DEVELOPING A JADE-BASED MULTI-AGENT E-COMMERCE ENVIRONMENT

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

Costin Badica
University of Craiova, Software Engineering Department
Bvd.Decebal 107, Craiova, 200440, Romania
badica_costin@software.ucv.ro

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

ABSTRACT

Agent technology is often claimed to be the silver bullet for the task of automating e-commerce business processes, to bring efficiency and profitability to businesses and individual users. Despite the fact that research on software agents can be traced at least as far back as the beginnings of the Internet, it is difficult to find successful large-scale agent-based e-commerce applications to confirm this claim. Our paper addresses this issue by discussing the development of an e-commerce system using a state-of-the-art agent platform – JADE. At this stage, focus of our work is on implementing agents of different types / roles engaged in activities usually encountered in a real e-commerce environment.

KEYWORDS

Multi-agent systems, e-commerce, automated negotiations.

1. INTRODUCTION

E-commerce involves complex processes with many facets, spanning areas that cover business modeling, information technology and social and legal aspects [Laudon, 2004]. A recent survey [Kowalczyk, 2002] pointed out to useful applications of intelligent and mobile agents in support of advanced e-commerce. In summary, agent technology is expected to bring efficiency to businesses (and thus improve its profitability), but also to benefit individual users (e.g. by assuring "price-optimality" of purchases). However, taking into account the high diversity of e-commerce activities involving electronic payments, business document processing (orders, bills, requests for quotes, etc.), advertising, negotiation, user mobility, delivery of goods, etc., it is clear that **a lot more work needs to be done** to achieve the vision of a global distributed e-commerce environment supported by intelligent software agents. This claim 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 200000 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**.

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. First, we have implemented a set of lightweight agents that are capable of adaptive behavior in context of price negotiations (by dynamically loading appropriate software modules [Paprzycki, 2004]). Second, we have implemented a simplistic skeleton for an e-commerce simulation [Chmiel, 2004a]. In the remaining parts of this paper we, first, present the top level description of the system. We follow by a summary of the implementation-specific information as well as an example illustrating its work. We conclude with a summary of future research directions.

2. 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 targeted to solve a specific business problem that uses a limited number of application-specific agents. In other words, we are facing the multi-agent challenge from the very beginning.

2.1. System Architecture

Our e-commerce model extends and builds on the e-commerce structures presented in [Galant, 2002], [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 we assume that the price is the **only** factor determining the purchase and, furthermore, only shops are allowed to advertise their products. Obviously, these are serious restrictions 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. Our environment supports dynamic agent creation and destruction and agent migration to engage in negotiations. The top level conceptual architecture of the system illustrating proposed types of agents and their interactions in a particular configuration is shown in Figure 1 (we have omitted *Personal* agents assuming that their role is obvious, as they were responsible for creating *Shop* and *Client* agents and thus initiating operation of the system). Let us now describe each agent appearing in that figure (as well as *Personal* agents) and their respective functionalities.

Personal agents facilitate communication between the system and the "real-world" users (*shoppers* and *merchants*). A shopper will employ its *Personal* agent to communicate to the system his sought after product(s) and possibly buying policies (currently only the policy of price minimization is available). The *Personal* agent will then create a *Client* agent to act within the marketplace on his behalf. A merchant will utilize her *Personal* agent to create a *Shop* agent, responsible for advertising and selling her products within the marketplace. After being created both *Shop* and *Client* agents register with the *CIC* agent to be able to operate within the marketplace. Returning 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 (in the future we may need to address this potential bottleneck [Chmiel 2004b]). It is responsible for storing, managing and providing information about all "participants." 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.

