send agents negotiating in place of human negotiator, after preliminary allotted the appropriate behavior. The behavior depends on the type of transaction.

The negotiation is however only one angle of vision. We should not either be unaware of the trade under its competing aspect; i.e. the purchasers are virtually in competition to obtain common resources. The term *virtual* is justified by the fact that agents which are in competition do not have any direct link between them.

In this paper, we would like to contribute our share on this second aspect, which is the competition. The case study is typically the electronic commerce in its aspect of remote competition between several purchasers. We then propose an agent approach for analyzing this kind of situation. Our objective is to build software agents which can manage competition in a distributed environment.

Keywords: Competition, Reaction, E-commerce, Agent, Simulation

1 Introduction

In multi-agent domain, many case studies were already taken to analyze the form of existing cooperation between agents. The reference examples are for instance a community of ants which help each other to exploit a same source of foods [BON 94] or the robots which transport ores [FER 97]. In such cooperation, two aspects may be generally found when conflicts are happening: negotiation and/or competition. Since many works already related to the negotiation process ([SIE 99], [FAR 98], [WUR 98]), our present one was focused on the competition aspect. [FER 97] already outlined this one that he designed as antagonism, but his main work develops cooperation rather than antagonism.

Our framework intends to build software agents which can manage competition in a distributed environment. This type of work is of considerable interest, in particular for (economic or social) entities in remote competition. Everyone who practices the stock-exchange on Internet is the first concerned. Since user-machine (and then, software agents in it) have response times faster than the human being, decision time becomes beneficial for the user, in the conquest of one market. Anyway, agents also have to manage time constraint as well as its suitable reaction facing the current market event.

In this paper, we propose a model of this competition. Each machine is, in fact, composed by agents which act coordinately. These agents can be considered as a team which is in concurrence with another one in different machine. Our simulation aims to test such a structure in competition.

The term 'virtual' is chosen because of the members of the group, which (generally) do not know each other. Members are geographically distant but are connected by a network (often the Internet). The example of the auction sale on Internet, presented in [MAN 98] and [WUR 98] would not then any more be considered under the angle purchaser-salesman in negotiation, but purchaser-purchaser in competition.

Section 2 of this paper introduces the issues generated by this type of situation. Section 3 outlines our agent organization and architecture. From this architecture, we describe in section 4 all actors of our system and the interaction between them to perform the concurrence processing. An example of competition simulation is presented in section 5 before we conclude and give the prospects for our work.

2 Issues

They can be seen under various angles. The first one is the time corresponding to the network information

crossing. Considering that our agents are software ones, which process data in a fast time, the time of transport is a significant factor. [LEE 00] already considered this problem by analyzing the intensity of the traffic on the Web, not in agent analysis. In this paper, we only simulate this moment of transmission to make the model more realistic. However, future may envisage other method such for instance an information agent, found in [KLU 00].

The second parameter is the dynamic of the group. There are moments when a member may come to be added to the competition or another withdraws itself. This dynamism does not have to block the system functioning by making transactions as independent as possible.

The third factor is related to the application. As we take the case of the sale, 3 cases can appear:

- in the case of auction sale, there is a time constraint for the acquisition of the product. Thus, the first which will propose the greatest value at the last time will gain the goods. In other words, if we have *n* agents on the system and indicate by $Prop(A_i, t)$ the proposal of A_i at the instant *t*, all agents *Ai* must have as objective,

$$\forall t \in [0, tEndAuction],$$
$$if\ val(Bidding) = Max(\{Prop(A_k, t)\}_{k=\overline{1,n}})\ then,$$
$$Prop(A_i, t) >= val(Bidding)$$

In this type of sale, there is competition;

- if sale is a 'classic' one and the number of products is lower than the number of potential buyers, there is also conflict and the determining factors are the network crossing time as well as the stocks values;
- but if merchandises are in sufficient quantity, the concept of competition disappears.

So, according to the type of sale, our agent must modify its behavior. In the case of an auction sale, it is obliged to react a little more quickly since it knows that the time of action is limited and information will still have to transit by the network.

