Analele Universităţii din Timişoara
Vol. XXXXIX, Fasc. 2, 2001
Seria Matematică–Informatică

# JADE BASED MULTI-AGENT E-COMMERCE ENVIRONMENT: INITIAL IMPLEMENTATION

Presented at 6th Int. Symposium SYNASC04, Timişoara, Romania

Maria Ganzha *, Marcin Paprzycki **, Amalia Pirvănescu ***,
Costin Bădică ****, Ajith Abraham *****

**Abstract.** Recent advances in software engineering, business process management and computational intelligence resulted in methods and techniques for developing advanced e-commerce applications as well as supporting automating e-commerce business processes. Despite this fact, up to now, the most successful e-commerce systems are still based on humans to make the most important decisions in various activities within an e-business transaction. In this context, development of automatic negotiations is one of the most important research issues. While, depending on the type of the transaction, different negotiation procedures could be utilized, only few proposed frameworks are generic and flexible enough to handle multiple scenarios. On the other hand, agent technology is often claimed to be the best approach for automating e-commerce business processes (including price negotiations). However, it is difficult to find successful large-scale agent-based e-commerce applications to confirm this claim. This paper presents negotiating agents that change their negotiation protocol and strategy through dynamic loading of negotiation modules. These, as well as other agents of different types and playing different roles have been implemented to in-

*Gizycko Private Higher Educational Institute, Department of Informatics, ul. Daszynskiego 9, 11-500 Gizycko, Poland ganzha@pwsz.net

**Oklahoma State University, Computer Science Department Tulsa, OK, 74106, USA and SWPS, Computer Science ul. Chodakowska 19/31, 03-815 Warszawa, Poland marcin@cs.okstate.edu

*** SoftExpert SRL, Str.Vasile Conta, bl.U25, Craiova, Romania, amaliap@soft-expert.com

**** University of Craiova, Software Engineering Department, Bvd.Decebal 107, Craiova, RO-200440, Romania, badica_costin@software.ucv.ro

*****School of Computer Science and Engineering, Chung Ang University, Seoul, Korea, ajith.abraham@ieee.org

teract within an abstract e-commerce environment. Description of this complete e-commerce system is the goal of this paper.

**Key words:** multi-agent systems, e-commerce, automated negotiations

**AMS Subject Classification:** 68T99

**1. Introduction.** Already long time ago, the six main stages of consumer buying behavior have been conceptualized: need identification, product brokering, merchant brokering, negotiation, payment and delivery, service and evaluation (Howard, 1969). These stages have not changed when commerce moved to the Internet and became e-commerce. To be able to support consumers, e-commerce research involves all aspects of complex processes, spanning areas that cover business modeling, information technology and social and legal aspects (Laudon, 2004). Currently, e-commerce systems operate following the principle "to select and to accept choices". Thus users can browse through catalogues of needed goods (tickets, films, knifes CD's and so on) and utilize them to make purchasing decisions. There exist systems, which support users during the product or/and merchant brokering stages of the buying process (for instance www.shopping.com). At the same time, the most interesting part of product buying: automated negotiations, is not properly supported within B2C environments (however, limited use of automated negotiations within B2B is known to exist for some time now, and apparently was partially responsible for some of the more spectacular crashes on the New York stock market in the late 1990th).

It is exactly in the context of agent-based automated price negotiations, where a lot of research activity can be observed. Here, negotiation is a process by which group of agents communicate with another to try and come to a mutually acceptable agreement on some matter. While there exist many definitions of agents, for the purpose of this paper we will define them as: encapsulated computer programs, situated in an environment, and capable of flexible, autonomous actions focused on meeting their design objectives. For such agents, e-commerce is considered to be one of the paradigmatic application areas. As described in the recent survey (Kowalczyk, 2003), multi-agent technology (involving intelligent mobile agents) should have an important economical impact, by bringing efficiency to businesses (and thus improving their profitability), as well as benefiting

individual users (e.g. by assuring "price-optimality" of purchases). It is exactly in the latter context where, multi-agent systems are expected to assure price fairness and reduce the negotiation time.

While a large amount of work has been devoted to study agent-negotiations, the current state of the art and practice is still rather unsatisfactory. Our research indicates that most currently existing automated trading systems are not robust enough to become the foundation of the next generation of e-commerce. Claim that there is **a lot more work to be done** to develop real-life agent-based support for e-commerce is further supported by the fact that it is almost impossible to point out to an existing large-scale implementation of an e-commerce agent system. While a number of possible reasons for this situation have been suggested (see, for instance, (Paprzycki, 2003)), one of them has been recently dispelled. It was shown that modern agent environments (e.g. JADE) can easily scale to 1500 agents and 300000 messages (Chmiel, 2004b). Since these results have been obtained on a set of 8 antiquated Sun workstations, it is easy to extrapolate the true scalability of JADE on modern computers and thus **it is possible to build and experiment with large-scale agent systems**. Secondly, recently new methods and techniques of software engineering, business process management and computational intelligence in support of the development of advanced e-commerce applications have been proposed. For example, we now have generic software frameworks for automated negotiation (Bartolini, 2003), Semantic Web support (ontologies and semantic Web services), at least in theory, for the full B2B e-commerce life cycle, and finally, multi-agent solutions for business process management.

