Computing and Informatics, Vol. 22, 2013, 1001–1032, V 2013-Jan-21

DISTRIBUTED AGENT-BASED ONLINE AUCTION SYSTEM

Costin Bădică, Sorin Ilie, Alex Muscar

University of Craiova
Software Engineering Department
Bvd. Decebal, 107, Craiova, Romania
e-mail: {cbadica,silie,amuscar}@software.ucv.ro

Amelia Bădică

University of Craiova
Statistics and Business Information Systems Department
Str. A.I.Cuza, 13, Craiova, Romania
e-mail: ameliabd@yahoo.com

Liviu Sandu, Raluca Sbora

University of Craiova
Software Engineering Department, Bvd. Decebal, 107
Craiova, Romania
e-mail: ssanduliviu@yahoo.com, ralus08@yahoo.com

Maria Ganzha, Marcin Paprzycki

Polish Academy of Sciences
Systems Research Institute, ul. Newelska 6, 01-447
Warszawa, Poland
e-mail: Maria.Ganzha@ibspan.waw.pl, Marcin.Paprzyck@ibspan.waw.pl

Abstract. This paper concerns the design and development of a distributed agent-based online system for English auctions. The proposed system is composed of two parts: an *Agent-based Auction Server* and a *Web-based Graphical User Interface.*

The first part of our work brought about the advantages introduced by the multi-agent systems technology to the high-level of abstraction, modularity and performance of the server architecture and its implementation. On the server side, bids submitted by auction participants are handled by a hierarchical organization of agents that can be efficiently distributed on a computer network. This approach avoids the bottlenecks of bid processing that might occur during periods of heavy bidding, like for example snipping. We present experimental results that show a significant improvement of the server throughput compared with the architecture where a single auction manager agent is used for coordinating the participants for each active auction that is registered with the server.

The second part of our work involved analysis of external functionalities, implementation and usability of a prototype online auction system that incorporates the *Agent-based Auction Server*. Our solution is outlined in terms of information flow management and its relation to the functionalities of the system. The main outcome of this part of the work is a clean specification of the information exchanges between the agent and non-agent software components of the system. Special attention is also given to the interoperability, understood here as successful integration of the different data communication protocols and software technologies that we employed for the implementation of the system.

Keywords: distributed system, multi-agent middleware, online auction

1 INTRODUCTION

The vision of e-commerce automation proposes the development of global e-commerce environments populated by software agents, thus enabling the dynamic development of trading relationships between business partners. In particular, increasing the level of automation of negotiations is needed, to allow the engagement of stakeholders, either individuals or business organizations, into nontrivial dynamic business relationships.

As it was argued in [8], auctions provide a general solution to the problem of discrete resource allocation among selfish agents in a multi-agent system. There are many types of auctions including single-good, multi-unit, combinatorial auctions, and double auctions. Auctions represent a special class of negotiations with many applications in conducting e-business transactions [32]. In particular auctions are useful for trading in the following specific areas: spectrum licenses, electricity markets, emission rights, airports takeoff and landing slots, exploitation rights of natural resources (e.g. oil-drilling), selling of collectibles, antiques, luxury and second-hand products, government procurement contracts, foreign exchange, a.o. [7].

As online auctions spread with the advent of the web, many types of online applications for auctions were proposed including: auction directories, auction tops, meta-auctions, and auction servers [2]. Recently, the research focus was set on the development of more process-generic, flexible and reusable auction solutions, with an increased potential for applicability both to the B2C and B2B sectors. In this context, application of agent-based systems was proposed as a new approach that takes the idea of an auction service from the human-driven web to the software agents' world [13].

In this paper we consider specific example of English auctions. In this case, the seller (or the auctioneer that represents the seller) announces an initial price for the good (assuming an e-commerce application setting) and auction participants bid increasing amounts of money during a predefined time frame, usually by some minimum increment set by the seller. At the end of the bidding process, the agent with the highest bid is declared the winner of the English auction. Note that, we consider here the deadline-driven model of the English auction, rather than the time of inactivity model.

Our current work in this area is focused on the analysis, design and implementation of an open, flexible infrastructure for agent-based automated negotiations. The two objectives of this paper are:

(i) To underline the advantages of using multi-agent systems and state-of-the-art agent frameworks and middleware [25] when developing a realistic auction server. Here, the focus is on employing clean software engineering principles (abstraction and modularity), as well as on evaluating and improving the performance and scalability of the implementation. Initial discussion of software engineering principles that underline this work, as well as preliminary performance assessment were presented in [2]. The scalability and performance aspects were further expanded by realization of a cluster-based implementation that was outlined and initially evaluated in [11]. Here, we combine and further extend the results presented in these two conference papers.

(ii) To develop a tool that can be used for online auctions in B2C systems. This goal is addressed by focusing on the details of incorporating the *Agent-based Auction Server* into a Web-based application for online auctions.

The proposed agent-based solution for the auction server combines the best features of: (i) generic software framework for automated negotiations [3]; (ii) market architecture for auction development [6]; (iii) rule-based declarative representation of auction mechanisms [1, 4]; (iv) special computing nodes available in active networks and realized by means of proxy agents [9]; (v) agent-based service-oriented architecture [2]. Consequently, it provides certain features including: openness, generality, and scalability. For example, with this approach an auction is seen as a separate service, rather than being entirely incorporated into an e-shop infrastructure. This can be seen as a gain in openness, as the service is now open for rental and configuration by the e-shop that would like to sell its products through an auction, for example for clearing its shelves during the "sales" time. Generality comes form the fact that the e-shop can now choose the most appropriate auction server depending on factors like performance, reliability or trustworthiness of the service. Finally, scalability comes from our new approach that combines the use of *Proxy* agents with two level balanced tree structures for handling participants' bids.

The second part of our work was focused on the analysis of the external functionalities, implementation and usability of a prototype online auction system that incorporates the *Agent-based Auction Server.* We present the details of our solution in terms of the information flow management and its relation to the functionalities of a system for online auctions.

In summary, our work brings several contributions to the research on agent-based e-commerce.

When an auction (in particular an English auction) is modeled as a multi-agent system, the agents' interaction through message exchanges is required for the bidding and price update activities. In our previous approach [2], the central agent for auction management was responsible with handling all the communication with, and between, the auction participants. So, if a large number of participants joined the auction, this agent became highly stressed. Second, although the approach introduced in [2] was described as distributed, understood as possibility for agents to be arbitrarily located on networked computers, it was still biased towards centralization, because the central agent was a system bottleneck, and thus hindered the scalability.

In this paper we propose a solution aiming to improve the architecture of our *Agent-based Auction Server* by relaxing the central agent from part of the stress caused by its heavy load of message handling. Based on the idea that was initially proposed by [9], for improving the performance of on-line auction systems using special computing nodes available in active networks, we enhanced our system with the introduction of third party agents called proxies. Each *Proxy* agent will handle a part of the communication with the auction participants.

So, while in our prior approach the *Auction Manager* would receive all the bids from the *Participants*, with this new approach the *Participants* are split into disjoint groups, and each group is managed by a single *Proxy* agent. The *Participants* communicate heavily with their proxies, while proxies pass on to the *Auction Manager* agent only the relevant bids, while the other bids are filtered out and processed locally, thus reducing the amount of messages handled by the *Auction Manager.*

Note that the communication between two agents is faster when the agents are located on the same machine, rather than when they are located on separate machines. When our auction server is distributed on several computers, some agents will have to exchange messages over the network, thus increasing the communication time, as well as the overall server response time. So, with our solution we also aimed to improve the communication time between agents, when the server is distributed on several computers, by keeping, whenever possible, *Proxy* agents on the same machine with their "participant agents".

Finally, our work contributes also to research concerning usability of agent-based e-commerce solutions. While very attractive, the complete automation of e-commerce processes is probably impossible to achieve, and therefore the human user involvement, through an appropriate online system, will be always necessary. Therefore, we found it important to experiment with the integration of the proposed *Agent-based Auction Server* into a usable online auction system that allows the direct human user involvement in auctions, via a Web-based GUIs.

Furthermore, we provide a clean specification of the information exchanges between the agent and non-agent software components of the system, which is particularly interesting from the software engineering point of view. In this context, special attention is also given to the heterogeneity of the different data communication protocols (for example: HTTP, FIPA, parameter passing via method invocation, a.o.) and software technologies (Web technologies vs software agent technologies) that we utilized for implementing the system.

The paper is structured as follows. We start in Section 2 with a brief overview of related works in the field. In Section 3 we present the architecture of our *Agent-based Auction Server* covering agent types, interaction protocols and mechanisms for efficient bid processing. Next (in Section 4), we propose the design of a Web system for online auctions that incorporates the auction server. Here, the discussion is focused on three aspects: (i) system architecture, (ii) design details of the Web layer as well as of the interfacing of the agent and non-agent software, and (iii) interaction protocols. In Section 5 we present results of experiments carried out with the auction server, including recorded values of latency and throughput parameters. Here we also discuss the usability of the online auction system. In the last Section 6, we present our conclusions and we point to future works.

2 RELATED WORKS

The interest in development of online software systems for online negotiations, with a special focus on online auctions, increased significantly during the last 15 years. Traditionally, auctions were utilized for trading support in economic markets in offline as well as in online environments. Recently auctions started to be applied in market environments for trading resources for utility computing, including grids and clouds [14].

One of the first and most influential works in the area of auction servers for online applications is the Michigan Internet AuctionBot introduced in [22]. This is a versatile and robust server for online auctions supporting both agent-oriented and human-oriented auction execution. The Michigan Internet AuctionBot introduced the principles of software design for supporting flexible auction mechanisms, including: separation of the user interface from the core auction engine, the capability of running concurrently multiple auctions, as well as the abstraction of the auction process. Most of these principles are currently employed by state-of-the-art auction servers including our own.

