Developing a Model Agent-based Airline Ticket Auctioning System

Mladenka Vukmirovic[1], Maria Ganzha[2], and Marcin Paprzycki[3]

[1] Montenegro Airlines, Industry Development Department
Beogradska 10, 81000 Podgorica, Serbia and Montenegro
mladenka.vukmirovic@mgx.cg.yu
[2] Elbląg University of Humanities and Economy,
ul. Lotnicza 2, 82-300 Elbląg, Poland
ganzha@euh-e.edu.pl
[3] Computer Science Institute, SWPS, 03-815 Warsaw, Poland
marcin.paprzycki@swps.edu.pl

Abstract. Large body of recent work has been devoted to multi-agent systems utilized in e-commerce scenarios. In particular, autonomous software agents participating in auctions have attracted a lot of attention. Interestingly, most of these studies involve purely virtual scenarios. In an initial attempt to fill this gap we discuss a model agent-based e-commerce system modified to serve as an airline ticket auctioning system. Here, the implications of forcing agents to obey actual rules that govern ticket sales are discussed and illustrated by UML-formalized depictions of agents, their relations and functionalities.

1 Introduction

Broadly understood e-commerce is often viewed as a paradigmatic application area of software agents [11]. In this context software agents are to facilitate higher quality information, personalized recommendation, decision support, knowledge discovery etc. When developed and implemented, agent systems are to be, among others, adaptive, proactive and accessible from a broad variety of devices [13]. Such systems are also to deal autonomously with information overload (e.g. large number of e-shops offering the same product under slightly different conditions—price, delivery, warranty etc.). In this context, modern agent environments (such as JADE [9]) can support implementation of quasi-realistic model e-commerce scenarios [8]. Moreover, advances in auction theory have produced a general methodology for describing price negotiations [6, 7]. Combination of these factors gives new impetus to research on automating e-commerce and autonomous software agents are cited as a potentially fruitful way of approaching this goal [10].

Unfortunately, the picture is far from perfect. While there exist a large number of attempts at developing agent-based systems, they are mostly very small-scale demonstrator systems—later described in academic publications. In the meantime, some applications utilize the agent metaphor, but not existing agent tools and environments. Separately, it is almost impossible to find out if agent systems exist in the industry; e.g. establish the true role of the Concordia agent system within the Mitsubishi Corp.

Finally, and this is very important in the context of the current paper, most of work devoted to either automatic price negotiations (see [6, 7], or multi-agent e-commerce systems (see [8, 10, 11]) involves "virtual realities." In other words, auctions are conceived, for instance, as a general case of an English auction used to negotiate prices of product P, while for multi-agent systems buyer agents are send by users U_1 and U_2 to e-shops $S_1, S_2, \ldots, S_n$ to buy products P_1 and P_2. As a result, proposed systems do not have much to do with real-life. When virtual agents compete to purchase non-existent products, their behaviors are also virtual, as they are not grounded in any possible actual application.

The aim of this paper is to make an initial attempt at bridging the gap between theory and practice. We start with a model agent system presented in [1–4, 8]. In this system we have modeled a distributed marketplace that hosts e-shops and allows e-buyers to visit them and purchase products. Buyers have an option to negotiate with representatives of e-stores to choose the shop from which to make a purchase. Conversely, shops may be approached by multiple buyers and through auction-type mechanisms, have an option to choose the "best?' potential buyer. Furthermore, this system attempts at remedying the well-known conflict between agent intelligence and mobility. By precisely delineating which agent modules have to be transferred and when should this happen we were able to reduce network utilization. Since this system is theoretical in the above described sense, in this paper we will discuss its modification required to apply it to a more realistic airline ticket auctioning scenario. What we found particularly interesting was that this could have been achieved with only relatively minimal changes in the overall system.