All things considered, we show in our work a structure of agents and a form of organization of the system taking these various parameters into account. Each agent is sufficiently flexible to adapt itself to different kind of purchases.

3 Architecture

3.1 Basic organization

Our architecture follows that of GEAMAS [MAR 98] in which there are 3 levels of agents (by downward approach):

- the highest level contains MacroAgent which is the agent having a total sight of the system;
- the lowest one is the MicroAgent, which corresponds to fine-grain agents. Interactions, micro-behaviors and evolution capabilities describe each agent.
- the Medium-Level is an intermediate level between the Micro and the Macro-level. It is composed of a set of micro-agents that have a common characteristic or a common aim. The Medium-Level may be short-lived.

Our agents do not have position in the environment. According to the application, they can however be represented there by situated objects. In this manner, these agents can evolve independently of a given environment.

3.2 Autonomy and role

Our agents are reactive ones [FER 97] whose behavior varies according to the events which reach them. For that, one agent A_i has a set of reactions $REACT_i$ that it can undertake depending on these events. We consider that an agent always reacts even when it does nothing. In this case, the reaction is represented by one element $none \in REACT_i$. So, *REACTi* always contains at least one element which is *none*. If *SYS* is the set of agents in our system, then:

$$\forall A_i \in SYS, REACT_i \neq \{\} \text{ and } REACT_i = \{none, \ldots\}$$

Values in *REACTi* depend on the role agents play in the system. If an agent A_i plays several roles, $REACT_i$ is made of the union of all reactions involved by each role. In other words, if $\{role_{i1}, role_{i2}, \ldots, role_{ik}\}$ designates the set of *k* roles which one agent plays, then:

$$REACT_i = reaction(A_i) = reaction(\{role_{i1}, \ldots, role_{ik}\}) = reaction(\{role_{i1}\}) \cup \ldots \cup reaction(\{role_{ik}\})$$

The real example is that of an agent which plays the role of purchaser and salesman at the same time.

The fact that $REACT_i$ always contains the element *none* shows well the autonomy of our agents whatever the roles they are playing. Indeed, an agent is allowed to ignore all events by adopting the *none* reaction. In addition, it is completely free in its decision to continue or not its transaction.

For information, each agent also has a set of properties *PropAg* which is an influencing parameter for its behavior. This behavior may be internal and depends on the application. For instance, if we consider a culture as an agent, its internal behavior may be its growth.

3.3 Interaction between agents

Events arriving to agents are presented in the form of messages. Each agent has an internal mailbox to temporarily store them. Agent progressively treats the events according to their arrival, and in an atomic way.

In other words, the next message is treated only when the precedent is finished. An event is a message M which is expressed in the form:

M = < reference, time, keyword, sender, receiver, list of arguments (lArgs) >

the significance of arguments is the following:

- *reference* makes it possible to identify the message in a single way. It can for example consist of the sender identifier combined with the number of events occurrences;
- *time* is the moment of sending the message. Its value is obtained owing to the timer Agents (see next section);
- *keyword* is a primitive which will depend on the reaction of the agent according to the current treated event. Each agent A_i has an inner table *reaction* establishing the relation between the value of *keyword* and the corresponding reaction *react* that must be undertaken. Thus; *react = reaction[keyword]* and *react Î REACT*$_i$.
- *lArgs* consists of a set of values useful for *receiver* when it carries out *react*.

For instance, the following message

M = <"AgServ.1", "28/05/00 10:30:35", auctiondelay, AgServ, AgCli, {3, "minutes"}>

indicates that at 10:30 am, the server identified by *AgServ* informs *AgCli* that the first auction sale (so .1) will be closed in 3 minutes.