Therefore, we have set up a goal of developing, implementing and experiment with a large-scale agent-based e-commerce system. Since this is a long-term undertaking, at this stage our focus is on creating a system with a multitude of agents that play variety of roles and interact with each-other (system skeleton). Currently, we follow and combine our earlier work in two areas. This paper is a follow up to (Parakh, 2003)where we proposed a system in which agents can operate according to different business models including auctions, reverse auctions, trading, e-sales etc. This was to be made possible by constructing agents out of independently plugable, loaded remotely on-demand modules (Paprzycki, 2004). Second, we have implemented a simplistic skeleton for an e-commerce simulation (Chmiel, 2004a) and proceeded to create a unified e-commerce environment witch supports automatic negotiations in the case of one-to-many negotiation process and experimented with the system running on two networked

computers. This paper is a summary of our work starting from (Parakh, 2003) and ending with (Paprzycki, 2004). In the paper we, first, discuss the most important issues involved in negotiations. In Section 3 we introduce the top level description of our system and agents that populate it. We follow (in Section 4) with a summary of implementation-specific information (including UML diagrams conceptualizing most important agents) as well as an example illustrating system's work. We conclude with the research agenda of our team.

**2. Negotiations.** Let us start from conceptualizing negotiations both in theory and in the context of development of negotiating agents. Negotiation is a method for coordination and conflict resolution. Conflict can be in the form of resolving goal disparities in planning, resolving constraints in resource allocation, and resolving task inconsistencies in determining organizational structure. As indicated above, in the context of the Internet it will be autonomous agents that will be involved in negotiations. Overall, research on agent-mediated negotiation can be divided into approaches based on game theory or artificial intelligence.

Game-theoretic approach is directed towards developing optimization algorithms, AuctionBot etc. This approach takes into account both cooperative and non-cooperative agents. In the case of cooperative agents, the problem space can be divided efficiently between all agents. In some cases, agents form a team, each having its own local goals and a team goal. When a conflict arises, the team members can negotiate about the matter, and give evidence supporting their stance. However, the team members are assumed to have total knowledge about the system. In non-cooperative approach, theories like Nash equilibrium are applied to the bargaining problem to find the optimum solution for agents. Their main drawback is that, due to their roots being highly theoretical, they provide us with highly abstract models that assume unrealistic properties of the game: e.g. agents are assumed to have the entire common knowledge and unbounded rationality; in addition, they are assumed to have unlimited computation power and indefinite negotiation time. This makes such approaches impossible to implement. Nevertheless, the extensive research carried out in this field helped develop other theories. An example of game-theoretic approach is the work on modeling and implementing techniques for agents participating in auctions e.g. Dutch auction, English auction, Vickery auction, etc.

Artificial Intelligence based approaches utilizes trading heuristics for different market mechanisms. AI techniques focus on the negotiation process rather than the outcome of the negotiation. Mostly learning approaches like decision trees, Q-learning and evolutionary algorithms have been used to improve bargaining strategies. The agents used are adaptable, realistic and sociable. AI theories are based on realistic assumptions of an imperfect world with bounded rationality and limited knowledge of the world. Since agents do not know a'priori what type of agents they are interacting with, this creates conflicts between agents. Simulations of agents learning different bargaining strategies, through genetic algorithms, have been carried out and their results have been quite promising.

When considering the practical aspects of designing multi-agent negotiations, the negotiation protocol, negotiation objects and the reasoning models need to be taken into account. Let us consider each of them separately:

**Negotiation protocol** consists of a set of rules that govern the interaction among agents. Some examples of the rules are permissible types of participants: negotiators, third parties; negotiation states: accepting bids, negotiation closed; valid actions of the participant in particular states. Typical negotiation protocols are

1. *FIPA English Auction Interaction Protocol*: the auctioneer seeks to find the market price of a good by initially proposing a price below that of the expected market value and then gradually raising it. Each time the price is announced, the auctioneer waits to see if any buyers will signal their willingness to pay the proposed price. As soon as one buyer indicates that it will accept the price, the auctioneer issues a new call for bids with a higher price. The auction continues until no buyers are prepared to pay the proposed price, when the auction ends. Commodity is sold only if the last accepted price exceeds the auctioneer's (privately known) reservation price.
2. *FIPA Dutch Auction Interaction Protocol*: the auctioneer attempts to find the market price for a good by starting bidding at a price higher than the expected market value, then progressively reducing the price until one of the buyers accepts the price. The rate of reduction of the price is up to the auctioneer and usually a reserve minimal price is involved.

**Negotiation objects** are ranges of issues over which agreement must be reached. These depend on the environment and can be different for different environments.

**Reasoning model** is the apparatus that participants employ to act in line with the negotiation protocol in order to achieve their negotiation objectives. Reasoning models are the thinking machines behind the process of carrying out the negotiation. Reasoning model is a mechanism by which the next counter-offer is calculated during negotiation so that the price fits into the goal of buying some good within the specified range. Some of the strategies developed so far are argumentation, persuasion and heuristics-based. It can be safely assumed that the kind of reasoning model chosen depends on both the protocol and the negotiation object. Moreover, the complexity of former depends on latter.

This way of conceptualizing negotiations results in agents that have to consist of the following three main modules:

**Communication module** — responsible for communication between the agents in a common, understandable way. Since FIPA, as a main agent standardization body, is supporting a number of communication technologies (e.g. the ACL communication language) (FIPA, 1999), we will assume that functions of this module are obvious enough for it to be omitted from further considerations. This is especially so, since this module is a static one.

**Protocol module** — contains general rules of negotiation; when an agent initiates negotiation, on the basis of negotiator table and/or meta-negotiations it finds out which negotiation protocol can be used and dynamically loads the correct module (from the user's local machine or any agent server, e.g. the nearest one).