Before proceeding let us make explicit some of the assumptions made in our current work. (1) In the original system e-stores were *drivers* within the marketplace — buyers could purchase *only* products that were available for sale through existing e-stores. This being the case, we have decided, in the initial phase of our work, to accept this approach (while planning to remove this limitation in the future). Therefore, currently multiple "travel agencies" sell tickets to a variety of popular destinations. They obey basic rules of airline ticket trading, but it is only "them" who decide which tickets to sell. In other words, if the user of the system would like to fly from Hattiesburg, MS to Tri Cities, WA, she may not find such a connection. At the same time, connections between Amsterdam and Detroit, MI may be sold by every e-store. While this assumption may seem limiting, we would like to point out that success of priceline.com (and other auction places that sell airline tickets) makes our model scenario "realistic enough." (2) We are still utilizing the *CIC* agent that stores both "yellow-pages" (what?) and "white-pages" (who?) information as the approach to matchmaking [12]. However, we see interesting extensions of its role in the system (e.g. by allowing it to study market trends and sell this information to interested sellers — see below). (3) In all situations where it was possible we utilize existing structures that have been described in [1–4, 8] and interested readers should consult these sources for further details. This being the case we can focus current paper on modifications introduced by grounding the system in the real-life rules that govern the air-ticket sales.

2 Description of System

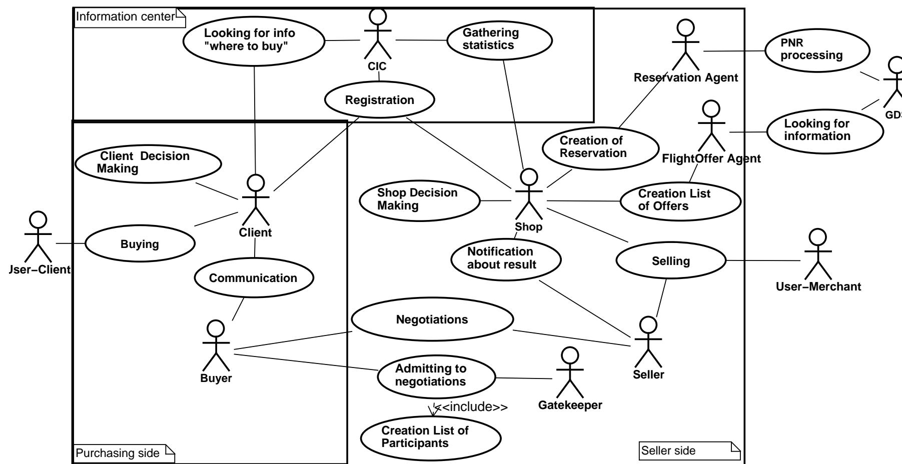

Fig. 1. Use case diagram

We describe our system through its UML-based formalizations. We start with the use case diagram depicted in Figure 1. Here we can see all agents present in the system, their interactions and their interactions with external systems — e.g. the Global Distribution System (GDS). This system stores all information about flights, number of available seats, booking classes and so on. In reality, there exist several GDS servers available to airlines to offer their inventory for sale; e.g. AMADEUS, SABRE, Worldspan, Galileo etc. In our work we base our description on the actual functioning of the AMADEUS GDS through which more than 200 airlines offer their services. Note, however, that replacing AMADEUS with a different server, or adding additional external servers to the system would not require substantial changes in its functioning and in functioning of its constituent agents (e.g. the *Flight Offer Agent* would only have to communicate with multiple external servers).

We can distinguish three major parts of the system. (1) The *information center area* where white-page and yellow-page data is stored and serviced by the *CIC* agent. Let us note that every request for which the *CIC* had to send a “negative answer” (i.e. no such travel is available for sale) is stored. As indicated above, in the future this data can be utilized (purchased) by *Shop Agent*s to decide which tickets should be offered to address existing and/or changing market-demand. (2) The *purchasing side* where agents and activities representing user-buyer are depicted, and (3) the *seller side* where the same is presented for the user-merchant. In the modified system, in comparison with the original one [4], we not only have the external *GDS* but also two new agents: the *FlightOffer Agent* and the *Reservation Agent*. Together they replace the functionalities provided in the original design by the *Warehouse Agent*. Let us now describe in more

detail these agents that are substantially different or new to the system (in comparison with these described in [4, 8]).