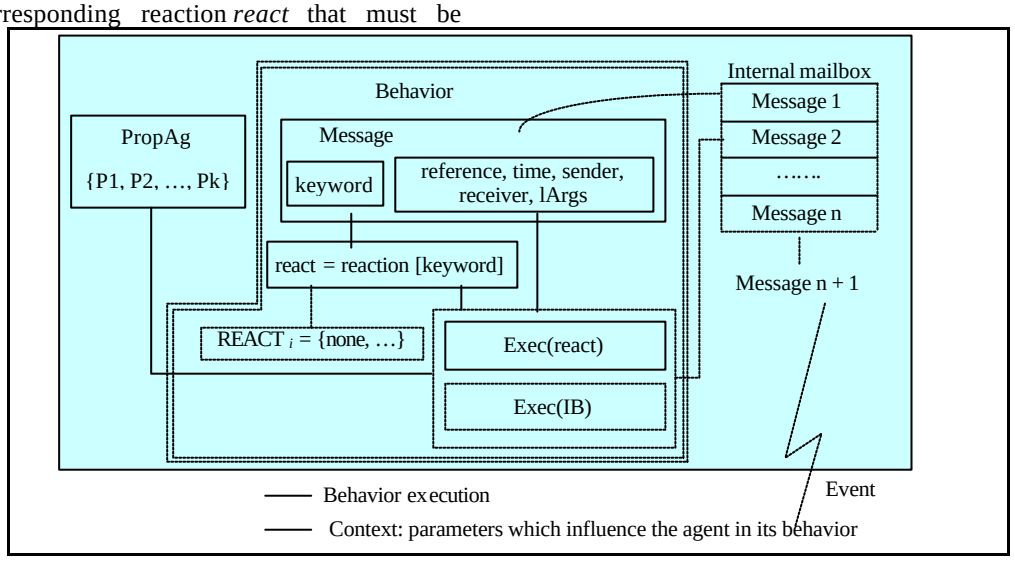

Figure 1: Agent architecture

4 The actors

Our agents are classified in 4 categories according to their role: the timer agents, the client agents (*AgCli*), the server (AgServ) and the conveyors (AgTransp).

4.1 The timer agents

The competing aspect of our study implies us to take account of the time. It corresponds to a particular agent called Macro-timer. This one is the temporal reference of the system. Concretely, it is represented within different agents of the system by many Micro-timers. The principle is the same one like in which everybody perceives the notion of time and has an individual watch, or still software agents have the clock of the machine in which it is. Each agent *AgCli* and *AgServ* in the system is then in acquaintance link with one Micro-Timer agent.

In its internal behavior, the MacroTimer (i.e. all Micro-timers) increments the time of one *timeUnit*[1] and synchronization is automatic because the MacroTimer is universal.

The time management is based on alarm. This last contains a message that one agent must perform after a certain time. An alarm *A* is in form

A = Alarm(receptor, message, time_delay) where *receptor* is the agent which executes *message* after

[1] The real domain of *timeUnit* depends on the application. The domains often used are seconds, minutes, etc.

time_delay timeUnit. The parameter *message* is the same that we have seen in the previous section.

The simple example is that of the auction sale during which the machine server will announce the end of the sale after one hour. If the *timeUnit* we take is the minute, an alarm is then defined as follows:

if M = < *ref., time, endAuction, agent-server, agent-server, LArgs* > is the message to be sent,

$$Al = Alarm(agent\text{-}Server, M, 60)$$

4.2 The client agents(AgCli)

They are the agents in competition. Their attitude aims to obtain products in the server, by reacting to the convenient period. In the case of an auction sale for example, the set of reactions $REACT_k$ of an agent $Agcli_k$ is:

$REACT_k$ = *{none, connect, propose, bid_higher}*

4.3 The server agent (AgServ)

It is responsible for managing all requests coming from all client agents. For better understanding, we take a case study.