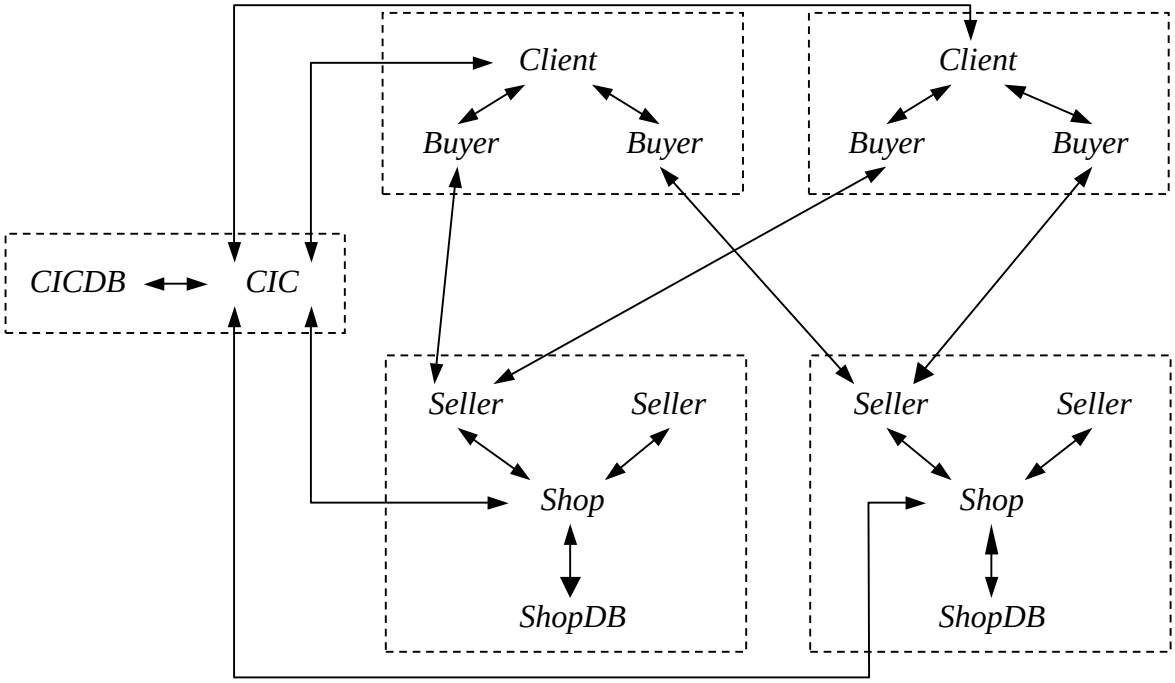

Figure 1. The conceptual architecture of our e-commerce environment (two-client; two-store version).

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* agent hereafter). One *Buyer* agent is created for each store, within the marketplace, selling sought goods.

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.

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 the future we may need to address this possible bottleneck [Chmiel 2004b]). 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* (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.

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 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.

2.2. 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 and a reserve price. A *Shop* agent obtains a list of pairs (product, reserve price) and the negotiation protocol that is to be used for interactions with incoming *Buyer* agents. The reserve price of a *Client* agent is the maximum price it agrees to pay for the product. The reserve price of a *Shop* agent is the minimum price at which it is to agree to sell a specified product. In the future, *Client* and *Shop* agents will have access to a collection of strategies to be used depending on the context of unfolding negotiations [Parakh, 2003].

The *Shop* agent creates *Seller* agents, one *Seller* agent for each product sold. *Seller* agents await incoming *Buyer* agents interested in buying their products and upon their arrival engage in negotiations with them. Let us now describe what happens in the marketplace after a customer has made a purchase request, until a request is completed.

(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 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]) 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. Note that there are various strategies that could be employed by a *Client* agent in order to decide where to buy from, and there is an associated risk also. For example, it may decide for the best offer, the safest offer or the most trusted offer, etc. Currently, we are using a simple strategy of choosing the best negotiated price.

3. IMPLEMENTATION AND EXPERIMENT

3.1. System Implementation

The current implementation of the proposed environment has been made within the JADE 3.1 agent platform ([JADE]). 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 matches well with our requirements. Negotiations between *Seller* and *Buyer* agents take place in 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 the *Client* agents are using JADE mobile agent technology to migrate to the *Shop* agent containers to engage in negotiations.

Figure 2 presents mapping of our conceptual architecture (Figure 1) onto JADE. In particular this diagram shows two machines running *Personal*, *Shop*, *Client*, *Buyer* and *Seller* agents, highlighting also the JADE containers involved (*Main* and *Container-1*), the sought product (p_1) and the advertised products (p_1, p_2, p_3). Single-arrow continuous lines denote agent creation. For example *Personal* agent P_c creates *Client* agents C_1 and C_2. Single-arrow dotted lines denote agent migration. For example *Buyer* agent B_{11} migrates from the *Main* container to the *Container-1*. Double-arrow continuous lines denote negotiations. For example *Seller* agent S_{11} negotiates with *Buyer* agents B_{11} and B_{21}.