2.1 Shop Agent

Shop Agent (*SA*) acts as the representative of the user-merchant and, at the same, time as a *Control Center* of the *Sale Process*. In the current stage of conceptualization of the system we follow the path selected in the original e-commerce system, where the user-merchant was specifying the input provided to the system. Thus we assume that user-merchant inputs all necessary data: departure airport code, destination airport code, booking class, fare basis code, initial rule by which seats are to be offered for sale. For example, if user-seller wants to sell out seats that would have been offered for Advanced Purchase Excursion Fare—APEX, but time limit for this fare has expired, user-seller would specify the number and the period for which he wants to offer seats on flights. This info is used in availability check and price retrieval. The period is needed to set bounds within which flights will be offered. Optionally user-merchant can specify flight number as well. This narrows down the availability list and may be necessary in the case when there is more then one flight per day between two given destinations. Furthermore this can be used also in the case when, for instance, user-merchant wants to offer seats on morning flights, but not on the evening flights. So, she specifies which flight number(s) can be chosen from. In this way, all other possible flight numbers are excluded.

In the near future we plan to extend functionality of our system. In particularly, while at present our system acts only as a “distributor” of a predefined set of tickets, we would like to modify it in such way that the *SA* could start distributing (acquire and put for auction) not only what user-merchant wants to sell but also what user-clients are looking for. Observe that we have already introduced a mechanism to facilitate this goal. Since the *CIC* agent stores information about all unfulfilled user-client queries, an *SA* will be able to obtain an access to this data (e.g. purchase it), analyze it and decide that, for instance, there is a growing need for tickets between Warsaw and Delhi and offer these for sale.

Statechart diagram of the *Shop agent* is depicted in Figure 2. At first the *SA* creates the *Gatekeeper Agent* (which plays here exactly the same role as described in [4]) and waits for a user-merchant order. After receiving such an order the *SA* creates *FlightOffer Agent*, which communicates with the *GDS* and gathers needed information to create list of offers for the *Shop Agent* (one *FlightOffer Agent* is created for each route to be serviced and exists for as long as tickets for a given route are sold by the *SA*). List of offers includes information about every itinerary: data about both (inbound and outbound) flight numbers, number of seats and class of service for both flights etc. On the basis of this list *Shop Agent* creates *Seller Agent*(s) (one for every itinerary), introduce them to the *Gatekeeper* and enters a complex state called *Control Center*. Note here that *Seller Agent*s play exactly the same role as that described in [4]; they are to interact with incoming *Buyer Agent*s and through some form of price negotiation mechanism (e.g. an auction) select the *Buyer* that may purchase the ticket. In the *Control Center* state the *SA* is listening to its *Seller Agent*(s). After receiving a message from one of the *Seller Agent*s the *Shop Agent* acts depending on content of that message.