Case study

The server agent presents 2 types of sale possible to potential buyers : auction sale and classical sale. Depending on the kind of sale, it reacts differently to all requests which come to him. The exchange is performed in real time. We now show by a sample of messages how our structure is flexible enough to handle these two kinds of sale. To make it simplified, we do not write the parameters *reference, time, sender (AgCli)* and *receptor(AgServ)*

	Message from server	
Message from client	**Auction sale**	**Classical sale**
<connect,, {}>	<auctionInfo,, {timedelay, actualAuctionValue}>	<saleInfo,, {price}>
<proposePrice,, {price}>	<timeout,, {}> or <repAuction,, { actualAuctionValue }>	- (ignored)
<putBucket,, {product}>	- (ignored)	<infoBucket,, {nbproducts}> or <sorry,,{noStock}>
<buying,, {credit_card_number}>	<confirmBuying,,{product timeDeliver}>	

Table 1: interaction between buyer and seller

In our study, the server agent doesn't notify all client agents when any new winning bid has been received. If the client agent wishes to know it, it asks the server by sending it the 'currentBid' message. In Internet, this situation exists, probably to reduce the number of communication and to make the competition very interesting.

Concurrent access management

Its architecture itself allows one agent to manage the concurrent accesses whatever the type of sale which *AgServ* wants to undertake. Like the processing of the events is done in an atomic way, only one client will be treated at the same time.

The taking into account of a message depends on only 3 factors:

- the arrival of the message within the agent server,
- the processing of the preceding messages,
- the timeout, in the case of an auction sale. The server agent has the intelligent behavior to reject all messages which arrives after the end of auction sale.

In this paper, we do not present the protocol of interaction used during the exchanges.

4.4 The conveyor agents (AgTransp)

They are some particular agents which are created by client and/or server agents. Their role consists in transporting the messages towards its destination. In other words, sending the message towards the network. Thus, it is an agent by delegation. The existence of a conveyor agent is advantageous insofar as it makes it possible for its creator agent (client or server) to be quickly free to perform other tasks. The obvious example is about a server which treats several offers at the same time.

One conveyor agent is dynamically created by the agent which wishes to communicate. Compared to its creator, it is a subordinated agent. However, it acts in an autonomous way in its manner of sending messages on the network. Its life-span depends on its creator *AgCli*. When this one decides to stop transaction, it 'kills' all its *AgTransp* and dissolves the link which binds it to its Micro-timer agent. Out of respect of the principle of autonomy, *AgCli* cannot 'kill' its Micro-timer, but only cut the links which bind them. This mechanism allows us to manage the dynamics of the model, particularly if a client would suddenly decide to leave the network during the competition. Then, the system would not be overloaded by useless agents.

AgTransp is represented by a situated object which is the network of routing. As it is really impossible to specify the duration of transport of one message, we allot a certain property *tConvey* to *AgTransp* in order to

simulate this time of routing. Moreover, another value *d* simulates the variation of the network flow. Thus these values make it possible to simulate competition closer to reality.

In each machine, there are then three software agents which are in acquaintance, and cooperate during the concurrence period: one micro-timer agent, one client or server agent and one or more conveyor agent. They constitute one team. The competition is then held between several of these teams (except the server).

The next figure shows our system according to GEAMAS representation.