**Strategy module** — is designed to apply the proper reasoning module so that the negotiation ends in a success. The reasoning model contains policies, which are a set of goals, actions and action rules (triggers). In order to decide which reasoning model to use, the agent uses the mapping table that records the earlier history of the transactions, which the agents made with the seller. It also lists what was the success rate for the transactions. If an entry is not found, then the agent resorts to a default strategy. To keep the agent lightweight it

will carry only a part of the table, containing the "sites" most often negotiated with, while the remaining part of the table will be kept on the user's machine. In case the user's machine is off-line the default strategy will be used, again (since this will involve only sites that are visited rarely, such an occurrence should not be detrimental to the overall behavior of our system). The strategy also depends on which protocol module has been chosen. In fact, the strategy is almost restricted by that decision. For example, a strategy used for argumentation cannot be used if the protocol is an auction protocol. It should be noted that it is possible define two levels of strategies: coarse and granular. The granular strategies are tit-for-tat, conceder etc, while the coarse strategies are heuristics, using knowledge base etc. Their usage will depend on the negotiation context and the details of the selection strategy will be further studied. We expect that a special (rule-based) strategy selection meta-module will be developed. This module can, be split between the agent, that will carry the most important rules, and the user local machine that will contain the complete module. In this way the agent will be able to get involved in negotiations even if its contact with the user's machine will be impossible to establish.

The main problem in designing mobile e-commerce agents involved in negotiations is the fact that we cannot assume that agent will know a'priori which negotiation protocol will be used at a given site (we assume here, that the same site, for the same product can use at different times different negotiating protocols). This being the case, "buyer agents" would have to carry with them a large load of negotiation protocols and strategies and this would impede their mobility. This is particularly the case when sophisticated negotiation strategies are to be used (for all practical purposes we have to assume that the more sophisticated strategy, the larger the strategy module that implements it). Thus we are dealing with a clear case of the "no free lunch theorem" agents can be either mobile and lightweight or "intelligent." To address this problem we followed an approach where each agent consists of the skeleton and three separate modules: communication, strategy and protocol. The mobile part of an agent consists of the skeleton and the communication module (though it can be assumed that, for instance, multi-lingual communication modules can also be downloadable), while the protocol and strategy modules are downloaded after agent arrives at the e-marketplace and establishes which protocol is to be used.

Then, on the basis of the history of interaction with that site, the product to be purchased, the protocol used etc., an appropriate strategy module is also loaded. Observe, that the protocol module can be public - as its content has to be the same for all agents participating, for instance, in an English auction, and thus can be loaded form "any server". It is only the strategy module that is proprietary and has to be downloaded from the "home-base". The proposed design of negotiating agents is illustrated in Figure 1.

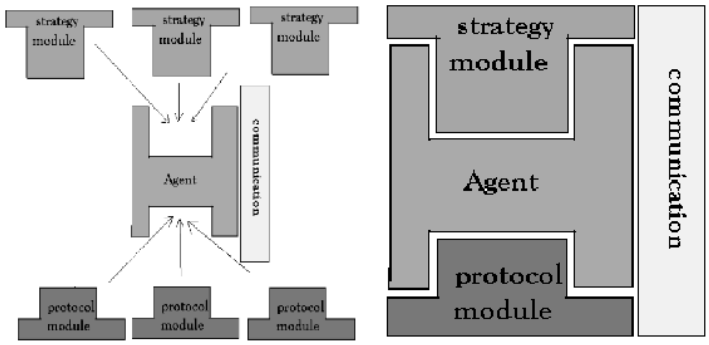


**Fig. 1.** Agents consisting of downloadable modules.

Since the aim of our work was to test-implement an agent system, and *not* to deal with specific negotiation strategies, we have implemented two extremely simple ones. The buyer/seller increments the price by 10 in, what we named, the *Heuristics Reasoning* module and by 20 in, what we named, the *Argumentation Reasoning* module (obviously, these names only indicate negotiation strategies that could have been used here). Currently we load these two modules randomly as a proof of concept.

**3. System Description.** In our work we aim at implementing a multi-agent e-commerce environment to help carrying out experiments with real-world e-commerce scenarios. Note that sometimes we use the word *environment* rather than system to point out the exploratory nature of our work; i.e. we are more interested in creating an artificial agent world in which e-commerce agents perform variety of functions typically involved in e-commerce, rather than developing a particular e-commerce system tar-

geted to solve a specific business problem that uses a limited number of application-specific agents.

**3.1. System Architecture.** Our e-commerce model extends and builds on the e-commerce structures presented in (Galant, 2000), (Chmiel, 2004a) and (Paprzycki, 2004). Basically, our environment acts as a distributed marketplace that hosts e-shops and allows e-clients to visit them and purchase products. Clients have the option to negotiate with the shops, to bid for products and to choose the shop from which to make a purchase. Conversely, shops may be approached "instantly" by multiple clients and consequently, through auction-type mechanisms, have an option to choose the buyer. At this stage the system is under development and has a number of limitations. (1) Only two auction protocols have been implemented. (2) The two strategy modules are only to show that such modules can be downloaded upon request. (3) We have only shops are allowed to advertise their products (clients cannot advertise goods they are seeking). (4) While various strategies could be employed to decide where to buy from (e.g. the best price, the safest offer, the most trusted offer, etc.), we are using only the best negotiated price. (5) We do not worry about system scalability and thus work with single agents (e.g. the *CIC agent*) that can become a bottleneck when the system size increases (Chmiel, 2004b). Obviously, these are serious restriction and we plan to address them in the near future.

Shops and clients are created through a GUI interface that links users (buyers and sellers) with their *Personal Agents*. However, these agents are in many ways spurious for the operation of the system. More precisely, a *Personal Agent* is considered to be a true representative of the user that resides on her machine and represents her interests in all aspects of e-life. Thus, in the context of our system its role is restricted to creation of *Client* / *Shop* agents that will be a part of the e-marketplace; and therefore the *GUI* and *Personal* agents are omitted from further considerations. The top level conceptual architecture of the system illustrating proposed types of agents and their interactions in a particular configuration is shown in Figure 2. Let us now describe each agent appearing in that figure and their respective functionalities.