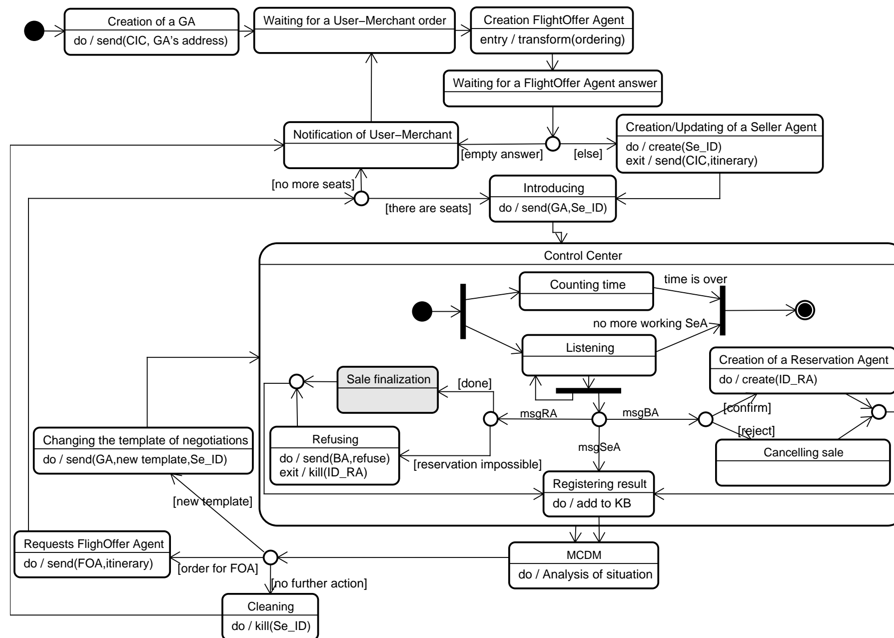

Fig. 2. Shop Agent statechart diagram

1. If the *Seller* informs about a winner of price negotiations the *Shop Agent* waits for the corresponding *Buyer Agent* to confirm that it plans to actually buy the ticket. Here, we have to stress, that in our general e-commerce model it is natural that multiple *Buyer Agent*s visit multiple e-stores [8]. Specifically, separate *Buyer* visits each e-shop that offers a given product (in our case a ticket satisfying needed itinerary). The end of price negotiation means that the *Buyer* should consult with the *Client Agent*. Therefore, the *SA* does not know if the auction winner will actually attempt at making a purchase.
2. If the *Buyer Agent* confirms it wants to buy ticket, the *Shop Agent* creates a *Reservation Agent* (*RA*), which communicates with the *GDS* to make a reservation. There are then the following possibilities:

   – If the *RA* was able to reserve tickets (it is possible that while the negotiations were taking place all tickets available in a given class of service etc. are already gone), it sends the reservation data to the *Shop Agent*. Upon reception of the data (all communication in the system is carried using ACL messages) the *Shop Agent* transfers it further to the *Buyer Agent* and carries out standard procedures involved in completing the sale (state "Sale finalization").
   – In the opposite case (the *RA* was not able to secure the reservation) the *Shop Agent* notifies the *Buyer Agent* that reservation is impossible and kills the *Reservation Agent*.

3. If the *Buyer Agent* sends message that it does not want to make a purchase, this fact is registered in a local *Knowledge Database*. More precisely, all information about processes that take place within the shop when it is attempting to sell tickets is recorded in the *Knowledge Database*. In the future, this information will be used by the *SA* to adapt its behavior. Currently we denote this fact by introducing the *MCDM* box, which denotes multi-criterial decision making. In our system we utilize a modified negotiation framework [2–4] introduced originally by Bartollini, Jennings and Price [6, 7]. In this framework, the negotiation process was divided into a generic *negotiation protocol* and a *negotiation template* that contains parameters of a given negotiation. These parameters specify, among others, the negotiation mechanism itself. Observe, in Figure 2, that one of possible results of *MCDM* is change of the negotiation template. In other words, the *SA* may decide that since only very few tickets are left but there is also only very short time to sell them, it will deep discount them and sell them with a fixed price, or through a very short time lasting English auction with a low threshold value and a relatively large increment.
4. If there is no winner, the *Shop Agent* writes information into the *Knowledge Database* and starts to analyze the current situation (the *MCDM* box in Figure 2. As a result it may change the negotiation template, or request another itinerary from the *FlightOffer Agent*. Finally, it may establish that for that given route (user-seller order) either there is nothing more to do (all tickets have been sold) or that nothing can be done (the remaining tickets cannot be sold in the current condition of the market). Then it will remove all "servant" agents servicing that route and inform its user-merchant about the situation.