In [17], the authors proposed an Internet-based negotiation server for e-commerce applications. Although this work does not explicitly address auction mechanisms (rather, the focus is on bargaining) and the use of software agent technologies, it is interesting for our approach for at least the following reasons: (i) the system is conceptualized as a replicable service that can be multiply instantiated by complementing standard Web server software, i.e. quite similarly to our proposal; (ii) the system incorporates methods of event-based rule processing and constraint satisfaction for checking negotiation proposals and implementation of negotiation strategies which, although they are not the focus of this paper, they were also employed in our previous work ([1, 2]).

The authors of [16] propose an agent-based modeling of the New York Stock Exchange specialist system. Although this work clearly differentiates from our own work, as the focus is not on the development of an online system incorporating an auction server, but rather on the agent-based modeling of the complex interactions occurring in the New York Stock Exchange specialist system, there are also similarities. First, their modeling addresses a non-trivial class of auctions – continuous double auctions and, second, the modeling could be further expanded to cover the development of an e-service system as part of the New York Stock Exchange.

The e-Game tool that supports the design and implementation of electronic market simulation games inspired by the real life problems, was proposed by the authors of paper [15]. These simulations can also incorporate various types of auctions, and they were used for teaching purposes. The e-Game tool provides both Web and agent interfaces, similarly to our system. Nevertheless, differently from our work, the aspects related to software engineering principles, performance and scalability were not addressed.

In [19], the authors present the principles of constructing online auction systems that were employed for building the Research Auction Server for performing both simulated and real auctions. However, many details are lacking from their description, especially those related to the interaction protocols. Therefore we could not compare our approach with [19] because of the missing information. Moreover, although the "agent" metaphor is used we noticed that the development of the Research Auction Server did not actually use software agent technologies.

A generic online auction server was discussed in [23]. The server supports a flexible bidding language based on the OR/XOR formulae. Although, apparently there are many similarities with our own work, the details of the design and implementation of the system are actually lacking; only a listing of available technologies is provided. In particular, the interaction protocols and the details of the interfacing of agent and non-agent software are not described.

A configurable auction server was also proposed by the authors of [20]. This server targets resource allocation in the grid and therefore its design addresses the heterogeneity of the grid environment by allowing the dynamic configuration of the auction mechanism to meet the application requirements.

The authors of [26, 27] propose an agent-based infrastructure for autonomous services for management of the contracting of Cloud resources that covers also negotiation. Their system generates a service-level agreement – SLA representing the result of the resource negotiation and booking with available providers. The use of SLA has the advantage that it can be re-negotiated and monitored – a feature, which is missing from our approach. Moreover, while our work is more suitable for auctions, papers [26, 27] are focused on other negotiation mechanisms, like Contract-Net for example [28]. Thus, they are closer to our proposed negotiation framework introduced separately in [29].

3 DISTRIBUTED AGENT-BASED AUCTION SERVER

There are many definitions of the agent concept [30]. For the purpose of this work, by software agent (agent in what follows) we understand a software entity that: (i) has its own thread of control and can decide autonomously if and when to perform a given action; (ii) communicates with other agents by asynchronous message passing; (iii) can be referenced using its name, also known as agent identifier; (iv) can be located on an arbitrary machine in a computer network, providing that a certain runtime environment is locally available. This runtime is usually known as agent platform (see [25] for a recent overview of agent programming languages and platforms). In our current work, we use the Java Agent DEvelopment Framework – JADE [10] agent platform.

In this section we outline the architecture of our *Agent-based Auction Server*, highlighting agent types and relationships between them and users. Furthermore, we describe agent interactions: (i) inside the auction server and (ii) with external agent and non-agent software.

3.1 Agent Types and Their Functions

Let us now summarize the types of agents included in our auction server, focusing on their functionalities. The initial architecture and the agent interaction protocols of the server were introduced in [2]. In [11] we proposed an improved architecture that enables the deployment of the server on a computer cluster.

The auction server was designed to support the innovative concept of *generic agent-based auction service*. It is represented by a collection of cooperating agents that interact inside the server, as well as with its external environment, using agent interaction protocols. The software infrastructure of the server contains the types of agents depicted in the class diagram in Fig. 1. The auction server is actually composed of three main parts or layers: core, resource manager, and interface.

3.1.1 Interface Layer

The *Personal Agent*, *Participant*, and *Initiator Participant* agents compose the layer that realizes the interface of the server with its external environment.

The *Personal Agent* is residing on the server side and it connects the user with the auction server. For each user registered with the server there is exactly one *Personal Agent* created on the server. This agent gets input from the user, through an external user interface¿ This can be achieved directly, i.e. the *Personal Agent* can

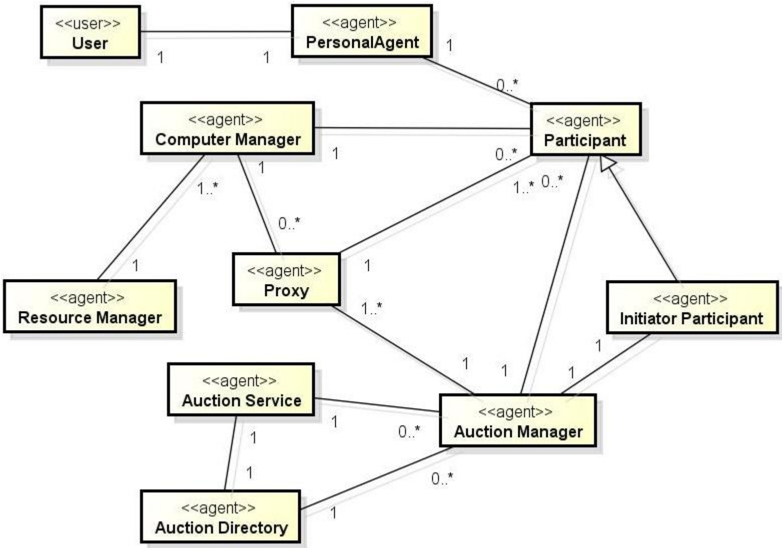


Fig. 1. Relationships between users, agent types and auctions on the auction server.

incorporate a user interface, or via a binding software that connects the *Personal Agent* with an external Web-based GUI. There is a one-to-one mapping between the user name and the identifier of the corresponding *Personal Agent*.

Whenever a human user connects to the server, usually using an external program (for example a GUI or a Web browser), a new *Personal Agent* is eventually created and assigned to the user (if no such agent is already active on the server). We setup our server to be used in a Web environment using the servlet technology on the server side (see Section 4 for details). In this case, it is the responsibility of the servlet to talk to the *Personal Agent*, when the user logged into the system, via an appropriate binding software.

The *Personal Agent* allows the user to perform the following operations:

- To create a new auction.
- To subscribe to an existing auction.
- To submit bids to one of his or her subscribed auctions, via his or her *Participant* agent.
- To receive notifications about status updates of his or her subscribed auctions.
- To receive accepts and rejects for his or her submitted bids.

Personal Agents can be enhanced with complex behaviors (like for example behaviors specific to Belief-Desire-Intention (BDI) architecture and agent-programming languages [35]) that would allow their truly autonomous operation for better serving the interests and goals of the human user. However, this path of investigation is outside the scope of this paper and it was left as future work.

Participant agent represents a *Personal Agent* that serves a user (usually with the *buyer* or *seller* role, although an *auctioneer* role can also be used, for example, in double auctions), registered and engaged into a particular auction. For each user registered to participate in an auction there is an associated *Participant* agent on the server. The *Participant* agents associated to a given user directly report to, and eventually get orders from, his or her *Personal Agent*.

Initiator Participant is a special *Participant* agent that represents the user, with the role to create and initiate the auction. For example, in an English auction the *Initiator Participant* represents the user that has the role of a *seller*, while the remaining *Participants* represent users that have the role of *buyers*. Usually, when a user initiates an auction he or she can specify also a condition that triggers the start of the auction (if this condition is missing then it is assumed to be *true* by default, which means that the auction will start immediately after creation). Example of starting conditions are: predefined starting time or registration of a minimum number of participants (this second condition is particularly useful for setting up experimental scenarios, see Section 5).

3.1.2 Core Layer

The core of the server is represented by *Auction Service*, *Auction Manager*, *Auction Directory*, and *Proxy* agents. This layer is responsible for the auctions' management and for the coordination of the auction participants by implementing the rules that govern the auction.

The *Auction Service* is the agent that manages all the active auctions registered with the server. This agent is the entry point of the auction service and it is responsible with creation of new auctions, as well as with registering of new participants to an active auction.

Auction Manager manages a single active auction on the server (also, known as auction instance in [2]). The *Auction Manager* coordinates the participants registered to that active auction. There is a separate *Auction Manager* agent for each active auction in the system. It implements a specific type of auction – English auction in this case, but in principle an *Auction Manager* can be configured to support an arbitrary auction type. The management of active auctions includes the activities that usually occur in an auction, i.e.: auction creation, bidding, agreement formation and auction termination. The *Auction Manager* is created by the *Auction Service* when a user wants to sell a product through an English auction. The *Auction Manager* has the following responsibilities:

- To request the creation and destroy of *Proxy* agents.
- To request the creation of *Participant* agents and to assign them to the right *Proxy* agent.
- To notify all the *Proxy* agents when the value of the highest bid was updated.

- To accept or reject the bids from *Participant* agents that are forwarded by *Proxy* agents.
- To manage the parameters of the auction, including: auction name, starting price, product name, starting and ending dates, as well as the status of the auction (for example, the currently highest bid).
- To trigger the auction termination when the time has expired by informing all the *Proxy* agents that the auction has finished.
- To record the winner and the final price of the auction.

