Experimenting With a Multi-Agent E-Commerce Environment

Costin Bădică[1], Maria Ganzha[2], Marcin Paprzycki[3], and Amalia Pîrvănescu[4]

[1] University of Craiova, Software Engineering Department
Bvd.Decebal 107, Craiova, 200440, Romania
badica_costin@software.ucv.ro

[2] Gizycko Private Higher Educational Institute, Department of Informatics
Gizycko, Poland
ganzha@pwsz.net

[3] Oklahoma State University, Computer Science Department
Tulsa, OK, 74106, USA and
Computer Science, SWPS, 03-815 Warsaw, Poland
marcin@cs.okstate.edu

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

Abstract. Agent technology is often claimed to be the most natural approach for automating e-commerce business processes. Despite these claims, up till now, the most successful e-commerce systems are still based on humans to make the most important decisions in various stages of an e-commerce transaction. Consequently, it is difficult to find successful actually implemented and working large-scale agent-based e-commerce applications to confirm agents superiority. Here, we discuss an abstract e-commerce environment that allows agents of different types to interact with each other and operate with an overarching goal of supporting an e-commerce transaction. A prototype system that implements this vision using JADE agent platform is also described. Finally, we report on experiments with the implemented system skeleton.

1 Introduction

E-commerce involves complex processes with many facets, spanning areas that cover business modeling, information technology and social and legal aspects ([9]). A recent survey ([8]) pointed out to useful applications of intelligent and mobile agents in support of advanced e-commerce. The main message permeating his (and other) work is that agent technology is expected to bring efficiency to businesses and thus improve its profitability (e.g. by improving the rate of successful transactions from the total number of attempted transactions, or by decreasing the total time required to complete a transaction), but also to benefit individual users (e.g. by assuring ”price-optimality” of purchases or by increasing customer satisfaction). 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, security 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 (and used in day-to-day operation) large-scale implementation of an e-commerce agent system. While a number of possible reasons for this situation have been suggested (see, for instance, [10]), one of them has been recently dispelled. It was shown that modern agent environments (e.g. JADE, [7]) can easily scale to 1500 agents and 300000 messages ([3]). 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*. This is the general direction of agent system development that will be addressed in this paper. If mobile and intelligent software agents are to become an important part of the e-commerce infrastructure, we have to start implementing such systems that involve large number of agents interacting in a way that is to model realistic scenarios arising in an e-marketplace. This process has to have at least two goals in mind: (1) to focus on the technical aspects of the system, such as agent functionalities, their interactions and communication, agent mobility etc., and (2) to focus on modeling the economical processes occurring in an e-marketplace, such as: effects of pricing strategies, of negotiation protocols and strategies, flow of commodities etc. The first goal attempts to address the problem of lack of large-scale agent systems implemented using agent environments (we are aware of large bio-inspired agent simulations written in C, but this is not what we are interested in). Without being able to show that it is actually possible to implement such systems, using tools that are apparently designed with this goal in mind, agent research will never be able to reach beyond academia. The second goal points to a possible application of the system. While we do not try to convince anyone that as system like ours will be immediately usable in real-life e-commerce, we can point to an interesting way to utilize our system. This possible application is e-commerce modeling. Due to the agent flexibility it will be relatively easy to experiment with the above described as well as other factors appearing in various e-commerce scenarios. While both of these developmental paths are very closely related to each other in this paper we are more concerned with the former.

In this broad context, our goal is to create a system with a multitude of agents that play variety of roles and interact with each-other in an abstract e-commerce environment. Currently, we follow our earlier work, where first, we have implemented a set of lightweight agents capable of adaptive behavior in context of price negotiations (by dynamically loading appropriate software modules; see [11] and work referenced there). Second, we have implemented a simplistic skeleton for an e-commerce simulation ([2]). Third, we have combined these two developmental threads into a unified e-commerce environment [12]. One of the important limitations of our work reported thus far was the fact that we have experimented only with a very limited number artifacts populating our system (products, negotiation mechanisms and strategies, agents of various types, computers). The aim of this paper is to report on the results of our experiments when the size of the system has been increased considerably. 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 in a larger-scale setting.