It is important to note that we assume that in all price negotiation mechanisms the *Seller* institutes a time limit for negotiations. This moment is presented within the *Shop Agent* diagram as a sub-state "Counting time" (within the "Control Center" state). If the *Seller* does not sell any tickets within that time the *Shop Agent*, again, registers this information in the *Knowledge Database*, kills this *Seller* and notifies its user-merchant accordingly. Following, the *SA* enters the *Multi-criterial Decision Making* state. As described above, here it can decide, among others, to sell more seats on some specific itinerary or to change the template of negotiations or to conclude that nothing more can be sold and its existence should be completed.

2.2 FlightOffer and Reservation Agents

These two agents have been added to the system and their role is to communicate with the *GDS*. The statechart diagram of the *FlightOffer Agent* is presented in Figure 3. This agent communicates with the *GDS* to find information about flights that satisfy conditions specified by the user-merchant. If such flights are available the *FlightOffer Agent* prepares (process represented by multi-state boxes "Checking availability," "Find Class of service capacity," "Price retrieval" and "Analyzing module") a "List of Offers" for the *Shop Agent*. All the multi-state states—"Checking availability," "Find Class of service capacity," "Price retrieval" and "Analyzing module"—involve communication with the *GDS*. On the Figures 4 and 5 we present statecharts of "Checking availability"

and “Price retrieval” states to illustrate the nature of proposed communications between the *FlightOffer Agent* and the *GDS*. Upon obtaining all the necessary information form

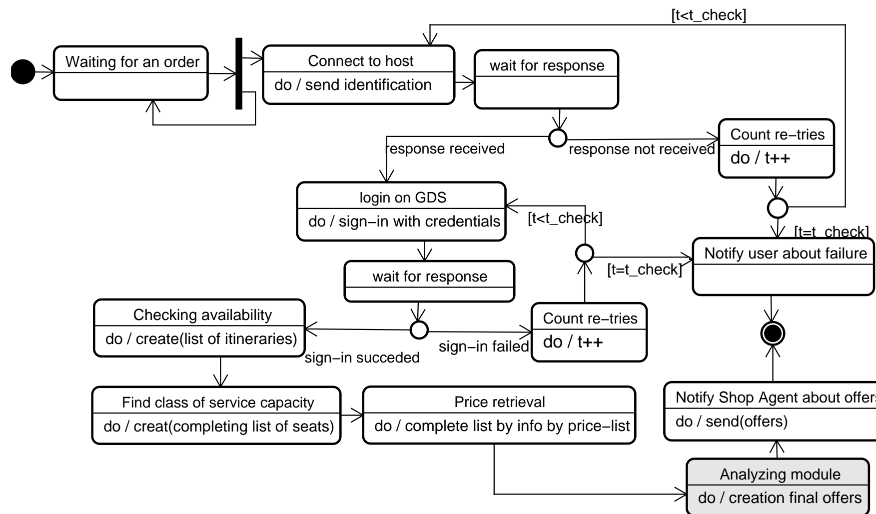

Fig. 3. FlightOffer Agent statechart diagram

the *GDS* it sends the information to the *Shop Agent*. Note that the role of the “Analyzing module” is to check the request of the user-merchant against the data retrieved from the GDS to make sure to assure consistency of the final offer (e.g. if the user-merchant requested 10 seats, but only 5 are available then only 5 can be in the offer).

The second agent that communicates with the *GDS* is the *Reservation Agent*. It is created by the *Shop Agent* after receiving confirmation of willingness of making a purchase from the *Buyer Agent*. Its function is to make an actual reservation within the *GDS* server. In case of successful completion of its task the *Reservation Agent* transfers all reservation’s data to the *Shop Agent*. If reservation is impossible it informs about it the *Shop Agent*. Both cases mean that its job is complete and it then self-destructs. Its extremely simple statechart diagram is omitted.