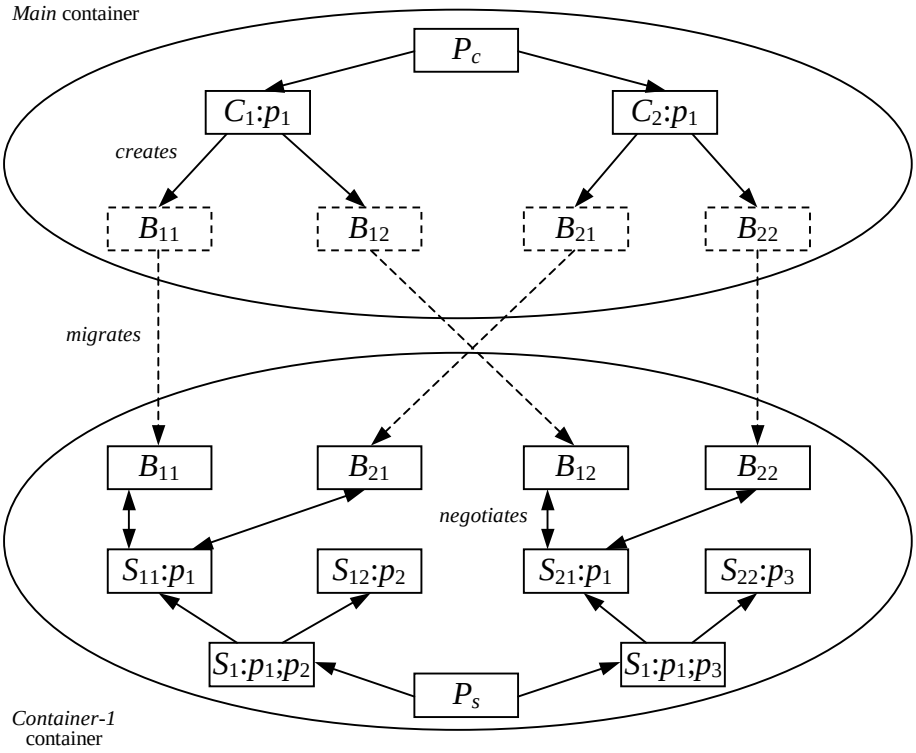

Figure 2. Mapping the conceptual architecture of the system into JADE

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

- *Agent classes*, 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 *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 except the *PersonalAgent* class that extends the *GuiAgent* class (provided by JADE).

- *Agent activity classes*, also called behaviors, 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*, 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] (for similar ideas see also [Jarraya, 2004]).

- *Ontology classes*, 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*, including the 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 participate 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.

3.2. Running the System

Here, we introduce a simple experiment to illustrate main features of our implementation. In order to run the experiment we set-up JADE platform on two computers: *comp1* and *comp2*. On computer *comp1* the *Main* container is initialized. On computer *comp2* a second container *Container-1* that is linked with the *Main* container on *comp1* was started. On both computers we have set-up MySQL database. Both the *CIC* and the *CICDB* agents are created by default within the *Main* container on *comp1*, while the *ShopDB* and the *ShopDB* agent are created in *Container-1* on *comp2*.

For the experiment we have chosen a simple scenario with two merchants and two customers. In order to enable price competition, we assume that both customers are seeking the same product – *good1*. The first merchant advertises two products: *good1* and *good2*, while the second merchant advertises two products: *good1* and *good3*. Thus merchants compete in selling *good1*. The diagram shown in Figure 2 illustrates this scenario.

Customers and merchants used *Personal* agents to create *Client* and *Shop* agents. In this experiment merchants used *Personal* agents running in *Container-1* container to create two *Shop* agents (Figure 3, upper left panel) and customers used *Personal* agents running in *Main* container to create two *Client* agents (Figure 3, upper right panel).

The process of starting *Shop* agents involved their registration with the *CIC* agent. Furthermore, for each product offered a *Seller* agent was created (in *Container-1*). Therefore, four *Seller* agents were created.

Similarly, the process of starting *Client* agents involves 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, in our experiment, four *Buyer* agents were created (two *Client* agents send two *Buyer* agents each to two e-stores).

At this stage, *Buyer* agents move to *Container-1* and register with appropriate *Shop* agents. As a result of message exchanges (Figure 3, bottom panel) negotiation protocol is identified and negotiation modules loaded by *Buyer* agents. In the next step, *Buyer* agents subscribe with *Seller* agents that sell product *good1*. *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 two auctions for *Seller* agents selling product *good1*.

When negotiations end, *Shop* agents inform *Client* agents about the result of negotiations. The *Client* agent collects all results and decides where from to buy the product *good1*, informing the *Shop* agent accordingly.

Figure 3. Screen captures showing our system in action

Figure 3 (bottom panel) 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.

4. CONCLUDING REMARKS

In this paper we have presented basic features of an e-commerce modeling agent system that we are in the process of developing. At 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, 2002] 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, *Cilents*, *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.

REFERENCES

Cooper, J. W., 2000. *Java Design Patterns. A Tutorial*. Addison-Wesley, USA

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

Srivastava, V., Mohapattra, P.K.J, 2003. PLAMUN: a platform for multi-user negotiation. *Electronic Commerce Research and Applications*, Vol.2, No.3, pp. 339-349

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

Paprzycki, M. et al., 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, Timisoara, Romania, pp. 369-380

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

Chmiel, K. et al, 2004b. 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

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

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

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

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

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, pp. 295-313

Jarraya, T., 2004. Using Generic Interaction Protocols to Develop Multi-Agent Systems. *Proceedings of the International Conference on Advances in Intelligent Systems: Theory and Applications*, Kirchberg, Luxemburg

Paprzycki, M., Abraham, A., 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

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