A *Client agent* (*CA*) is created by the *Personal agent* to act within the marketplace on behalf of a a user that attempts at buying something. Similarly, a *Shop agent* represents user who plans to sell something within the e-marketplace. After being created both *Shop* and *Client* agents register with the *CIC agent* to be able to operate within the marketplace. Returning

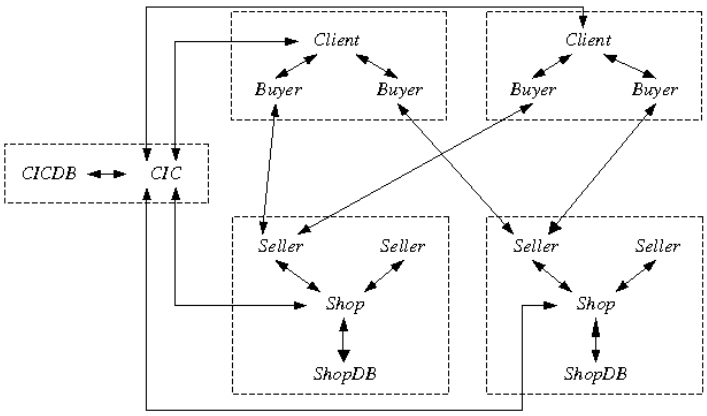


**Fig. 2.** The conceptual architecture of our e-commerce environment (two-client; two-store version).

agents will receive their existing IDs. In this way we provide support for the future goal of agent behavior adaptability. Here, agents in the system are able to recognize status of their counterparts and differentiate their behavior depending if this is a "returning" or a "new" agent that they interact with.

There is only one *Client Information Center* (*CIC*) agent in the system. It is responsible for storing, managing and providing information about all "participants" existing in the system. To be able to participate in the marketplace all *Shop* and *Client* agents must register with the *CIC agent*, which stores information in the *Client Information Database* (*CICDB*). The *CICDB* combines the function of *client registry*, by storing information about and unique IDs for all users and of *yellow pages*, by storing information about of all shops known in the marketplace. Thus *Client agents* (new and returning) communicate with the *CIC agent* to find out which stores are available in the system at any given time. In this way we are (i) following the general philosophy of agent system development, where each function is embodied in an agent and (ii) utilizing the publisher-subscriber mechanism based on distributed object oriented systems. Furthermore, this approach provides us with a simple mechanism of correctly handling the concurrent accesses to a shared repository without

having to deal with typical problems of mutual exclusion etc. Actually, all these problems are automatically handled by JADE's agent communication service.

A *Client agent* is created for each customer that is using the system. Each *Client agent* creates an appropriate number of "slave" negotiation agents with the "buyer role" (*Buyer agents* hereafter). One *Buyer agent* is created for each store, within the marketplace, selling sought goods. Operation of the *Client agent* is depicted in Figure 3; further details can be also found in the next section.

On the supply side, a single *Shop agent* is created for each merchant in the system and it is responsible for creating a slave negotiation agent with the "seller role" (*Seller agent* hereafter) for each product sold by the merchant within her e-store. Operation of the *Shop agent* is depicted in Figure 4, while the *Seller agent* is conceptualized in Figure 5; further details can be also found in the next section.

Finally, *Database agents* are responsible for performing all database operations (updates and queries). For each database in the system we create one database agent. In this way we decouple the actual database management activities from the rest of the system (i.e. the database management system can be modified in any way without affecting the agent side of the system and vice-versa). Currently, there are two databases in the system: a single *CICDB* database (operated by the *CICDB* agent) containing the information about clients, shops and product catalogues, and a single *Shop Database* (*ShopDB*) operated by the *ShopDB agent* storing information about sales and available supplies for each merchant registered within the system. Since operation the database agents is obvious, and in a way auxiliary to the system, we omit them form further discussions, including UML diagrams of other agents.

The central part of the system operation is comprised by price negotiations. *Buyer agents* negotiate price with *Seller agents*. For this purpose *Buyer agents* migrate to the e-stores known by the *CIC agent* to carry sought after commodity. In case of multiple *Buyer agents* attempting at purchasing the same item, they may compete in an auction. Results of price negotiations are send by the *Shop agent* to the *Client agent* that decides where to attempt at making a purchase. Note that the system is fully asynchronous and thus an attempt at making a purchase does not have to result in a success as by the time the offer is made other *Buyer agents* may have already purchased the last available item. In this way we proceed with a n e-commerce model similar to the airline ticket reservation

where until an actual purchase is made item is reserved, but may not be available at a later time.

Let us now illustrate operation of the system through a simplified usage scenario.

**3.2. Usage scenario.** A session with the system starts with the merchants and customers creating *Shop* and *Client* agents via the GUI provided by their *Personal agents*. A *Client agent* obtains name of the product of interest (possibly multiple products of interest) and a reserve price (see Figure 3). A *Shop agent* obtains a list of pairs (product, reserve price) and

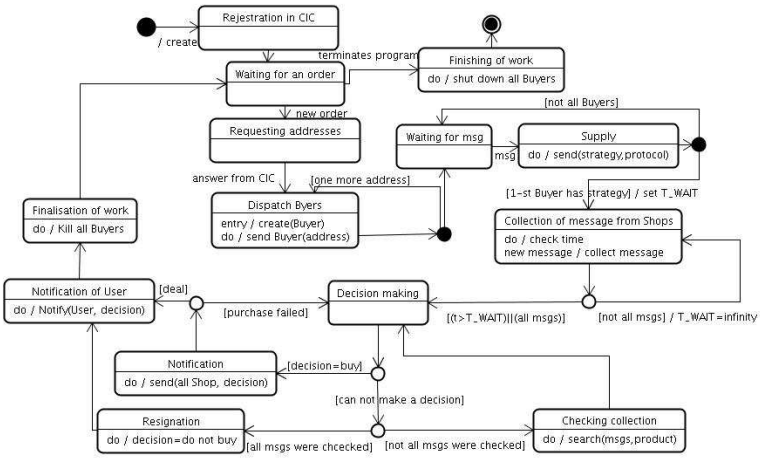


**Fig. 3.** UML statechart diagram of the *Client agent*.

the negotiation protocol that is to be used for interactions with incoming *Buyer agents* (see Figure 4).