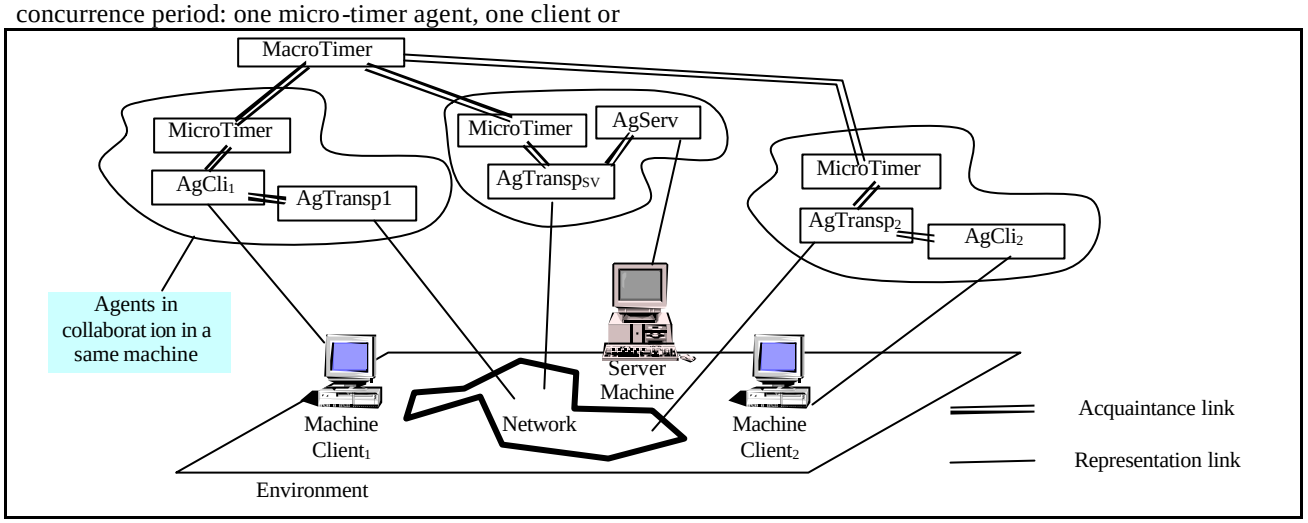

Figure 2: General structure of the system of concurrence

5 Simulation

To validate our model, we make simulation. The implementation has been realized in JAVA 1.2.2. The follow-up of simulation is carried out starting from the graphic user interface (GUI) of GEAMAS. This GUI makes it possible to follow the evolution of the system.

In particular, it is possible to observe:

- all events (messages) sent and received by each agent, (in textual form)
- all messages exchanged in the system, (also in textual form)
- for each agent, the current value or the evolution of their properties (*PropAg*),
- the duration of simulation

5.1 CPU resource problem

Each agent of the system must be able to obtain the resources of the processor to be able to perform its behavior. In the implementation, it was initially expected that each agent corresponds to a process (*Thread* in JAVA). Thus, there are several processes which run in parallel on the system. Their number is identical to the number of agents *(nbAgents)*: *nbProcess = nbAgents*.

However, this type of implementation presents some limits when the number of agents increases. Simulation slows down considerably when *nbAgents ³ 20*. In addition, the attribution of processor resource among threads becomes fortuitous. The checking of the exchanged messages enabled us to note that some agents have more processor resources than others. That enabled them in practice to be always the first to acquire the bought goods while others had more interesting bids but could not send them. The simulation results were then not conclusive.

To mitigate this type of problems, we have decided to manage the attribution of resources on the level of GEAMAS and not on the one of JAVA. In practice, the control is done at MacroAgent level which has a total sight of the system and which knows all agents.

With this intention, the MacroAgent traverses each agent and allots to it an execution time of its (internal and external) behavior. This time corresponds to the processing of one event. At the end which corresponds to one cycle, it returns to the first agent etc. The Macro-Agent has also an internal table making it possible to manage the priority of the processing. This priority is a numeric value. Thus, an agent of rank *n* will be treated only one time per *n* cycles. This principle is feasible insofar as an agent is autonomous and can carry out its behavior independently of the agents. Thus, there is neither inequality nor conflict.

5.2 Scenario

The following simulation brings into play 3 agents in competition in an auction sale on a server. The client

agents have, as an identifier respectively $client_1$, $client_2$, $client_3$ and server identifier is $Serv_1$.

The sequence of messages below shows the server agent can manage concurrent accesses, and competition between client agents is virtual.