3 Concluding Remarks

In this paper we have discussed how to modify a model agent-based e-commerce system to turn it to a simplified airline ticket auction-system. To achieve this goal we have studied the way that airline tickets are sold and assumed that e-stores in our system can connect and communicate with a Global Distribution System (e.g. AMADEUS) and act according to the existing rules governing its behavior. In this stage of the project we have decided to proceed with only minimal modifications to the existing e-commerce model system and were positively surprised that we will be able to keep unchanged the original

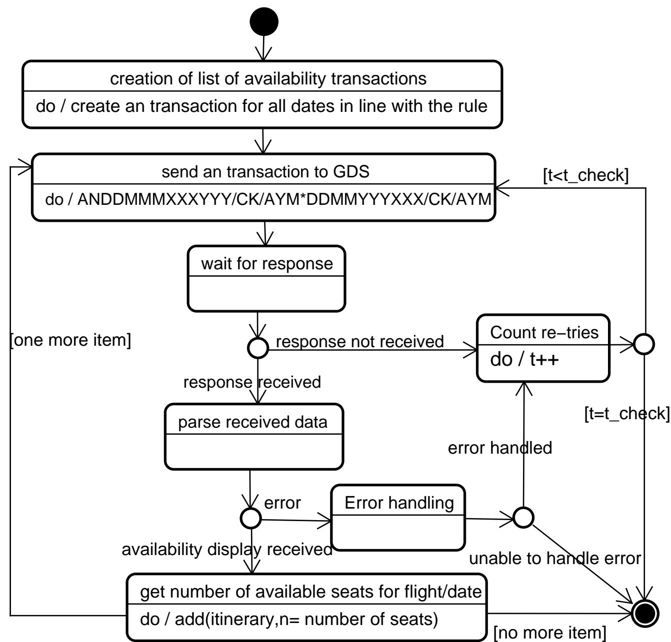

Fig. 4. FlightOffer Agent's "Checking availability" substate statechart diagram

Client and *Buyer* agents, as well as practically unchanged the *CIC* agent. Thus, the *purchasing* and *matchmaking* parts of the original system remain unchanged. Furthermore, the *Gatekeeper* agent, that is responsible for admitting *Buyer* agents to the host and to negotiations, as well as managing *negotiation template* changes can remain unchanged.

The proposed changes to the system consist of: (1) replacing the *Warehouse* agent by the *Reservation Agent* that is responsible for communicating with the *GDS* and establishing if it is still possible to make the requested reservation. (2) Introducing the *FlightOffer Agent* that acts as a liaison between the *Shop Agent* and the outside world. Its role is to communicate with the *GDS* and find out details of flights that satisfy the request of user-merchant. In other words, the user-merchant specifies which routes she world like to sell and the *FlightOffer Agent* finds connections that can be used to serve these routes and returns this info to the *Shop Agent* start selling specific itineraries. (3) Finally, the *Shop Agent* underwent a substantial modification to act as a seller of airline tickets. For each of the new and/or modified agents we were able to formalize its functioning through it UML statechart diagram.

Separately, we have identified the next step in the development of the proposed system. Currently, the system is merchant-driven, which means that users can buy only what merchants offer them and if they want to fly different routes, they are out of luck. We have introduced into the system appropriate mechanisms that when explored and utilized will mediate this problem. Namely, e-shops will be able to learn about unfulfilled user requests and respond to them. Furthermore, we have specified how, in the

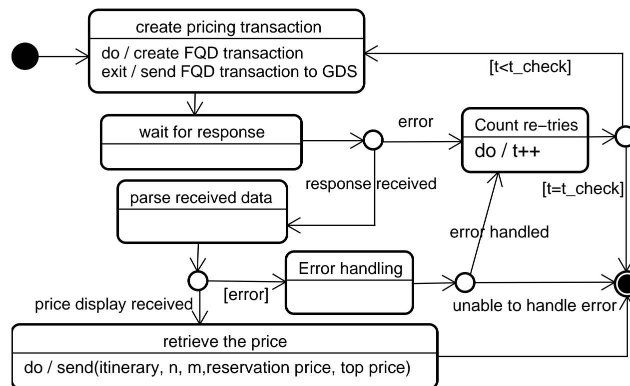

Fig. 5. FlightOffer Agent's "Price retrieval" substate statechart diagram

future, it will be possible to make *Shop Agent*s able to adjust their behavior. These (and an initial implementation of the system—which will involve modification of the existing code-base of the original system and which will require obtaining actual access to the GDS, which we hope to be able to secure in the near future) are also directions of our current research. We will report on our progress in subsequent papers.