2 System Description

In our work we aim at implementing a multi-agent e-commerce system that in a long run will help carrying out experiments with real-world e-commerce scenarios. In this context, note the exploratory nature of our work: the system is based on an abstract e-commerce environment describing an artificial world in which e-commerce agents perform variety of functions typically involved in e-commerce, rather than on a solution to a specific business problem in terms of a limited number of application-specific agents.

Our e-commerce model extends and builds on the e-commerce structures presented in [2] and [11]. 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 negotiation mechanisms (including auctions), have an option to choose the buyer. At this stage the system is under development and has a number of limitations. (1) Only four negotiation protocols have been implemented: FIPA English auction, FIPA Dutch auction, iterative bargaining and fixed price (also known as take-it-or-leave-it). Note that the first two are one-to-many negotiations while the last two are one-to-one negotiations (see [1] for a discussion on how various negotiation mechanisms can be parameterized). (2) The two strategy modules are trivial and are there only to show that such modules can be downloaded upon request. (3) We have only shops that are allowed to advertise commodities (clients cannot advertise products 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 have implemented only single-item negotiations. In the case of multi-item negotiations there exist a much large number of factors influencing purchase decision. (6) Our system can be (but is not) made adaptable through data mining (e.g. history of buyer-seller interactions can lead to negotiation strategy adjustment). We plan to address these serious restrictions in the near future.

Shops and clients can be 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 our system (especially in the context of e-commerce modeling - goal (2) above). Furthermore, a *Personal Agent* is considered to be an envoy 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 ”only” to create *Client* / *Shop* agents that will be a part of the e-commerce system; and therefore the *Personal Agent* is not further discussed. Note that it is also possible to create *Client* and *Shop* agents via a command-line line interface. This facility extremely is useful for preparing experiments via scripting programs.

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. Let us now describe each agent appearing in that figure and their respective functionalities.

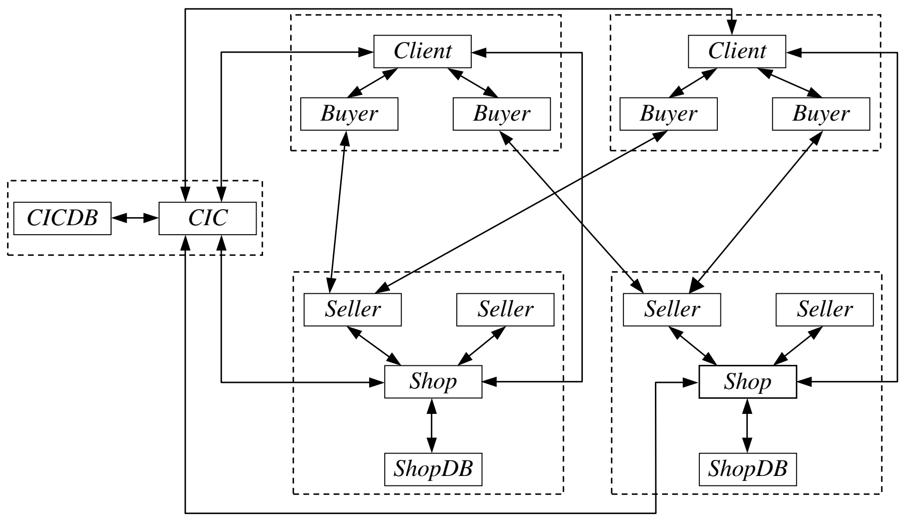


Fig. 1. The abstract e-commerce environment (two-client; two-shop version).

A *Client agent* (*CA*) is created by the *Personal agent* to act within the marketplace on behalf of a user that attempts to buy ”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 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 [3]). 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.

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.

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 [3]. 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.