The reserve price of a *Client agent* is the maximum price it agrees to pay for the product. The reserve price of the *Shop agent* is the minimum price at which it is to agree to sell a specified product. Each *Shop agent* creates *Seller agents*, one *Seller agent* for each product sold (see Figure 5). *Seller agents* await incoming *Buyer agents* interested in buying their products and upon their arrival engage in negotiations with them (see Figure 5). Let us now describe what happens in the marketplace after a customer has made a purchase request, until a request is completed (Figures 3, 4 and 5 further illustrate the process).

1. As specified above, a *Client* agent registers with the *CIC* agent. It obtains a new ID if it is a new *Client* or recovers its original ID if it is a

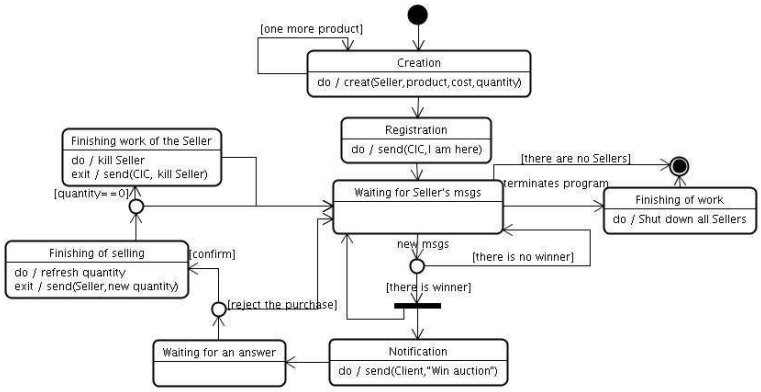


**Fig. 4.** UML statechart diagram of the *Shop agent*.

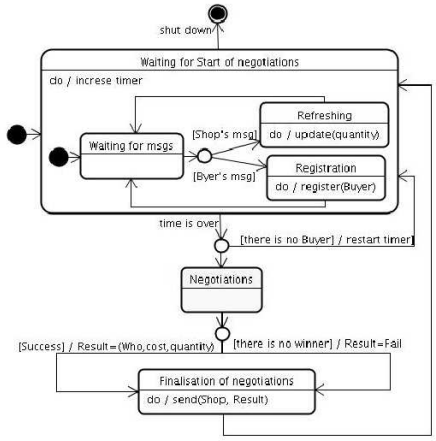


**Fig. 5.** UML statechart diagram of the *Seller agent*.

returning *Client*. The information that an agent with a given ID is active in the marketplace is stored in the *CICDB* database (this step involves interactions between the *CIC* agent and the *CICDB* agent).

2. The *Client agent* queries the *CIC agent* to obtain the list of Shop agents selling the product it is expected to purchase. For each Shop agent on this list it creates a *Buyer agent* to negotiate conditions of purchase.

3. *Buyer agents* migrate to *Shop agent* sites and query *Shop agents* about the negotiation protocol used in a given e-store and which Seller agent they should negotiate with. Then, *Buyer agents* dynamically load

appropriate negotiation protocols (and, in the future, strategy modules (Parakh, 2003) from *Client agents* and subscribe to the designated *Seller agent*, waiting for the negotiation process to start.

4. The *Seller agent* checks periodically (currently in 1 minute intervals) for the set of *Buyer agents* that subscribed to bid for its product. If this set is nonempty, it starts an auction. At the end of an auction a *Seller agent* informs the *Shop agent* about the winner. *Shop agents* are recording the auctions winners and inform the corresponding *Client agents* that a purchase is possible. The decision to buy and where to buy from is made by the *Client agent*, depending on the winning offers made by the *Shop agents*.

5. The *Client agent* obtains results of auctions from the *Shop agents*, finds the best negotiated price and makes an attempt at purchasing the product by informing the corresponding *Shop agent* about the decision to buy. When the confirmation is received, it informs the customer about the result of its request: success of failure of purchase, the shop where the purchase was made from and the negotiated price.

## 4. Implementation and Experiment.

**4.1. System Implementation.** The current implementation of the proposed environment has been made within the JADE 3.3 agent platform (JADE, 2003). The main reason for this selection was the fact that JADE is one of the best modern agent environments. JADE is open-source, it is FIPA compliant and runs on a variety of operating systems including Windows and Linux. Furthermore, in (Chmiel, 2004b) we have observed its very good scalability.

JADE architecture, consisting of a platform within which agents "live" and containers, where agents "reside" matches well with our requirements. Negotiations between *Seller* and *Buyer* agents take place inside of JADE containers. There is one Main container that hosts the CIC agent. Users (customers and merchants) can create as many containers they need to hold their *Client* and *Shop* agents (e.g. one container for each e-store). *Buyer agents* created by *Client agents* use JADE mobile agent technology to migrate to the *Shop agent* containers to engage in negotiations.

Figure 6 presents a mapping of our conceptual architecture from Figure 2 onto JADE. In particular, this diagram shows two machines running *Personal*, *Shop*, *Client*, *Buyer* and *Seller* agents, highlighting also JADE containers involved (*Main* container in the upper half of the figure, and

*Container-1* container in the lower half, separated by the horizontal dotted line). Here, continuous lines denote agent creation. *Personal agent* $P_1$ represents user u1 (shopper) and creates two *Client agents*: $C_{11}$ and $C_{12}$ to purchase items $p_1$ and $p_2$ respectively. Single-arrow dashed lines denote agent migration. *Buyer agent* $B_{111}$ migrates from the *Main* container to the *Container-1* container. Double-arrow continuous lines denote negotiations. *Seller agent* $S_{11}$ negotiates with *Buyer agents* $B_{111}$ and $B_{211}$ for product $p_1$. The sample scenario from Figure 6 is discussed in more detail in section 3.2.