References

1. Bădică, C., Badita, A., Ganzha, M., Iordache, A., Paprzycki M.: Implementing Rule-based Mechanisms for Agent-based Price Negotiations. In: Proceedings of the SAC'2005 Conference (in press)
2. Bădică, C., Ganzha, M., Paprzycki, M., Pîrvănescu, A.: Combining Rule-Based and Plug-in Components in Agents for Flexible Dynamic Negotiations. In: M. Pěchouček, P. Petta, and L.Z. Varga (Eds.): *Proceedings of CEEMAS'05*, Budapest, Hungary. LNAI 3690, Springer-Verlag, pp.555-558, 2005.
3. Bădică, C., Ganzha, M., Paprzycki, M., Pîrvănescu, A.: Experimenting With a Multi-Agent E-Commerce Environment. In: V. Malyshkin (Ed.): *Proceedings of PaCT'2005*, Krasnoyarsk, Russia. LNCS 3606, Springer-Verlag, pp.393-402, 2005.
4. Bădică, C., Ganzha, M., Paprzycki, M.: UML Models of Agents in a Multi-Agent E-Commerce System. In: *Proceedings of the IEEE Conference of E-Business Engineering, ICEBE 2005*, Beijing, China. IEEE Computer Society Press, Los Alamitos, CA, pp.56-61, 2005.
5. Bădică, C., Bădiţă, A., Ganzha, M., Iordache, A., Parzycki, M.: Rule-Based Framework for Automated Negotiation: Initial Implementation. In: *Proceedings 1st Conference on Rules and Rule Markup Languages for the Semantic Web, RuleML'2005*, Galway, Ireland. Lecture Notes in Computer Science 3791, Springer-Verlag, pp.193-198, 2005.
6. Bartolini, C., Preist, C., Jennings, N.R.: Architecting for Reuse: A Software Framework for Automated Negotiation. In: *Proceedings of AOSE'2002: Int. Workshop on Agent-Oriented Software Engineering*, Bologna, Italy, LNCS 2585, Springer Verlag, pp.88-100, 2002.
7. Bartolini, C., Preist, C., Jennings, N.R.: A Software Framework for Automated Negotiation. In: *Proceedings of SELMAS'2004*. LNCS 3390, Springer-Verlag, pp.213-235, 2005.

8. Ganzha, M., Paprzycki, M., Pîrvănescu, A., Bădică, C, Abraham, A.: JADE-based Multi-Agent E-commerce Environment: Initial Implementation, In: *Analele Universităţii din Timişoara, Seria Matematică-Informatică*, 2005 (to appear).
9. JADE: Java Agent Development Framework. See http://jade.cselt.it.
10. Kowalczyk, R., Ulieru, M., Unland, R.: Integrating Mobile and Intelligent Agents in Advanced E-commerce: A Survey. In: *Agent Technologies, Infrastructures, Tools, and Applications for E-Services, Proceedings NODe'2002 Agent-Related Workshops*, Erfurt, Germany. LNAI 2592, Springer-Verlag, pp.295-313, 2002.
11. Maes, P., Guttman, R.H., Moukas, A.G.: Agents that Buy and Sell: Transforming Commerce as we Know It. In *Communications of the ACM*, Vol.42, No.3, pp.81-91, 1999.
12. Trastour, D., Bartolini, C., Preist, C.: Semantic Web Support for the Business-to-Business E-Commerce Lifecycle. In: *Proceedings of the WWW'02: International World Wide Web Conference*, Hawaii, USA. ACM Press, New York, USA, pp.89-98, 2002.
13. Wooldridge, M.: *An Introduction to MultiAgent Systems*, John Wiley & Sons, 2002.