Auction Directory agent manages the registry of active auctions, as well as the identifiers of their associated *Auction Manager* agents. Potential auction participants can search through this registry to find an active auction that meets their requirements.

Each *Proxy* agent handles the bids received from a subset of *Participant* agents. The *Participant* agents are split into disjoint groups, and each group is managed by a single *Proxy* agent. *Proxy* and *Participant* agents are linked into a balanced two-level hierarchical structure rooted at the *Auction Manager* such that the total number of *Proxy* agents is at most equal to the number of *Participant* agents that are linked to each *Proxy* agent. Using this balancing criterion we can reduce the time for processing incoming bids. During bidding, *Participant* agents communicate heavily with their *Proxy* agents, while *Proxy* agents pass on to the *Auction Manager* agent only the relevant bids and the other bids are filtered out and processed locally, thus reducing the amount of messages handled by the *Auction Manager* that results in enhancing the server response time. Note that, this is particularly important in the assumed model of an English auction (deadline-driven), where a large number of bids can materialize near the auction-deadline (due to the snipping).

Proxy agents form an intermediate layer between the *Auction Manager* and the *Participant* agents. Their main responsibility is to take over a part of the load that is necessary for bid processing that was initially the sole responsibility of the *Auction Manager* [2]. Each *Proxy* agent records a local currently highest bid (that can be different from the currently highest bid of the auction recorded by the *Auction Manager*) and updates it regularly based on the notifications received from the *Auction Manager* and bids received from *Participant* agents. The responsibilities of the *Proxy* agent are:

- To filter out the bids received from the *Participant* agents passing up to the *Auction Manager* only those bids that meet the improvement rule, i.e. are higher than the local currently highest bid known by the *Proxy*.
- To notify *Participant* agents about the acceptance or rejection of their bid.
- To receive notifications about the update of the currently highest bid from the *Auction Manager*.
- To notify *Participant* agents after receiving such an update from the *Auction Manager*; the local currently highest bid of the *Proxy* is also updated.

Let us suppose that, on average, a *Proxy* agent is managing k *Participant* agents and that the total number of *Proxy* agents is p. This means that the total number of participants is $n = p \times k$. According to our coordination model of an English auction, whenever a new bid is accepted by the *Auction Manager*, all the participants must be notified accordingly, and this process obviously will require a time $O(n)$. However, with the new *Proxy*-based hierarchical approach the time will be $O(p+k)$.

This value can be optimized to $O(\sqrt{n})$ for n *Participant* agents if we set the constraint that the number of participants managed by each *Proxy* agent will never outnumber the total number of *Proxy* agents connected to the *Auction Manager*, while the number of *Proxy* agents is increased whenever this is really necessary to manage all the registered *Participant* agents. Basically this means that $p \sim k$ from which we can derive the $O(\sqrt{n})$ average time needed to process a bid.

Proxy and *Participant* agents are linked into a balanced two-level hierarchical structure rooted at the *Auction Manager* such that the maximum number of *Participant* agents that are linked to each *Proxy* agent is at most equal to the total number of *Proxy* agents. The dynamic creation and destruction of *Proxy* agents will take into account the preservation of this balancing requirement.

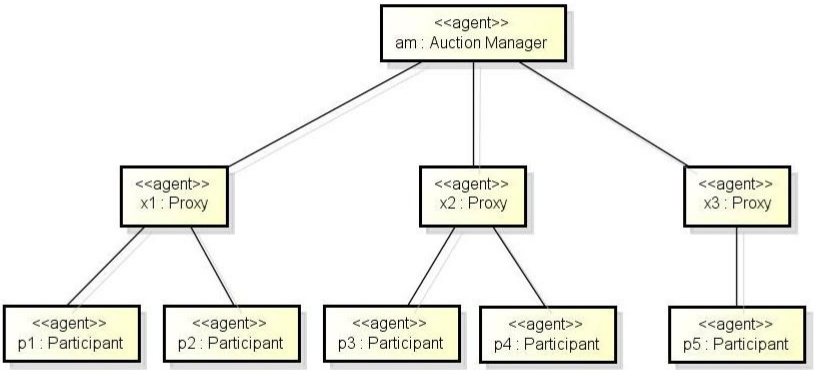


Fig. 2. Relationships between *Participant*, *Proxy*, and *Auction Manager* agents involved in an auction on the auction server.

An example showing the details of the relationship between the *Auction Manager*, the *Proxy* agents, and the *Participant* agents is presented in Fig. 2. This figure shows 5 *Participant* agents $p1$, ..., $p5$ registered to the same auction. Let us assume that the participants registered to the auction in this order. When participant $p5$ registered, only 2 *Proxy* agents $x1$ and $x2$ were present onto the server. According to the balancing criterion, the registration of the new *Participant* agent $p5$ triggered the creation of a new *Proxy* agent $x3$, as well as the link of $p5$ to $x3$. The new $p5$ *Participant* agent cannot be assigned to one of the existing *Proxy* agents because that would mean that at least one of them would have 3 *Participant* agents, while the number of *Proxy* agents is 2. So a new *Proxy* agent $x3$ must be created.

Requests for creating and destroying *Proxies* are issued by the *Auction Manager*. A *Proxy* agent is created when a new *Participant* is created and $n > p^2$ where n is the total number of *Participants* including the newly created one, and p is the number of *Proxies*.

When a *Participant* quits an auction, if $n \leq (p-1)^2$ then the *Proxy* with the smallest number of participants is deleted, where n is the total number of participants (after the removal of the current *Participant* from the server) and p is the current number of *Proxies*. The *Participant* agents that were linked to the *Proxy* that will be deleted will be evenly reallocated to the remaining *Proxy* agents with the smallest number of *Participants*.

Finally, there is an issue concerning what happens with the *Proxy* agents when the auction terminates. An English auction is running for a certain time duration, which is set when the auction is created. Optionally, this duration can be extended by the *Auction Manager* with a short period – timeout, in order to avoid problems created by "late bidding", by allowing all interested bidders to submit their bids. This is a solution used by auction servers to increase their "fairness" [5].

The timing of the auction is controlled by the *Auction Manager* agent. When this agent detects and declares the auction terminated, it will inform the *Proxy* agents about the auction termination and the auction result. Consequently, *Proxy* agents will reject forthcoming bids submitted by non-acceptably late bidders. Moreover, *Proxy* agents will notify accordingly their *Participant* agents, while *Participant* agents notify in turn their *Personal Agent*. Finally, *Participant* agents self-destroy. This process can optionally trigger up in the tree the self-destroy of *Proxy* agents, to maintain the tree balanced. Note that the process for managing the creation and destruction of *Participant* and *Proxy* agents can be improved using the technique of managing resource pools – in this particular case we would have agents pools on each computer of the network [36].

3.1.3 Resource Layer

The agents *Computer Manager* and *Resource Manager* constitute the part of the server that is responsible for the management of the computational resources, which run the server software.

When the server is installed on a computer network, the *Computer Manager* agents are responsible for the basic management (i.e. creation / destruction) of agents on each available machine. There is one *Computer Manager* agent for each computer that is part of the auction server infrastructure. This agent keeps track of all the *Participant* and *Proxy* agents that were created and are active on that computer. A *Computer Manager* agent gets requests from the *Resource Manager* agent that contain the agent name and the agent type and then creates the agent on that computer accordingly.

Resource Manager contains a registry of all the agents of type *Computer Manager* from the system. More exactly, when the *Auction Manager* decides to create a new *Participant* or *Proxy* agent, it will ask the *Resource Manager* to perform this operation. Currently, the *Resource Manager* agent keeps track of the number of agents already created on each available machine. On the basis of this information, it decides where should the new required agent be created and orders the creation accordingly to the corresponding *Computer Manager* agent. However, in a more general setting we can expand the functionality of the *Computer Manager* agents to monitor the resources and the network load of each single computer of the server. Then, the *Resource Manager* would be able to query *Computer Manager* agents before deciding on what computer to order the creation of new agents. This extension was left as future work.

3.2 Agent Interactions

Let us know summarize the protocols for interacting with the core of the auction server. Basically, they follow the same rules as the initial proposal of [3] that we also considered in [2]. According to these protocols, a user represented by his or her *Personal Agent* can create auctions, subscribe to active auctions, submit bids, receive replies about bid acceptance or rejection, receive notifications about the update of an auction status, and receive notifications about auction termination and auction winner.

Note that the interaction protocols presented and discussed in this section are focused on what is happening "inside" the auction server. However, the management of the link between a user and his or her *Personal Agent* is realized "outside" the auction server – see Section 4.

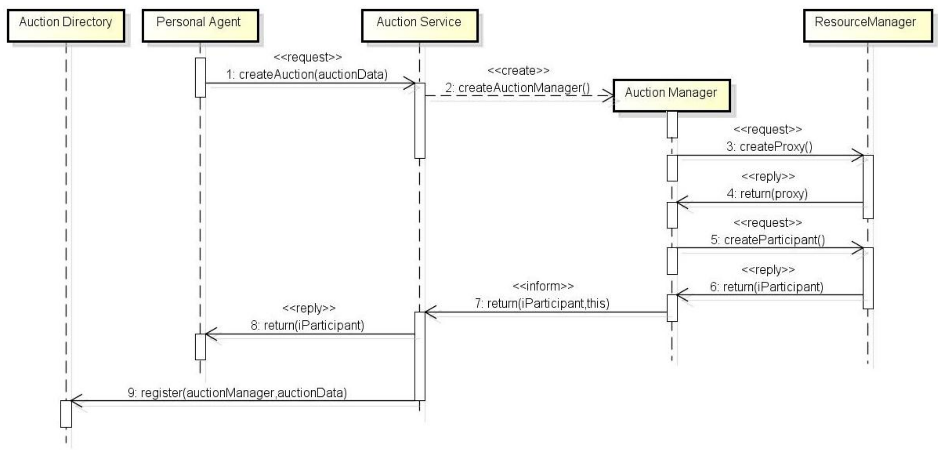


Fig. 3. Agent interactions for initiating an auction.