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 an 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. Note that once the complete system is created, changing this policy will require only a limited amount of work. Furthermore, it will be possible to add different scenarios of completing negotiations to the system and build a mega-system, where all of these strategies will exist together. Ability to achieve this goal by simply adding new agents with new behaviors illustrates the power of agent-based system design.

3 Implementation and Experiments

3.1 System Implementation

The current implementation of the proposed environment has been made within the JADE 3.3 agent platform ([7]). 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 (and, as illustrated below, it is also possible to run JADE in a mixed environment). Furthermore, as reported above, in [3] we have observed its very good scalability.

JADE provides a flexible and configurable architecture that 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 *Client* agents use JADE mobile agent technology to migrate to the *Shop* agent containers to engage in negotiations. In this context, a container simulates a marketplace where various *Seller* and *Buyer* agents meet and negotiate. Moreover, all these containers linked via the agent platform simulate a bazaar filled with marketplaces filled with trading agents.

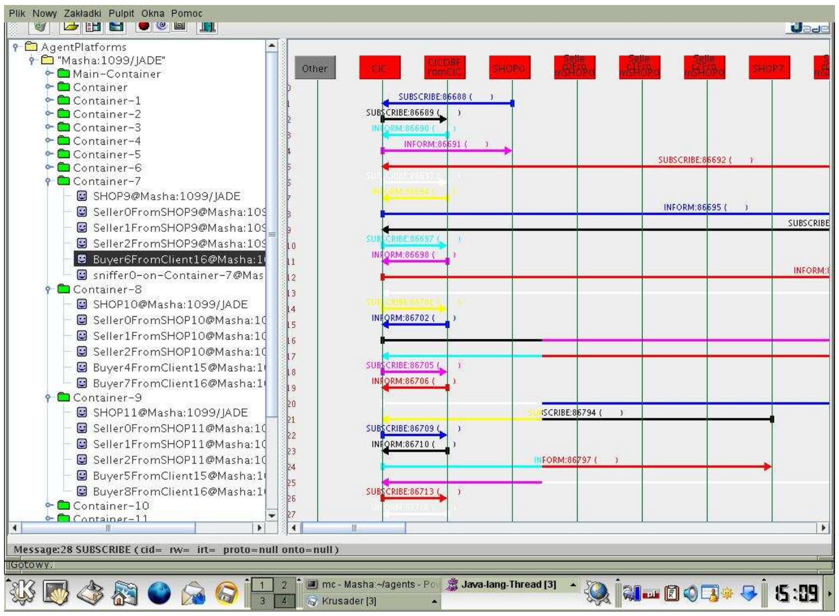
The current implementation is based on several Java classes organized into several categories. Each category is implemented as a separate Java package.

– *Agent classes*. Classes of this package are used for describing various agent types used in the system. Each agent class incorporates a subset of agent activity classes, also called behaviors. Behaviors are used as an abstraction that represents an atomic activity performed by an agent.
– *Database classes*. Classes of this package are used for describing agents that are responsible for management of database connections.
– *Negotiation classes*. Classes of this package implement a simple framework for describing various negotiation protocols. This framework uses the Initiator and Participant roles, as defined by the FIPA Contract Net Interaction protocol ([5]).
– *Reasoning classes*. These classes used for the implementation of the various reasoning models employed by the negotiation agents; see [11] for more details concerning model of negotiation agents. Our implementation supports agents that dynamically load their negotiation protocols and reasoning modules. The implementation combines the Factory design pattern ([4]) and dynamical loading of Java classes ([11]).
– *Ontology classes*. These classes are necessary for implementing agent communication semantics, using concepts and relations. 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*. This package contains various helper classes.
In our system, agent communication is implemented using FIPA ACL messages [5]. 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 Experiments

The system can be run in a simple setting for demonstration purposes by manually creating *Shop* and *Client* agents via the GUI, or directly from command-line when a large number of agents, containers, products etc. is to be created [6].