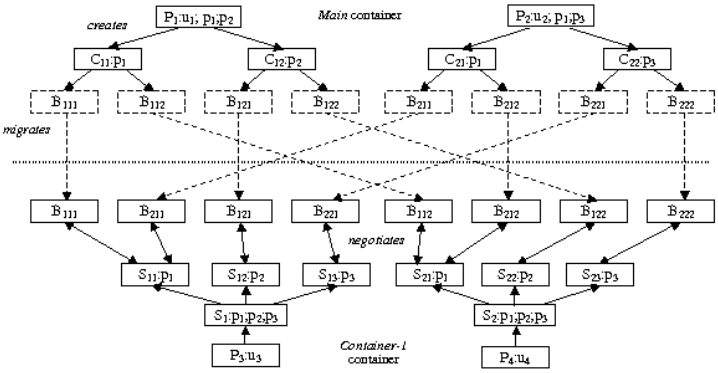


**Fig. 6.** Mapping the conceptual architecture of the system to JADE.

The current implementation is based on several Java classes organized into the following categories:

*Agent classes.* They are used for describing the agent types. In this category we have utilized: class *ClientAgent* that implements *Client agents*, class *ShopAgent* that implements *Shop agents*, class *CIC* that implements *CIC agents*, class *NegoAgent* that implements negotiating *Buyer* and *Seller* agents, classes *CICDatabaseAgent* and *ShopDatabaseAgent* that implement em Client Information Database and *Shop Database* agents and class *PersonalAgent* that implements *Personal* agents. An agent is implemented in JADE by extending the provided *Agent* base class and overriding the default implementation of the methods that are automatically invoked by the platform during the agent lifecycle, including *setup()* and *takedown()*. In our implementation all agent classes extend the *Agent* base class ex-

cept the *PersonalAgent* class that extends the *GuiAgent* class (provided by JADE).

*Agent activity classes*, also called behaviors. They are used for describing the activities performed by agents in the system. A behavior is an abstraction that represents an atomic activity performed by an agent. In our implementation we have used local classes for defining behaviors that describe the agent responses to FIPA messages, like INFORM and SUBSCRIBE. There are also two global classes for defining auction initiators and auction participants: class *AuctionInitiator* and class *AuctionParticipant*. Note that in the current implementation our agents are negotiating only using FIPA defined English and Dutch auction schemas, but the approach can be easily extended to other automated negotiation models. A behavior is implemented in JADE by extending the provided *Behaviour* abstract base class. The class *Behaviour* is the root of a class hierarchy abstracting various agent behavior types. We have found useful to use the class *CyclicBehaviour* as the base class for the class *AuctionParticipant* and the class *FSMBehaviour* as the base class for the class *AuctionInitiator*. As concerning the definition of the responses to FIPA messages, we have extended the class *CyclicBehaviour*.

*Reasoning classes*. They are used for the implementation of the various reasoning models employed by the negotiation agents; see (Paprzycki, 2004) for more details on the model of negotiation agents. Our implementation supports agents that dynamically load their negotiation protocols and reasoning modules. The implementation combines the Factory design pattern (Cooper, 2000) and dynamical loading of Java classes (Paprzycki, 2004).

*Ontology classes*. They are necessary for implementing the agent communication semantics using concepts and relations. The current implementation uses an extremely simple ontology that defines a single concept for describing *Client* and *Shop* preferences including prices, product names and negotiation protocols.

*Other classes*. Here including there is class *PersonalAgentGUI* for the implementation of the graphical user interface of *Personal* agents.

In our system, agent communication is implemented using FIPA ACL messages (FIPA, 1999). We have used the following messages: SUBSCRIBE, REQUEST, INFORM, FAILURE, CFP, PROPOSE, ACCEPT-PROPOSAL, REJECT-PROPOSAL, REFUSE.

SUBSCRIBE messages are used by the *Shop* and *Client* agents to register with the *CIC agent* and for the *Buyer agents* to register (to par-

ticipate in auctions) with the *Seller agent*. REQUEST messages are used by *Client* agents to query the *CIC agent* about what shops are selling a specific product and for *Client agents* to ask the *Shop agent* for a final confirmation of a transaction. INFORM messages are used as responses to SUBSCRIBE or REQUEST messages. For example, after subscribing to the *CIC agent*, a *Client agent* will get an INFORM message that contains its ID, or after requesting the names of the shops that sell a specific product, a *Client agent* will receive a list of the Shop agent IDs in an INFORM message. Buyer agents are using FAILURE messages to inform the master *Client agents* about the unsuccessful result of an auction. Finally, CFP, PROPOSE, ACCEPT-PROPOSAL, REJECT-PROPOSAL and REFUSE messages are being used by negotiating agents.

**4.2. Running the System.** For the experiment, we set up JADE on 4 computers. On the first computer, the *Main* container is initialized. On the remaining computers, containers *Container-1,2,3* that are linked with the Main container on the first computer were started. Both the *CIC* and the *CICDB* agents were created by default within the *Main* container, while the *Shop* and the *ShopDB* agents were instantiated in *Container-1,2,3* containers.

In this experiment we have used a simple scenario with 4 merchants – *Shop*0, *Shop*1, *Shop*2, *Shop*3 and 4 customers — *Client*0, *Client*1, *Client*2, *Client*3. Customer *Client*0 is seeking products $p_1$ and $p_4$, customer *Client*1 is seeking products $p_1$ and $p_3$, customer *Client*2 is seeking products $p_2$ and $p_4$ while customer *Client*3 is seeking products $p_2$ and $p_3$. In order to enable price competition, we have set the experiment in such a way that some customers are seeking a common product. Merchants and customers used *Personal* agents running in *all* containers to create four *Shop* and four *Client* agents (Figure 7.