A new auction is created when a *Personal Agent* sends to the *Auction Service* a *createAuction* request (see Fig. 3). Then the *Auction Service* creates an *Auction*

Manager that represents the new auction. The *Auction Manager* orders creation of a new *Initiator Participant* agent linked to the *Personal Agent* of the user seller, as well as of a new *Proxy* for this participant, to the *Resource Manager*. The *Auction Service* then confirms to the *Personal Agent* the creation of the *Initiator Participant* agent. The auction description is also added to the *Auction Directory*. The *Initiator Participant* and *Auction Manager* agents will further interact during the auction process.

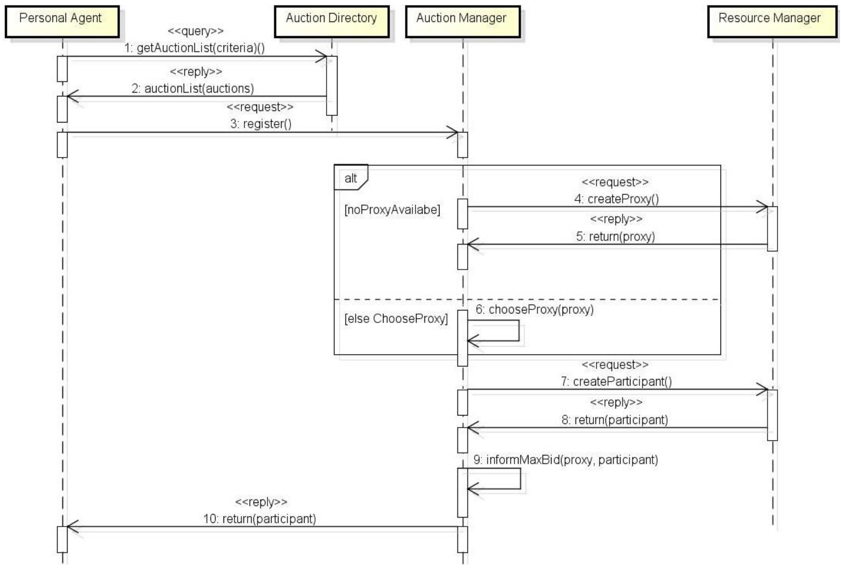


Fig. 4. Searching for and joining an auction.

A *Personal Agent* can query the *Auction Directory* about active auctions (see Fig. 4). The *Personal Agent* then chooses an auction and contacts the *Auction Manager* of that auction which should create a *Participant* for it.

The *Auction Manager* decides to create a new *Participant* agent and, optionally, a new *Proxy* agent. Their creation is handled by the *Resource Manager* agent, while the actual creation is performed by the *Computer Management* agent. The *Auction Manager* requests the *Resource Manager* agent to choose an appropriate computer where these agents will be created. The *Resource Manager* determines this computer and instructs the corresponding *Computer Manager* agent, to perform the creation action. When a computer reached the maximum acceptable load another computer should be used. If all computers are fully loaded then the computer with the smallest number of proxies or participants (in that order) is chosen; however, it is expected that the performance of the server will degrade in such situations. All *Participant* agents are created on the same computer as their *Proxy* agent.

The message exchanges that are needed for creating *Participant* and *Proxy*

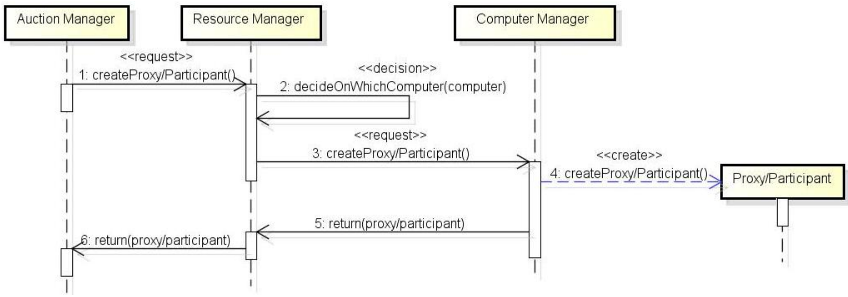


Fig. 5. Interaction protocol for automated creation of *Proxy* and *Participant* agents.

agents are shown in Fig. 5.

Each *Participant* agent sends his or her bids to their designated *Proxy* according to the protocol shown in Fig. 6. This bid is then compared by the *Proxy* to the currently best bid. Higher bids are reported to the *Auction Manager* while lower bids are rejected with a *refuseBid* message. Then the *Auction Manager* saves the highest bid (*bestBid* state variable) and refuses submitted bids that are lower than the *bestBid*. The refusal message is then returned by the *Proxy* agent back to each originating *Participant* that did not submit the current *bestBid*. If a bid higher than *bestBid* is received then the *Auction Manager* responds to the *Proxy* with a *bidAccepted* message, which is propagated to the originating *Participant*. If the *bestBid* value has changed then the *Auction Manager* informs all the *Proxy* agents. Then each *Proxy* will propagate the value of the new *bestBid* to its *Participant* agents. This forwarding is represented in Fig. 6 as a multicast message that is displayed as a little circle at the end of the message arrow.

4 INTEGRATION INTO AN ONLINE AUCTION SYSTEM

In this section we outline the design of an online auction system that incorporates the *Agent-based Auction Server*. The system design is composed of three parts: system architecture, interaction protocols, and design details of the system components.

4.1 System Architecture

The auction server was implemented using the JADE agent platform [10]. For experimenting with the usability of this auction server we have developed an online system equipped with a Web-based GUI that allows human users to create, search for and participate in English auctions. Therefore, the architecture will contain a special subsystem dedicated to the interface of JADE agent middleware with the Web-server.

The architecture of the application follows the classical separation between the

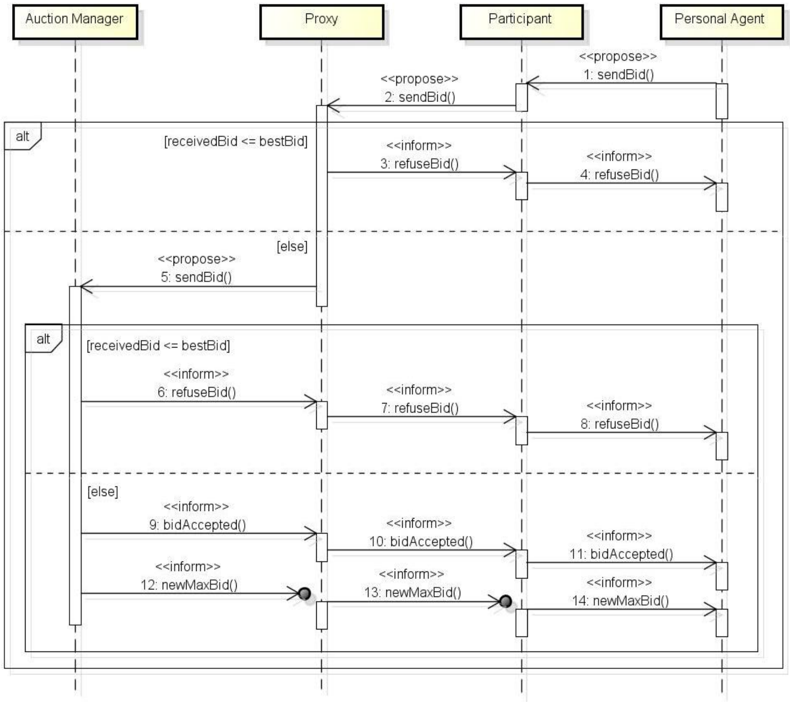


Fig. 6. The bid submission protocol.

client side, comprising the human user equipped with a Web browser, and the server side comprising a Web server that interacts with the auction server. Therefore we designed and developed a software for binding the Web server part that is non-agent software with the auction server that consists solely of JADE-based agent software.

The system has a multi-layer architecture composed of the following layers, as it is shown in Fig. 7:

- *User layer*. This layer represents the client part of the system that consists of a Web browser combined with HTML content including JavaScript code that is downloaded from the Web server.
- *Web layer*. This layer supports the user interaction functionalities. It consists of a Web server enhanced with a set of Java servlets that implement the user functionalities of the online application.
- *Binding layer*. This layer is represented by the software that enables the interfacing of the non-agent Web server software with the *Agent-based Auction Server*. This software is encapsulated into a special servlet called *Agent servlet* that is able to communicate with the JADE platform.