For the purpose of this paper we have utilized experiments involving multiple agents residing on multiple computers. First, *Client* agents resided on a single computer and *Buyers* migrated to *Shop* agents residing on the remaining 19 machines. Second, *Client* agents resided on 4 computers, while the remaining 16 machines contained *Shop* agents. Furthermore, to illustrate heterogeneity of the environment in which our system can run, in both experimental settings the Main container of the agent platform resided on a computer running Linux, while the remaining 20 computers run Windows. In addition JADEs *Sniffer* agent also was executed, on the Linux PC,. This agent is provided by JADE and its role is to report on communications between agents in the system. Figure 2 presents agent communication captured with help of this agent (note Linux environment).



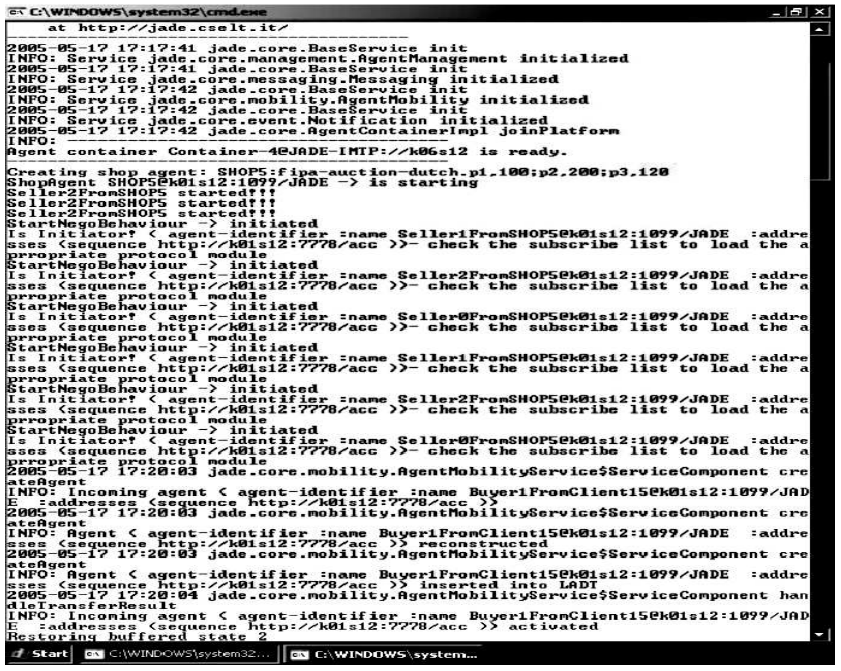
```
AgentPlatforms
  "Masha:1099/JADE"
    Main-Container
    Container
    Container-1
    Container-2
    Container-3
    Container-4
    Container-5
    Container-6
    Container-7
      SHOP9@Masha:1099/JADE
      Seller0FromSHOP9@Masha:109
      Seller1FromSHOP9@Masha:109
      Seller2FromSHOP9@Masha:109
      Buyer6FromClient16@Masha:1
      sniffer0-on-Container-7@Mas
    Container-8
      SHOP10@Masha:1099/JADE
      Seller0FromSHOP10@Masha:10
      Seller1FromSHOP10@Masha:10
      Seller2FromSHOP10@Masha:10
      Buyer4FromClient15@Masha:1
      Buyer7FromClient16@Masha:1
    Container-9
      SHOP11@Masha:1099/JADE
      Seller0FromSHOP11@Masha:10
      Seller1FromSHOP11@Masha:10
      Seller2FromSHOP11@Masha:10
      Buyer5FromClient15@Masha:1
      Buyer8FromClient16@Masha:1
    Container-10
    Container-11
```

```
SUBSCRIBE:86688 (     )
SUBSCRIBE:86689 (     )
INFORM:86690 (     )
INFORM:86691 (     )
SUBSCRIBE:86692 (     )
INFORM:86695 (     )
SUBSCRIBE
SUBSCRIBE:86697 (     )
INFORM:86698 (     )
INFORM:
INFORM:86702 (     )
SUBSCRIBE:86705 (     )
INFORM:86706 (     )
SUBSCRIBE:86794 (     )
SUBSCRIBE:86709 (     )
INFORM:86710 (     )
INFORM:86797 (     )
SUBSCRIBE:86713 (     )
```

```
Message:28 SUBSCRIBE ( cid=  rw=  irt=  proto=null onto=null )
```