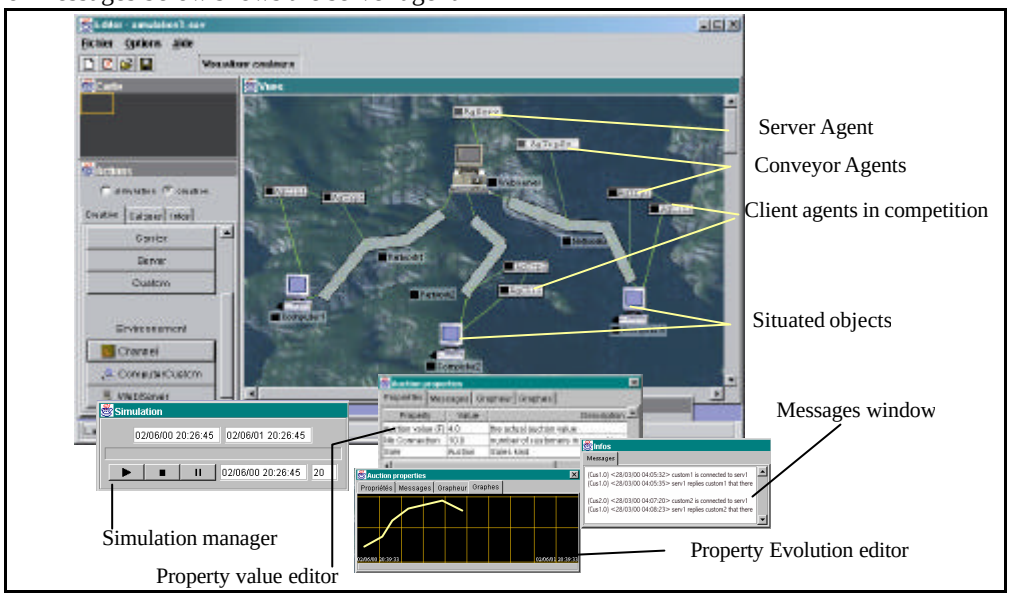

Figure 3: Simulation of 3 client agents in competition for an auction sale on a server

The following sequence presents then series of messages displayed during simulation

// Initialization phase

(Cli1.0) < 28/03/00 04:05:32 > client1 wants to be connected on serv1

(Cli2.0) < 28/03/00 04:05:33 > client2 wants to be connected on serv1

(Cli3.0) < 28/03/00 04:05:35 > client3 wants to be connected on serv1

(Serv1_Info) < 28/03/00 04:05:37 > serv1 presents an auction sale. The initial amount is 40F and auction sale duration is 1 hour

// network crossing... client2 is the first connected: crossing time: 1mn 47s

(Cli2.0) < 28/03/00 04:07:20 > client2 is connected on serv1

(Cli2.0) < 28/03/00 04:07:40 > serv1 answers the connection of client2 and informs it of an auction sale. The current amount is 40F (it remains 58 minutes)

// then, it is the arrival of the 2 others with a little delay due to a low flow

(Cli1.0) < 28/03/00 04:08:15 > client1 is connected on serv1

(Cli1.0) < 28/03/00 04:08:35 > serv1 answers the connection of client1 and informs it of an auction sale. The current amount is 40F (it remains 57 minutes).

(Cli3.0) < 28/03/00 04:08:59 > client3 is connected on serv1

(Cli3.0) < 28/03/00 04:09:19 > serv1 answers the connection of client3 and informs it of an auction sale. The current amount is 40F (it remains 56 minutes)

// first proposal of client2

(Cli2.0) < 28/03/00 04:09:48 > client2 is informed of the auction sale of serv1. It sends a proposal of 50F

// first proposal of client1 by being unaware of what client2 proposed. It is the beginning of the virtual competition

(Cli1.0) < 28/03/00 04:11:33 > client1 is informed of the auction sale of serv1. It sends a proposal of 45F

/ / first proposal of client3 by being unaware of what the 2 others proposed

(Cli3.0) < 28/03/00 04:13:00 > client3 is informed of the auction sale of serv1. It sends a proposal of 60 F

// network crossing of the proposal of Client2, but there is falls of the flow. Lasted course = 4mn 35s

(Cli3.0) < 28/03/00 04:14:23 > serv_1 is informed of a proposal of 50F on behalf of client2. It raises the bidding with 50F and informs client2 that it is for the moment the winner (it remains 51 minutes)

// arrival of the proposal of client1 a little faster than its first crossing

(Cli1.0) < 28/03/00 04:14:50 > serv_1 is informed of a proposal of 45F on behalf of client1. It informs to him that the amount is already of 50F (it remains 51minutes).