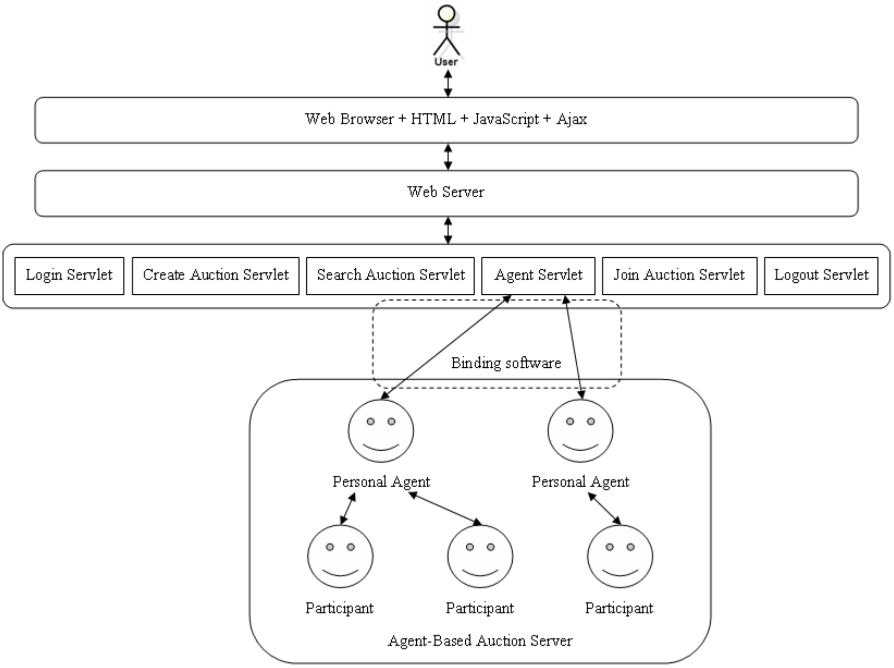


Fig. 7. System architecture.

- *Agent layer*. This layer represents the *Agent-based Auction Server* that was built on top of JADE platform.

4.2 Design Details

4.2.1 User and Web Layers

The *Web layer* is responsible for management of users and their accounts, while the auction server (i.e. the *Agent layer*) is responsible for auction management. With this separation of functionalities, the *Web layer* will support the interaction of the application with the human user, via a Web-based GUI that is based on HTML, Asynchronous JavaScript, and XML (i.e. Ajax [31]).

As we did not create a functionality for user authentication in the *Agent-based Auction Server*, we had to provide a solution for this problem at the level of the Web layer. So, in our prototype system, the Web layer is responsible for user authentication (login and logout functionalities) and management of user accounts. The addition of this functionality requires the Web layer to maintain a separate database for the management of user account information.

At the *User layer*, the servlets must provide HTML responses with information extracted from the *Agent layer* using software from the *Binding layer*. This operation can be time consuming and thus it can slow down the load time of the Web page. This issue was addressed by inserting JavaScript code into the HTML responses. This code allows the Web page to update itself quickly and efficiently using asynchronous requests. The JavaScript code issues automatic or user generated HTTP requests to the *Agent servlet* which was configured to reply with the XML responses. Note that the *Agent servlet* is the only servlet of the *Web layer* that can communicate with the agents on the *Agent layer* (via the *Binding layer*).

4.2.2 Binding Layer

The *Binding layer* communicates with the *Agent layer* using a specialized software. This software benefits from the JADE facilities for interfacing agent and non-agent software materialized as the *JadeGateway* class ([18]). The interface is achieved using agents (also known as *Gateway* agents) that are created locally by the *Agent servlet*. One *Gateway* agent is created for each user logged into the system. These local agents are created in a local container that is connected to the agent platform that hosts the auction server. This container is created and started together with the *Web layer*.

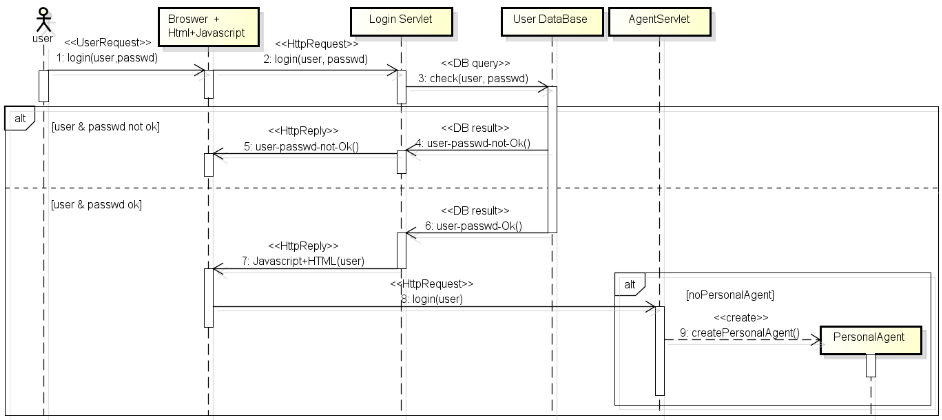


Fig. 8. User login.

Whenever a new user logins into the system, the *Agent servlet* automatically creates a local *Gateway* agent with the role of relaying messages from the *Web layer* to the agents on the auction server. Then the *Agent servlet* locally creates and passes a serializable object (called "blackboard object," in [18]) to the *Gateway* agent assigned to the current user. The *Gateway* agent then sends the message using the JADE messaging functionality to an agent located on the auction server, according to one of the interaction protocols presented in subsection 4.3. This type of interaction is marked with the ≪*object2agent*≫ stereotype in Fig. 8, Fig. 9, Fig. 10, Fig. 11, and Fig. 12. Conversely, whenever an agent of the auction server must send information to the *Web layer* it will use the JADE messaging to send this information to the corresponding *Gateway* agent located in the *Agent servlet*. Then the *Gateway* agent will invoke a method to update the "blackboard object" and thus achieving the correct transfer of the information to the *Agent servlet*. This type of interaction is marked with the ≪*agent2object*≫ stereotype in Fig. 10 and Fig. 11.

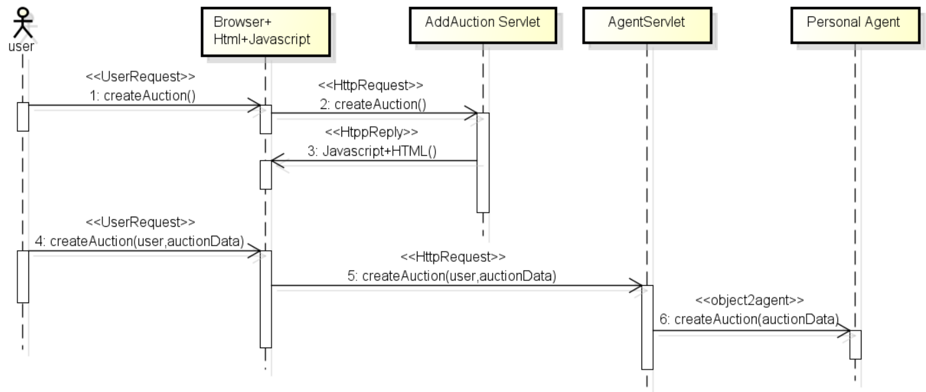


Fig. 9. Initiation of an auction.

Note that, whenever the JavaScript code of the Web page asynchronously requests an information update from the *Agent servlet* the servlet will respond with a message containing the relevant data extracted from the "blackboard" object and represented in the XML format. This type of interaction is represented with the ≪*XML response*≫ stereotype in Fig. 10 and Fig. 11.

4.3 Interaction Protocols

In this section we formally describe the interactions that happen between the software components of our auction system, "outside" the auction server, as opposed to the agent interaction protocols described in Section 3 that are focused on the "inside" of the auction server. Please note that although some of these interactions are related to the same activity – for example the bidding activity has a part inside the server, as well as a part outside the server, they are presented separately (in Section 3 and Section 4) for at least two reasons: i) their common part is minimal (it is reduced to the *Personal Agent*) and they can be well-understood separately; ii) the auction server is a separate subsystem that can be integrated in other types of applications, for example using a Web-service interface.

The *user login* operation is detailed in Fig. 8. The first part (interactions

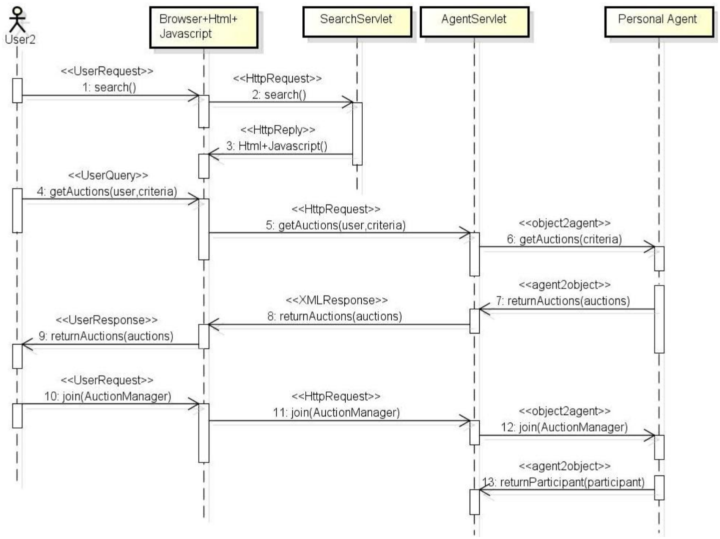


Fig. 10. Searching and joining an auction.

numbered from 1 to 7) achieves the authentication function. If the authentication is successful (i.e. the interactions proceed according to the branch consisting of messages 6 and 7) the *Agent-based Auction Server* is notified accordingly, via the interaction consisting of messages 8 and 9. Message 8 is a notification sent to the *AgentServlet* that the user logins into the system. Consequently, the link between the user name and the identifier of his or her *Personal Agent* is retrieved. Eventually, the *Personal Agent* of the user must be created and started (message 9), either if the user logins into the system for the first time or if his or her *Personal Agent* was offline.

The operation of *initiating an auction* is detailed in Fig. 9. The first 3 interactions activate the user menu for setting the auction data. The next 3 interactions support the function of creating a new auction. The actual creation is achieved after interaction 6. Note that we assume that when a new auction is created the user is already logged in and its *Personal Agent* is active.