The process of starting *Shop agents* involved their registration with the *CIC agent*. Hereafter, for each product offered, a *Seller* agent was created. So there are two *Seller agents* in every container. Similarly, starting *Client agents* involved their registration with the *CIC agent*, followed by the "search" of *Shop agents* that sell sought products and creation of a *Buyer agent* for every *Shop agent* found. Therefore, finally, 16 *Buyer agents* were created (4 *Client agents* send 4 *Buyer agents* each to 4 e-stores).

At this stage of the experiment, *Buyer agents* move to *all* appropriate containers and register with appropriate *Shop agents*. As a result of message exchanges (Figure 8, bottom panel) negotiation protocol is identified

and negotiation modules loaded by *Buyer agents*. Next, *Buyer agents* subscribe with *Seller agents* that sell sought products. *Seller agents* react to a timer that periodically triggers start of auctions with subscribed *Buyer agents* (an English auction in this experiment). Thus we have 8 auctions - 2 for selling each product $p_1$, $p_2$, $p_3$ and $p_4$. Note that because both customers *Client*0 and *Client*1 are requesting product $p_1$, *Buyer agents* $B_{001}$, $B_{011}$, and respectively $B_{101}$ and $B_{111}$ are competing for buying $p_1$ from *Seller agents* within *Shop*0 and respectively *Shop*1.

Figure 8 presents message exchanges captured in the experiment with the help of a JADE provided *sniffer agent*. This figure shows: i) *Shop* and *Client* agents subscribing to the *CIC agent*; ii) *Client agents* asking the *CIC agent* where to find out a specific product; iii) *Buyer agents* subscribing to *Seller agents* for negotiation; iv) the start of a negotiation when a *Seller agent* issues a call-for-proposal request to a *Buyer agent*.

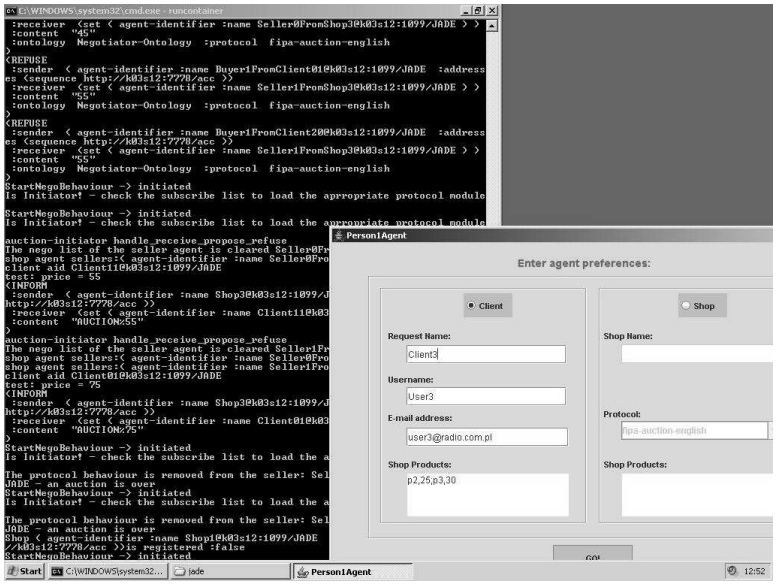


**Fig. 7.** Screen captures showing our system in action.

Figure 9 presents finalization stage of negotiations — notification of User about result of negotiations.

**5. Conclusions.** In this paper we have presented basic features of an e-commerce modeling agent system that we are currently developing. At