// arrival of the proposal of client3 … and so on….

(Cli3.0) < 28/03/00 04:16:23 > serv_1 is informed of a proposal of 60F on behalf of client3. It raises the bidding with 60F and informs client3 that it is for the moment the winner (it remains 49 minutes)

6 Conclusion and perspectives

In this paper, we were dealing with the analysis of the agents which are in competition to get a common goal. Our agents behave as reactive one which react sequentially to events (in the form of messages) which reach him. The case study which we took is a set of agents which are virtually in competition for a sale on the Internet. Our objective was to build software agents which can manage competition in a distributed environment. Each machine in competition contains 3 kinds of agents (considered as one team) which act coordinately during competition: client/server agent, the Micro-Timer agents, making it possible to ensure the temporal coherence of the system (each client or server agent is in acquaintance link with one Micro-timer) and agents, created to transport information towards its destination. The model must indeed consider competition as a situation in which time is a fundamental factor.

The model we presented and simulated has the following advantages:

- its architecture allows one agent to treat all offers in real time;
- the dynamism of the system is controlled. If a competitor enters or leaves the competition, the process is held in a transparent way to the server agent and the other client agents of the system.
- so that simulation is into in conformity with reality, we took account of the duration of the crossing of the network.

In the majority of sales, negotiation is necessary before having the product. It involves many transactions in the network. Then, if an agent wants to gain a competition of purchase, it must also have the talent of negotiator. Being reactive is not sufficient any more. One of our subteam is now working on this negotiation aspect. Afterwards, our next work is to allot this negotiation capacity to our agents.

7 Acknowledgement

This work is planned to be carried out jointly with Alem Leila coming from CSIRO Australia. We would like to thank her for accepting to join us in this project.

8 References

[BON 94] E.Bonabeau, Guy Theraulaz, *Intelligence collective*, Edition Hermes, 1994

[FAR 98] P. Faratin, C. Sierra, N. R. Jennings, *Negotiation decision functions for autonomous agents*, Robotics and Autonomous Systems, 24 (3-4) pp. 159-182

[FER 97] J.Ferber , *Les systèmes Agents, Vers une intelligence collective*, Inter-Editions, 1997

[HAM 96] Z.Hamzah, D Girivendhen, *Automated Negotiation*, final report in Surprise 96, also visible in *www-dse.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/zah /report.html*

[KLU 00] M.Klusch, *Information Agent Technology for the Internet: A Survey*, Journal on Data and Knowledge Engineering, Special Issue on Intelligent Information Integration, D. Fensel (Ed.), Kluwer. forthcoming issue, 2000

[LEE 00] J. Lee, M. Podlaseck, E. Schonberg, R. Hoch, and S. Gomory, *Understanding Merchandising Effectiveness of Online Stores*, in International Journal of Electronic Commerce and Business Media, Vol 10(1):1-9, 2000.

[MAN 98] M.Kumar, SI.Feldman. *Business Negotiations on the Internet*, Technical Papers of IBM Institute of Advanced Commerce, in *http://www.ibm.com/iac/reports-technical/reports-bus-neg-internet.html*

[MAR 98] P.Marcenac, R.Courdier, S.Calderoni: *Zooming on a MultiAgent Simulation System: from the Conceptual Architecture to the Interaction Protocol*. In Third International Conference on MASs (ICMAS-98), pages 411-413, July 1998.

[SIE 99] C.Sierra, *Agent-Mediated Electronic Commerce. A European viewpoint* in JFIADSMA'99, November 1999, Edition Hermes

[WUR 98] PR.Wurman, WE.Walsh, MP.Wellman, *Flexible Double Auctions for Electronic Commerce: Theory and Implementation.* Decision Support Systems 24:1 (1998), 17-27*;*