The operation of *searching and registering at an active auction* is detailed in Fig. 10. The first 3 interactions activate the user menu for setting the search criteria for the desired auction. The next 6 interactions (numbered from 4 to 9) support the function of searching auctions available in the auction directory. This is achieved with the help of the *Auction Directory* agent, residing on the auction server. Then the user chooses the desired auction (this is achieved by interaction 10). Note that after this action the name of the corresponding *Auction Manager* agent becomes known. Finally, the last 3 interactions (numbered from 11 to 13) allow the user to join the desired auction.

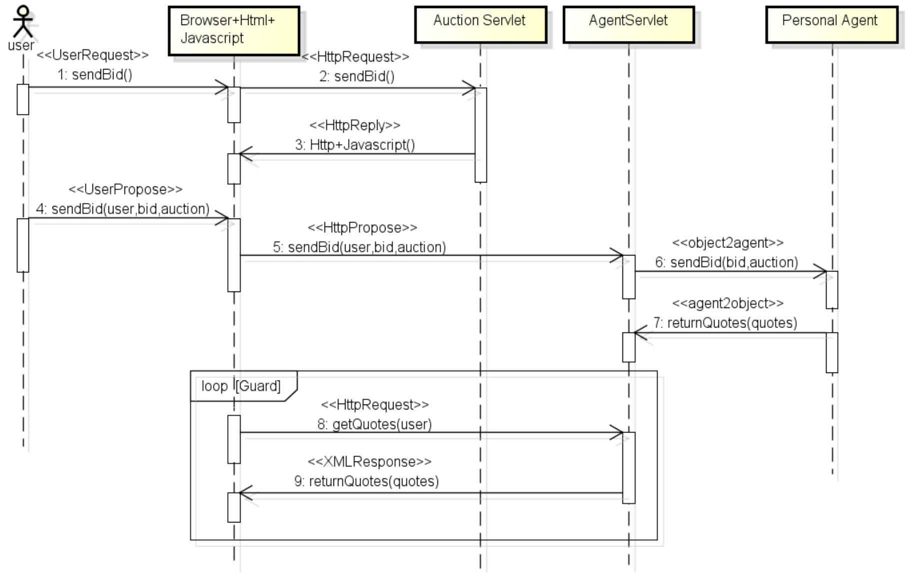


Fig. 11. Participating in an auction.

The operation of *bidding in an auction* is detailed in Fig. 11. The first 3 interactions activate the user menu for setting the bid data. The next 3 interactions (numbered from 4 to 6) support the function of submitting the bid to the auction server. Note that we assume that at this point the user knows the identifier of the auction where he or she wishes to submit the bid (parameter *auction*). Interaction 7 is happening whenever the *Personal Agent* is notified by the *Agent Layer* that at at least one of the auctions where the user is registered has updated its status. The last 2 interactions (8 and 9) allow the user to visualize the quotes of the auctions where he or she is subscribed. In particular, for an English auction, the user can check if a given bid was accepted or not, by the auction server. The updates are periodically triggered by a timer incorporated into the JavaScript code that runs in the user's browser.

The *user logout* operation is detailed in Fig. 12. Similarly to the login operation, the auction server is notified that the user is leaving the system. However, note that in this case the notification is sent directly to the *Personal Agent* that represents the user.

Finally, it is important to observe that there is an interesting relationship (not shown in Fig. 8 and Fig. 12) between the *Auction Service* and the *Personal Agent* that represents a specific human user on the auction server. This is the result of the fact that the *Personal Agent* has a very important role, by controlling the user participation in auctions, even when the user is disconnected from the online system. This fact has two important consequences:

- During the login operation the system must check if the user already has an active *Personal Agent* at the auction server, and if not it must create one. We assume that this operation is achieved by the *Auction Service* agent. So, the *Auction Service* agent has the responsibility of creating and setting up of a new *Personal Agent* according to user requirements. An example of such requirements could be: what to buy, the maximum price and the acceptable auction duration. Note that such requirements will become goals of the *Personal Agent*.

- During the logout operation the system must inform the *Personal Agent* that the user has left the system. However, the *Personal Agent* can behave more or less autonomously (according to the user requirements) in representing the user preferences and interests. So, the *Personal Agent* can autonomously decide to go offline in situations when, for example, there are no more active auctions in which the user is participating or, more generally, when a certain user goal was either achieved or is considered not achievable given the current state-of-affairs. Alternatively the *Personal Agent* can decide to continue its execution on the server.

Nevertheless, we set the requirement that the *Personal Agent* must always notify the *Auction Service* that it will go offline before doing so, such that if the user logins again onto the system then the *Auction Service* will be able to create and setup a new *Personal Agent* accordingly. The *Personal Agent* will autonomously decide to go offline whenever there is nothing left to do for the user. In particular this could happen when the user logouts and he/she is not currently involved in any auctions, as well as if there are no active goals of the *Personal Agent* agenda to be pursued. This can happen for example either when all the auctions where the user was involved are finished or when the user just logged in for the first time, did not create any auctions and did not set any requirements for the *Personal Agent* but it just decided to leave the system, i.e. to logout.

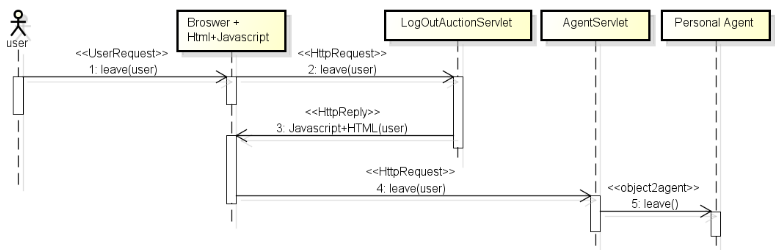


Fig. 12. User logout.

5 EXPERIMENTS AND USABILITY

5.1 Experiments with the Distributed Agent-Based Auction Service

We experimentally evaluated the current architecture by comparing it with our initially proposal from [2], where no *Proxies* were used. Then we conducted initial scalability experiments by running our system on 2 and 3 computers. For the experiments we used a network of dual core processors at 2.5 GHz and 1GB of RAM memory. These workstations were interconnected using a high-speed Myrinet interconnection network at 2Gb/s. According to [33], "Myrinet is a [...] high-performance, packet-communication and switching technology that is widely used to interconnect clusters of workstations". As multi-agent middleware platform we have used JADE 4.0 [34].

In this experiment the participants are allowed to bid automatically, so they were equipped with a bidding strategy to tell them if, when, and how (much) to bid. The bidding follows a snipping scenario, i.e. as soon as a participant receives a notification that he was outbid by another participant, he immediately submits a higher bid by adding a predefined increment to the value of the currently highest bid. During the snipping scenario the auction server is heavily loaded with bid processing activities.

The starting price of the auction was set to 0. The auction duration was set to 1.5 minutes in all the experiments. The increment value for the participants' strategy was set to 10. The agents were allowed to bid up to a very high reserve price (100000). The auction terminates when its allocated time expires. In our cases, to assure that we actually study the performance of the server, as the reserve prices were set to very high values, the auctions end before any of the agents reaches their reserve price.

Note that whenever a *Participant Agent* bids according to this strategy, it must receive an answer confirming if the bid was successful (or that it was rejected). This is very important, as we are in a distributed environment with multiple agents bidding concurrently, and it might happen that even if an agent is choosing a high enough value to bid, it might be outbid by another agent that submits its bid almost simultaneously. The following performance measures were recorded in our experiments:

- *Latency* = the average time it takes the system to answer a bid.
- *Throughput* = the number of bids handled per unit of time; this value is calculated by dividing the total number of answered bids by the duration of the auction.

The setup of the experiment assumes running a script that starts the JADE multi-agent platform and automatically creates the *Auction Service* and *Auction Directory* agents. Then the *Personal Agents* are created for each user that participates in the auction.

The *Personal Agent* that initiates the auction was configured to set the condition for starting the bidding when a specific number of participants has joined the auction. This condition is configured into the *Auction Manager* that governs the auction. When a certain given number of participants is reached, this *Auction Manager* will enable the starting of the bidding process. Basically, with this approach we were only looking for a simple method to setup our experiment consisting of many agents bidding aggressively in an auction, while keeping the consistency with the design philosophy of the auction server.

We ran our experiments using an increasing number of *Personal Agents* and we calculated the performance measures by running the framework on one, two and three computers. In order to also compare with our prior approach we also ran a version of the program without proxies, forcing the *Auction Manager* to handle all bids. In each test we ran only one auction. The case when multiple auctions are run in parallel was left as future work.

The results of our experiments are shown in Fig. 13 and Fig. 14, as well as in Table 1 and Table 2.

Table 1. LATENCY [MS]

Agents	No *Proxy*	1 Comp.	2 Comp.	3 Comp.
500	5087.53	2172.14	148.55	104.39
1000	20038.9	8114.66	266.85	217.01
1500	39752.5	8781.39	426.50	277.88
2000	66852.3	11691.2	575.06	452.73

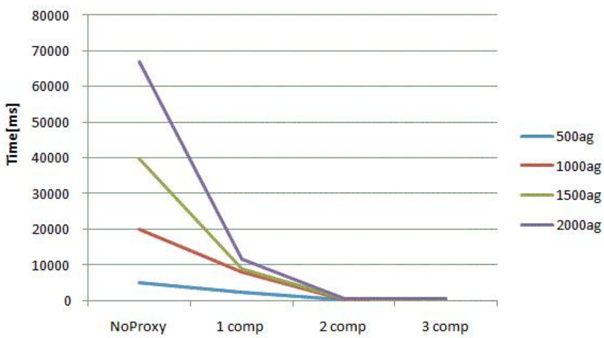


Fig. 13. Latency [ms].