Fig. 2. The beginning of work of the system — registration with CIC and CICDB agents

In the experiment shown in Fig 2 every *Shop* had three different products. Thus, at the beginning of an experimental run every *Shop* registered with the *CIC agent*, then created 3 *Sellers* (one *Seller* for each product). *Seller* agents also registered with the *CIC agent* and then waited for the incoming *Buyer(s)*. Communication involved in these operations can be seen in Fig 2. There exist two events which are necessary for to start negotiations: appearance of at least one *Buyer* and an interrupt caused by the timer (see Figure 3).



C:\WINDOWS\system32\cmd.exe

```
     at http://jade.cselt.it/
----------------------------------------------------------
2005-05-17 17:17:41 jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
2005-05-17 17:17:41 jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
2005-05-17 17:17:42 jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
2005-05-17 17:17:42 jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
2005-05-17 17:17:42 jade.core.AgentContainerImpl joinPlatform
INFO: --------------------------------------
Agent container Container-4@JADE-IMTP://k06s12 is ready.
----------------------------------------------

Creating shop agent: SHOP5:fipa-auction-dutch.p1,100;p2,200;p3,120
ShopAgent SHOP5@k01s12:1099/JADE -> is starting
Seller2FromSHOP5 started!!!
Seller2FromSHOP5 started!!!
Seller2FromSHOP5 started!!!
StartNegoBehaviour -> initiated
Is Initiator? ( agent-identifier :name Seller1FromSHOP5@k01s12:1099/JADE  :addre
sses (sequence http://k01s12:7778/acc ))- check the subscribe list to load the a
prropriate protocol module
StartNegoBehaviour -> initiated
Is Initiator? ( agent-identifier :name Seller2FromSHOP5@k01s12:1099/JADE  :addre
sses (sequence http://k01s12:7778/acc ))- check the subscribe list to load the a
prropriate protocol module
StartNegoBehaviour -> initiated
Is Initiator? ( agent-identifier :name Seller0FromSHOP5@k01s12:1099/JADE  :addre
sses (sequence http://k01s12:7778/acc ))- check the subscribe list to load the a
prropriate protocol module
StartNegoBehaviour -> initiated
Is Initiator? ( agent-identifier :name Seller1FromSHOP5@k01s12:1099/JADE  :addre
sses (sequence http://k01s12:7778/acc ))- check the subscribe list to load the a
prropriate protocol module
StartNegoBehaviour -> initiated
Is Initiator? ( agent-identifier :name Seller2FromSHOP5@k01s12:1099/JADE  :addre
sses (sequence http://k01s12:7778/acc ))- check the subscribe list to load the a
prropriate protocol module
StartNegoBehaviour -> initiated
Is Initiator? ( agent-identifier :name Seller0FromSHOP5@k01s12:1099/JADE  :addre
sses (sequence http://k01s12:7778/acc ))- check the subscribe list to load the a
prropriate protocol module
2005-05-17 17:20:03 jade.core.mobility.AgentMobilityService$ServiceComponent cre
ateAgent
INFO: Incoming agent ( agent-identifier :name Buyer1FromClient15@k01s12:1099/JAD
E  :addresses (sequence http://k01s12:7778/acc ))
2005-05-17 17:20:03 jade.core.mobility.AgentMobilityService$ServiceComponent cre
ateAgent
INFO: Agent ( agent-identifier :name Buyer1FromClient15@k01s12:1099/JADE  :addre
sses (sequence http://k01s12:7778/acc )) reconstructed
2005-05-17 17:20:03 jade.core.mobility.AgentMobilityService$ServiceComponent cre
ateAgent
INFO: Agent ( agent-identifier :name Buyer1FromClient15@k01s12:1099/JADE  :addre
sses (sequence http://k01s12:7778/acc )) inserted into LADT
2005-05-17 17:20:04 jade.core.mobility.AgentMobilityService$ServiceComponent han
dleTransferResult
INFO: Incoming agent ( agent-identifier :name Buyer1FromClient15@k01s12:1099/JAD
E  :addresses (sequence http://k01s12:7778/acc )) activated
Restoring buffered state 2
```