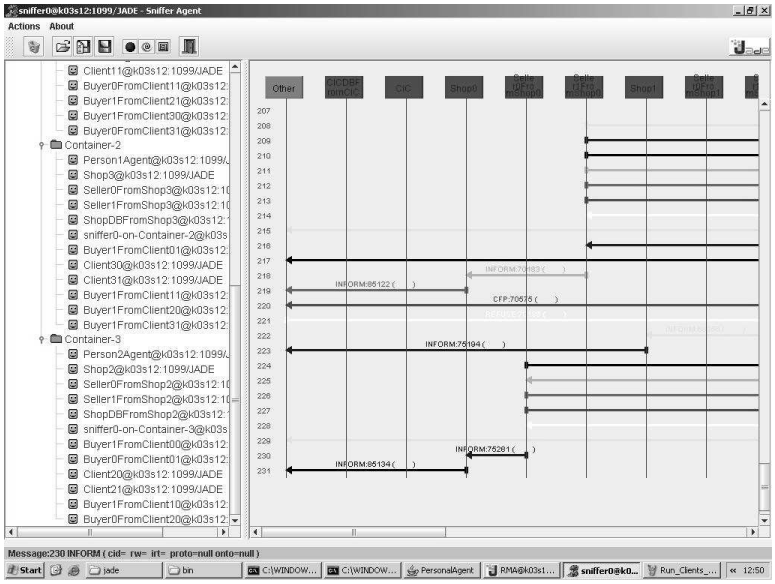


**Fig. 8.** Screen captures showing our system in action.

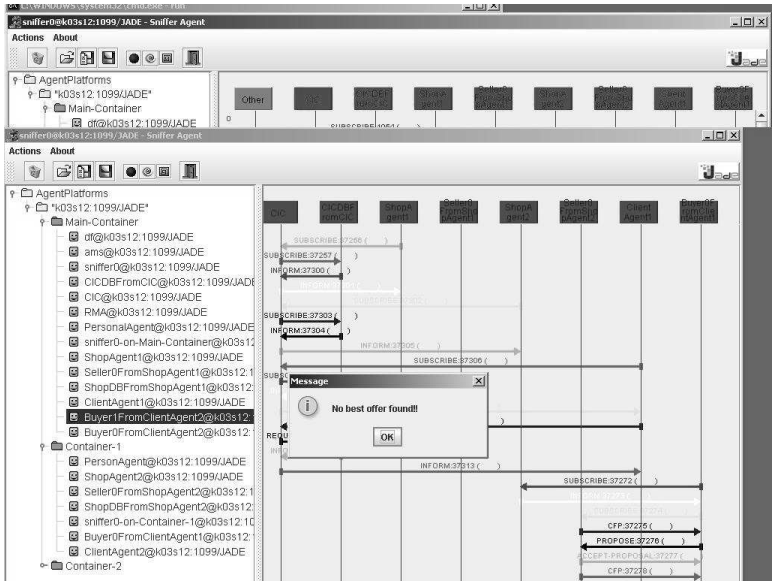


**Fig. 9.** Screen captures showing finish of action.

this stage its capabilities are limited, but we have already considered a number of future research directions that we plan to pursue.

(1) Currently price is the only factor determining purchase. Other factors, such as the speed of delivery, trust, history of involvement with a given merchant should be also taken into account. Overall, we plan to combine the framework for multi-section contract formation discussed in (Karp, 2003) with the software framework for automated negotiation presented in (Bartolini, 2003) and results on negotiation framework targeted to multiple buyers and sellers reported in (Srivastava, 2003).

(2) Currently only shops can advertise available commodities. We plan to extend this to the scenario in which also clients will be able to advertise their "needs".

(3) We will complete implementation of negotiation protocols. Currently we have implemented Dutch and English auctions. We will add the remaining, FIPA defined auction protocols as well as simpler strategies such as: fixed pricing, fixed pricing with a discount for volume purchases, special prices for returning customers etc.

(4) Currently we have been running our experiments on two computers, where all seller data is located in a single database. In the near future we will experiment with a larger number of computers and adjust them so that each store has a separate database. More generally, we plan to experiment with a large number of computers, clients, shops, commodities and negotiation protocols. The aim of these experiments is to establish scalability of the systems as well as locate its performance bottlenecks.

(5) Our system works on the basis of an extremely simplistic ontology that has to be refined. In the process we plan to add, among others, features representing: delivery options (and prices), trust / reliability and other concepts useful in carrying out e-commerce processes.

(6) Currently, the negotiation strategy module is only a placeholder (agents increase or reduce their offers - depending on the auction - by a fixed amount). A set of somewhat more realistic options will be introduced shortly.

We will be reporting the progress of our research in the subsequent publications.

## REFERENCES

Bartolini, C. et al (2002). *Architecting for Reuse: A Software Framework for Automated Negotiation. Proceedings of the 3rd Int. Workshop on Agent-Oriented Software Engineering, Bologna, Italy, LNCS 2585*, Springer Verlag, pp. 88-100.

Chmiel, K. et al (2004). *Agent Technology in Modelling E-Commerce Processes; Sample Implementation. In: C. Danilowicz (ed.), Multimedia and Network Information Systems*, Volume 2, Wroclaw University of Technology Press, pp. 13-22.

Chmiel, K. et al (2004). *Testing the Efficiency of JADE Agent Platform, Proceedings of the 3rd International Symposium on Parallel and Distributed Computing, Cork, Ireland*, IEEE Computer Society Press, Los Alamitos, CA, pp. 49-57

Cooper, J.W. (2000). *Java Design Patterns. A Tutorial*. Addison-Wesley, 329 pp.

FIPA (1999). *The foundation for intelligent physical agents*. See `http://www.fipa.org`.

Galant, V. et al (2002). *Infrastructure for E-Commerce. Proceedings of the 10th Conference on Knowledge Extraction from Databases*. Wroclaw University of Economics Press, pp. 32-47.

Howard, J., J.Sheth (1969). *The Theory of Buyer Behavior*, Wiley.

JADE. Java Agent Development Framework. See `http://jade.cselt.it`.

Karp, H. A., (2003). Rules of Engagement for Automated Negotiation. Technical Report HPL-2003-152. Intelligent Enterprise Technologies Laboratory, HP Laboratories Palo Alto, USA.

Kowalczyk, R. et al (2002). *Integrating Mobile and Intelligent Agents in Advanced E-commerce: A Survey. Agent Technologies, Infrastructures, Tools, and Applications for E-Services, Proceedings NODe'2002 Agent-Related Workshops, Erfurt, Germany, LNAI 2592*, Springer Verlag, pp. 295-313.

Laudon, K.C., C.G. Traver (2004). *E-Commerce. Business, Technology, Society (2nd ed.)*. Pearson Addison-Wesley, 949 pp.

Paprzycki, M., A ,Abraham (2003). *Agent Systems Today; Methodological Considerations, in: Proceedings of 2003 International Conference on Management of e-Commerce and e-Government*, Jangxi Science and Technology Press, Nanchang, China, pp. 416-421.

Paprzycki, M., A. Abraham, A.Pîrvănescu, C.Bădică (2004). *Implementing Agents Capable of Dynamic Negotiations. In: D.Petcu et. al. (eds.) Proceedings of SYNASC04: Symbolic and Numeric Algorithms for Scientific Computing*. Mirton Press, Timişoara, pp. 369-380.

Parakh, G. (2003). *Agents Capable of Dynamic Negotiations. In: M. Paprzycki (ed.), Electronic Commerce; Research and Development*, ACTEN Press, Wejherowo, Poland, pp. 113-120

Srivastava, V., P.K.J. Mohapattra (2003). PLAMUN: a platform for multi-user negotiation. *Electronic Commerce Research and Applications*, **2(3)**, 339-349.

University of the West,
Faculty of Mathematics
Department of Computer Science
B-dul V. Pârvan, 4
Timişoara, 1900, ROMANIA
emailaddress@info.uvt.ro

Received May 2005