Note that when we ran our system with 2000 bidding agents without proxies only 1273 agents actually got to bid at least once during the allocated time of 1.5 minutes.

Table 2. THROUGHPUT [NO. OF BIDS/MS]

Agents	No *Proxy*	1 Comp.	2 Comp.	3 Comp.
500	0.141	1.115	1.527	1.801
1000	0.065	0.949	1.714	1.744
1500	0.035	0.851	1.43	1.86
2000	0.025	0.779	1.502	1.672

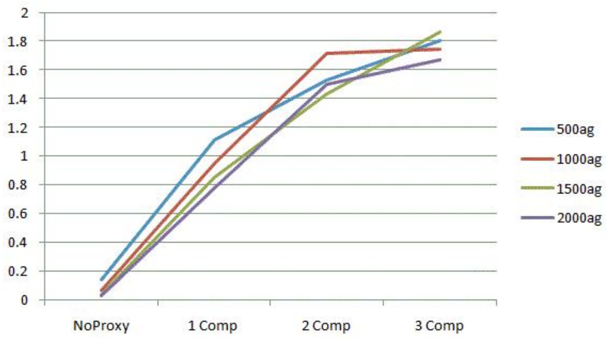


Fig. 14. Throughput [no. of bids/ms].

5.2 Usability of the Online Auction System

We now consider a sample use case involving four users U_1, U_2, U_3, U_4 participating in auctions with the help of our system. On this use case we highlight the usability aspects of the system, as well as some of the details regarding the information exchanged by the various components. In particular, we are interested in checking the information flow that is triggered into the system by user initiated actions. Please note that in this description we will make references to the diagrams introduced in Section 4.

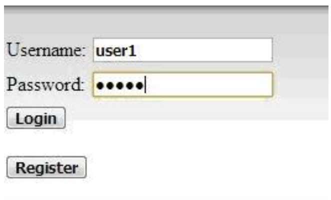


Fig. 15. Login GUI.

The users log-in into the system at the *User Layer* using the *Login GUI* in the

Web browser shown in Fig. 15. This operation, shown as message 1 in Fig. 8, is needed to communicate the username and password to the *Web Layer*. The login request reaches the *Login Servlet* (message 2 in Fig. 8 – an *Http Request* message). The user credentials are then verified and an HTML page is returned to the user presenting one of the following two possible outcomes:

- If the credentials are incorrect then an HTML page reports an error to the user's browser (message 5 in Fig. 8).
- Otherwise the HTML page contains JavaScript code for interacting with the *AgentServlet* (message 7 on in Fig. 8). After the interaction with the servlet (message 8 on in Fig. 8), a *Personal Agent* is created for each user.

In our sample use case, four *PersonalAgent*s (PA_1, PA_2, PA_3, PA_4) will be created in the *Agent Layer*.

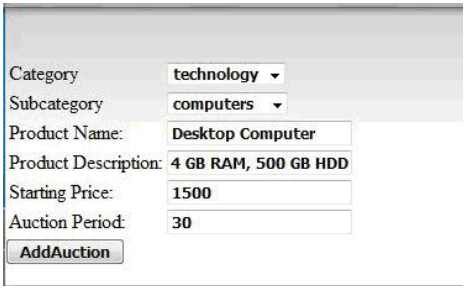


Fig. 16. Create Auction GUI.

We assume that user U_1 creates two auctions labeled A_1 and A_2 (their details are shown in Table 3). For each auction, participant (initiators in this case) *IPart1*, *IPart2* agents, as well as *AuctionManager1* and *AuctionManager2* agents are created. Interaction 1 in Fig. 9 represents the initial request sent by U_1 for creating an auction. Interaction 2 in Fig. 9 represents the *HTTP Request* message sent to the *AddAuction Servlet*. This servlet responds with an HTML file containing also JavaScript code for creating the form *Create Auction GUI* shown in Fig. 16. The form enables the user to input the product descriptions as shown in Table 3 (message 4 in Fig. 9). These are then passed to the *AgentServlet* using message 5 in Fig. 9. Finally, *AgentServlet* requests PA_1 agent (using message 6 in Fig. 9) to create the auction in the *Agent Layer*.

We now assume that users U_2, U_3 and U_4 are searching for a desktop computer which is provided by auction A_1, while user U_3 is also looking for a TV which is provided by auction A_2. Searching for a GUI starts when a user issues a request that triggers messages 1 and 2 in Fig. 10. The *SearchServlet* responds with an HTML form that represents the *Join Auction GUI* (see Fig. 17). Then the use communicates his or her search parameters to the system (message 4 in Fig. 10).

Table 3. Auction details

auction	A1	A2
product name	Desktop	TV
description	I3, 4GB RAM, 500 GB HDD	LCD, 81cm
starting price	1500 RON	800 EUR
increment	10 RON	5 RON
auction duration	3 h	5 h

The search parameters are passed to the *Auction Directory* using messages 5, 6 in Fig. 10. The *Auction Directory* replies by triggering the sequence of messages 7, 8 and 9. The result contains the description of a list of auctions as an XML file (more precisely message 8 in Fig. 10). The user can choose the auctions they want to join by triggering interactions 10, 11 and 12. Whenever a user joins an auction, a new *Participant* agent is created to represent the user bidding in that auction (this is achieved via message 13 in Fig. 10). In our scenario *Part2*, *Part3*, and *Part4* agents are created to represent users U_2, U_3 and U_4 acting in auction A_1, as well as *Part5* agent is created to represent user U_3 acting in auction A_2.

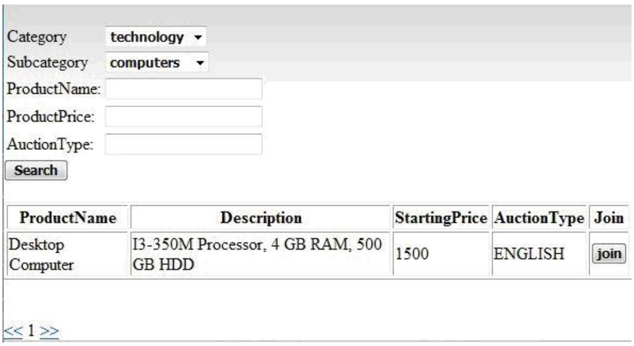


Fig. 17. Search and Join Auction GUI.

In what follows we shall focus only on what happens in auction A_1. Let us assume that user U_2 bids 1510 RON and his bid is accepted. Message 5 in Fig. 11 contains the bid information that user U_2 sends to the *AgentServlet*: the bid value 1510 RON and the participant id *Part2*. The bid value is input by the user via the GUI shown in Fig. 18. The *AgentServlet* determines the identifier of the *PersonalAgent* attached to user U_2, i.e. PA_2 and then forwards the bid information to PA_2 (message 6 in Fig. 11). Whenever a *PersonalAgent* receives a notification about the update of the currently highest bid, it notifies the *AgentServlet* (message 7 in Fig. 11). The GUIs of all the users participating in auction A_1 (i.e. U_2, as well as U_3 and U_4 via their PA_3 and PA_4 agents) are automatically updated about the currently highest bid by the JavaScript code that periodically retrieves from the *AgentServlet* the updated information encoded an XML message (message 9 in Fig. 11). In this example the XML message contains the information shown on Table 4.

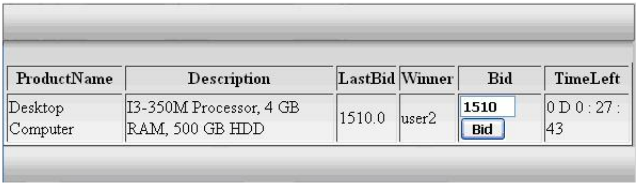


Fig. 18. Participating in an Auction GUI.

Table 4. Currently highest bid update.

DTD	Content
<!DOCTYPE Auctions [	<Auctions>
<!ELEMENT Auction (Participant,	<Auction>
BestPrice, Bidder)*>	<Participant>Part2</Participant>
<!ELEMENT Participant (#PCDATA)>	<BestPrice>1510</BestPrice>
<!ELEMENT BestPrice (#PCDATA)>	<Bidder>U2</Bidder>
<!ELEMENT Bidder (#PCDATA)>	/Auction>
]>	</Auctions>

In what follows let us assume that user U_2 logouts. However, his or her *PersonalAgent*, i.e. PA_2 is kept alive on the *Agent-based Auction Server* and continuously receives updates as auction A_1 is proceeding. Now, assuming that U_3 and U_4 both bid 1520 RON, with U_3 being slightly faster than U_4, the bid of U_3 is accepted, while the bid of U_4 is refused. Now, as U_4 is not happy that his or her bid was rejected, U_4 will submit a new higher bid of 1530 RON. At this point U_3 resigns the auction. Let us now assume that U_2 logs in again. It will be automatically informed by PA_2 agent that the currently nighest bid is 1530 RON and it was submitted by U_4, as shown in Fig. 19.

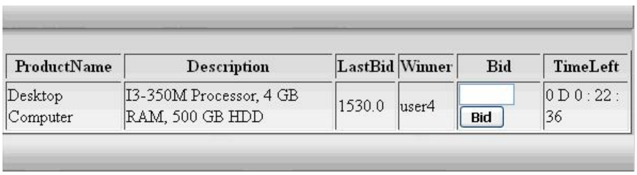
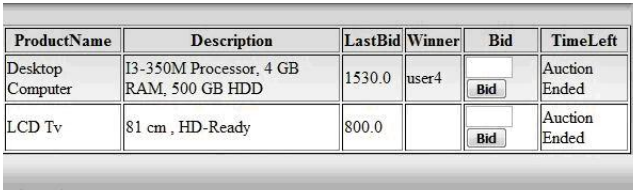


Fig. 19. New updated highest bid.