Fig. 3. The beginning of work of the system — DOS window

After creation, *Client* registered with the *CIC agent*. Upon user request, it obtained list of *Shops*, where product(s) of interest were sold and created *Buyer* agents and sends them to the selected *Shops*. When *Buyer* arrived at the marketplace it asked about current negotiation protocol, communicated with its *Client* and obtained a corresponding strategy module and waited for start of negotiations. After finishing negotiations, *Seller* informed *Shop* agent about their results and *Shop* agent notified appropriate *Client* about successful result of negotiations (see also Fig 2).

In the experiment represented in Fig 2 and Fig 3 we used three products, which *Client* could buy. Thus, we had a total of more than 200 agents populating the system. It should be pointed out that the most time-consuming operation is system initialization (creation of containers). However, since containers are created once, they have only minimal impact on the operations of the system.

We have run multiple experiments, changing the number of (a) containers, (b) computers, (c) *Clients*, (d) *Shops*, (e) negotiation protocols, (f) products (g) mixture of Linux and Windows environments, etc. In each case experiments run smoothly and supported our general claim that the proposed system, when further developed can: (1) can be scaled to a truly large size, and (2) be used for e-commerce modeling.

4 Concluding Remarks

In this paper we have introduced an agent-based e-commerce system that has actually been implemented and show to fulfill the basic promises of agent systems. The most important of them were: (1) system scalability, (2) flexibility, and (3) heterogeneity. Obviously, the proposed system has a number of shortcomings that we are aware off, and we will work vigorously to remove them and develop and implement a truly comprehensive system. We will report on our progress in subsequent reports.

References

1. Bartolini, C., Preist, C., Jennings, N.R.: A Software Framework for Automated Negotiation. In: *Proceedings of SELMAS'2004*, LNCS 3390, Springer Verlag (2005) 213–235.
2. Chmiel, K. et al.: 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, (2004) 13–22.
3. Chmiel, K. et al.: Testing the Efficiency of JADE Agent Platform. In: *Proceedings of the 3rd International Symposium on Parallel and Distributed Computing, Cork, Ireland*, IEEE Computer Society Press, Los Alamitos, CA, (2004) 49–57.
4. Cooper, J.W.: *Java Design Patterns. A Tutorial*. Addison-Wesley, (2000).
5. FIPA: *The foundation for intelligent physical agents*. See `http://www.fipa.org`.
6. 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* (to appear), (2005).
7. JADE: Java Agent Development Framework. See `http://jade.cselt.it`.
8. Kowalczyk, R. et al.: 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, (2002) 295–313.
9. Laudon, K.C., Traver, C.G.: *E-Commerce. Business, Technology, Society (2nd ed.)*. Pearson Addison-Wesley, (2004).
10. Paprzycki, M., Abraham, A.: 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, (2003) 416–421.
11. Paprzycki, M., Abraham, A.. Pîrvănescu, A., Bădică, C.: Implementing Agents Capable of Dynamic Negotiations. In: Petcu, P. and Negru, V. (eds.): *Proceedings of SYNASC'04: Symbolic and Numeric Algorithms for Scientific Computing*, Timişoara, Romania, Mirton Press, Timişoara, Romania (2004) 369–380.
12. Pîrvănescu, A., Bădică, C., Paprzycki, M.: Developing a JADE-based Multi-Agent E-Commerce Environment. In: Guimarães, N. and Isaías, P. (eds.): *Proceedings IADIS AC'05: International Conference on Applied Computing*, Algarve, Portugal, IADIS Press, Lisbon, Portugal (2005) 425–432.