Now, let us assume that U_2 decides to place a new bid of 1540 RON, but meanwhile the auction time expired. The bid submitted by U_2 will be ignored by the server. The server notifies all the users, including U_2, about the outcome of auction A_1. Assuming that auction A_2 is also finalized without any winner (more exactly, no bids were submitted in this auction), U_2 will receive the notification shown in Fig. 20.



ProductName	Description	LastBid	Winner	Bid	TimeLeft
Desktop Computer	I3-350M Processor, 4 GB RAM, 500 GB HDD	1530.0	user4	Bid	Auction Ended
LCD Tv	81 cm , HD-Ready	800.0		Bid	Auction Ended

Fig. 20. Final notification.

6 CONCLUSIONS

In this paper we described in details an improved agent-based architecture for an English auction server. The initial experimental results show that our hierarchical scheme of structuring the server using *Proxy* agents and a simple balancing scheme is effective and has good scalability, when the server is distributed on multiple machines. As future work we plan to: (i) strengthen the results by performing more experiments on larger networks; in particular we will target experiments on high-performance computer clusters; (ii) extend the architecture to other types of auctions.

This paper also introduced our design and implementation of an online auction system that incorporates the *Agent-based Auction Server*. The system provides a Web-based GUI for the *Agent-based Auction Server*. We outlined the main functionalities of the system, as well as their design and implementation, in terms of system architecture, design details and interaction protocols. The main outcome of our work is a clean specification of the Web-based and agent-based software layers of our system, as well as of their software interfaces. As future work we plan to: (i) expand our design by providing a Web services interface to our *Agent-based Auction Server*; (ii) investigate the relation between the human user and his or her *Personal Agent*, in particular on how human requirements in the area of auctioning and e-commerce can be mapped onto elements of the *Personal Agent* architecture, like for example those related to the BDI model.

REFERENCES

[1] Bădică, C.—Ganzha, M.—Paprzycki, M.: Implementing Rule-Based Automated Price Negotiation in an Agent System. Journal of Universal Computer Science, Vol. 13, 2007, No. 2, pp. 244–266.

[2] Dobriceanu, A.—Biscu, L.—Bădică, A.—Bădică, C.: The design and implementation of an agent-based auction service. International Journal of Agent-Oriented Software Engineering, Vol. 3, Inderscience, 2009, No. 2/3, pp. 116–134.

[3] Bartolini, C.—Preist, C.—Jennings, N. R.: The design and implementation of an agent-based auction service. Lecture Notes in Computer Science, Vol. 3390, Springer, 2005, pp. 213–235.

[4] Bădică, C.—Giurca, A.—Wagner, G.: Using Rules and R2ML for Modeling Negotiation Mechanisms in E-commerce Agent Systems. Lecture Notes in Computer Science, Vol. 4473, Springer, 2007, pp. 84–99.

[5] Stubblebine, S.—Syverson, P.: Fair On-line Auctions Without Special Trusted Parties, In: Lecture Notes in Computer Science, Vol. 1648 Springer, 1999, pp. 230–240.

[6] Rolli, D.—Luckner, S.—Gimpel, H.—Weinhardt, C.: A descriptive auction language. Electronic Markets, Vol. 16, 2006, No. 1, pp. 51–62.

[7] Ockenfels, A.—Reiley, D.—Sadrieh, A.: Online Auctions. In: Hendershott, T. (Ed.): Economics and Information Systems. Emerald Group Publishing, 2006, pp. 571–628.

[8] Shoham, Y.—Leyton-Brown, K.: Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. MIT Press, 2009.

[9] Hillston, J.—Kloul, L.: Performance investigation of an on-line auction system. Concurrency and Computation: Practice and Experience, Vol. 13, 2001, No. 1, pp. 23–41.

[10] Bellifemine, F. L.—Caire, G.—Greenwood, D.: Developing Multi-Agent Systems with JADE. John Wiley & Sons, 2007.

[11] Sandu, L.—Sbora, R.—Ilie, S.—Bădică, C.: Scalable distributed agent-based English auction server. In: Proceedings of the 15th International Conference on System Theory, Control, and Computing (ICSTCC'2011), IEEE, 2011, pp. 1–6.

[12] Ilie, S.—Bădică, C.—Bădică, A.—Sandu, L.—Sbora, R.—Ganzha, M.—Paprzycki, M.: Information flow in a distributed agent-based online auction system. In: Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics (WIMS'12), ACM, 2012, pp. 42.

[13] Benyoucef, M.—Rinderle, S.: Modeling e-negotiation processes for a service oriented architecture. Group Decision and Negotiation, Vol. 15, Springer, 2006, No. 5, pp. 449–467.

[14] Broberg, J.—Venugopal, S.—Buyya. R.: Market-oriented Grids and Utility Computing: The State-of-the-art and Future Directions. Journal of Grid Computing, Vol. 6, Springer, 2008, No. 3, pp. 255–276.

[15] Fasli, M.—Michalakopoulos, M.: e-game: A platform for developing auction-based market simulations. Decision Support Systems, Vol. 44, Elsevier, 2008, No. 2, pp. 469–481.

[16] Griggs, K.—Wild. R.: Intelligent support for sophisticated e-commerce services: An agent-based auction framework modeled after the new york stock exchange specialist system. e-Service Journal, Vol. 2, Indiana University Press, 2003, No. 2, pp. 87–104.

[17] Su, Y. W. S.—Huang, C.—Hammer, J.—Huang, Y.—Li, H.—Wang, L.—Liu, Y.—Pluempitiwiriyawej, C.—Lee, M.—Lam. H.:An Internet-based negotiation server for e-commerce. The VLDB Journal, Vol. 10, 2001, No. 1, pp. 72–90.

[18] Kelemen, V.: Jade tutorial: Simple example for using the JadeGateway class. Available on: `http://jade.cselt.it/doc/tutorials/JadeGateway.pdf`, 2006.

[19] Trevathan, J.—Read, W.—Balingit. R.: Online auction software fundamentals. International Proceedings of Computer Science and Information Technology, Vol. 2, 2009, pp. 254–259.

[20] Vilajosana, X.—Krishnaswamy, R.—Marquès. J. M.: Design of a configurable auction server for resource allocation in grid. In: Proceedings of International Conference on Complex, Intelligent and Software Intensive Systems, CISIS'09, 2009, pp.396–401.

[21] Wasielewska, K.—Gawinecki, M.—Paprzycki, M.—Ganzha, M.—Kobzdej. P.: Optimizing blackboard implementation of agent-conducted auctions. IADIS International Journal on WWW/Internet, Vol. 6, 2008, No. 1, pp. 50–60.

[22] Wurman, P. R.—Wellman, M. P.—Walsh, W. E.: The michigan internet auctionbot: A configurable auction server for human and software agents. In: Second International Conference on Autonomous Agents, Agents-98, 1998, pp. 301–308, 1998.

[23] Yao, D. Q.—Qiao, H.—Qiao. H.: A generic internet trading framework for online auctions. In: A. Becker (Ed.): Electronic Commerce: Concepts, Methodologies, Tools, and Applications, IGI Global, 2008, pp. 163–177.

[24] Muscar, A.—Bădică, C.: Exploring the Design Space of a Declarative Framework for Automated Negotiation: Initial Considerations. IFIP Advances in Information and Communication Technology, Vol. 381, Springer, 2012, pp. 264–273.

[25] Bădică, C.—Budimac, Z.—Burkhard, H. -D.—Ivanović, M.: Software agents: Languages, tools, platforms. Computer Science and Information Systems, Vol. 8, 2011, No. 2, p. 255–298.

[26] Venticinque, S.—Aversa, R.—Di Martino, B.—Petcu, D.: Agent based cloud provisioning and management: Design and prototypal implementation. In: Proceedings of the 1st International Conference on Cloud Computing and Services Science: CLOSER'2011, 2011, pp. 184–191.

[27] Amato, A.—Liccardo, L.—Rak, M.—Venticinque, S.: SLA negotiation and brokering for sky computing. In: Proceedings of the 2nd International Conference on Cloud Computing and Services Science: CLOSER'2012, 2012, pp. 611-620.

[28] Smith, R. G.: The contract net protocol: High-level communication and control in a distributed problem solver. IEEE Transactions on Computers, Vol. 29, 1980, No. 12, pp. 1104–1113.

[29] Scafeş, M.—Bădică, C.—Pavlin, G.—Kamermans, M.: Design and Implementation of a Service Negotiation Framework for Collaborative Disaster Management Applications. In: Proceedings of the 2nd International Conference on Intelligent Networking and Collaborative Systems, INCoS 2010, pp. 519–524.

[30] Wooldridge, M.: An Introduction to MultiAgent Systems - Second Edition. John Wiley & Sons, 2009.

[31] Zakas, N. C.—McPeak, J.—Fawcett, J.: Professional Ajax, 2nd Edition. Wrox, 2007.

[32] Fasli, M.: Agent Technology for E-Commerce. John Wiley & Sons, 2007.

[33] Myrinet Overview, `http://www.myricom.com/scs/myrinet/overview/`. Accessed in November 2012.

[34] JADE: Java Agent Development Framework. `http://jade.cselt.it`. Accessed in November 2012.

[35] Bordini, R. H.—Hübner, J. F.—Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason, John Wiley & Sons, 2007.

[36] Lu, J.—Gokhale, S. S.: Performance analysis of a Web server with dynamic thread pool architecture, In: Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering: SEKE'2010, 2010, pp. 99–105.

Costin Bădică works in the Department of Software Engineering. ...

Sorin Ilie works in the Department of Software Engineering. ...