Experiments of Running Parallel Auctions in an Agent-Based Auction Service

Adriana Dobriceanu
University of Craiova, Software Engineering Department
Bvd.Decebal 107, Craiova, 200440, Romania
dobriceanu_adriana@software.ucv.ro

Laurenţiu Bîscu
University of Craiova, Software Engineering Department
Bvd.Decebal 107, Craiova, 200440, Romania
biscu_laurentiu@software.ucv.ro

Amelia Bădică
University of Craiova, Business Information Systems Dept.
A.I.Cuza 13, Craiova, 200585, Romania
ameliabd@yahoo.com

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

Abstract

In this paper we discuss experimental results obtained with a prototype implementation of an agent-based service for generic auctions that is currently under development. We configured the service to allow parallel execution of multiple English auctions with buyer and seller agents bidding from different machines. The experimental results to assess the performance of the service include latency and throughput of the service as functions of the number of participants and the number of simultaneously active auctions.

1. Introduction

Traditionally, negotiations were defined and studied as human-driven processes that are conducted for conflict resolution in business communications. Electronic negotiations, including auctions, spread a lot with the advent of the Internet and the Web. Recently, agent-based e-commerce research started to spend a considerable effort on the study, design and implementation of automated negotiations [9].

Auctions have many practical applications in conducting economic transactions including selling treasury bills, foreign exchange, mineral (eg. oil-drilling) rights, radio spectrum licences, as well as for awarding government contracts in procurement and sub-contracting activities. Therefore, the interest for advancing digital technologies to support auction applications increased explosively during the last decade. Recently, the research focus has shifted from the development of Web-based specialized auction applications to provisioning of more process-generic, flexible and reusable solutions with an increased potential for the B2B sector [6]. In this context, agent-based service orientation is proposed as a new approach that provides improved levels of flexibility and reusability in development of process-generic automated negotiations [5, 12].

Our recent work in this area was focused on the study, design and implementation of an open flexible infrastructure for service-based automated negotiations in agent systems. The results consisted in proposing an innovative solution that integrates best features of: i) generic software framework for automated negotiations [4, 9, 8]; ii) market architecture for auction development [14]; iii) rule-based declarative representation of auction mechanisms [10, 11].

In this paper we present experimental results obtained with the application of our solution on a specific scenario of an agent-based English auction service [12, 11]. The service was updated to support several English auctions to be run in parallel with participant agents submitting bids from different machines. The results confirmed that, while the solution allows an increased level of flexibility and reusability as compared with existing approaches, it still suffers from performance and scalability problems that we plan to address in the near future.

The paper is organized as follows. We start by reviewing our proposed architecture of an agent-based auction service. We follow in Section 3 with description of experiments and results of system performance as a function of number of participants and number of active auctions (thus extending our results from [12]). The last section of the paper concludes and points to future work.

2. An Agent-Based Auction Service

Electronic auctions spread a lot with the advent of the Web. Now there are many providers of Web auctions and thus, it may be very difficult for a buyer or a seller to find the most appropriate auction to meet her needs. Usually this

978-1-4244-2881-6/08/$25.00 ©2008 IEEE

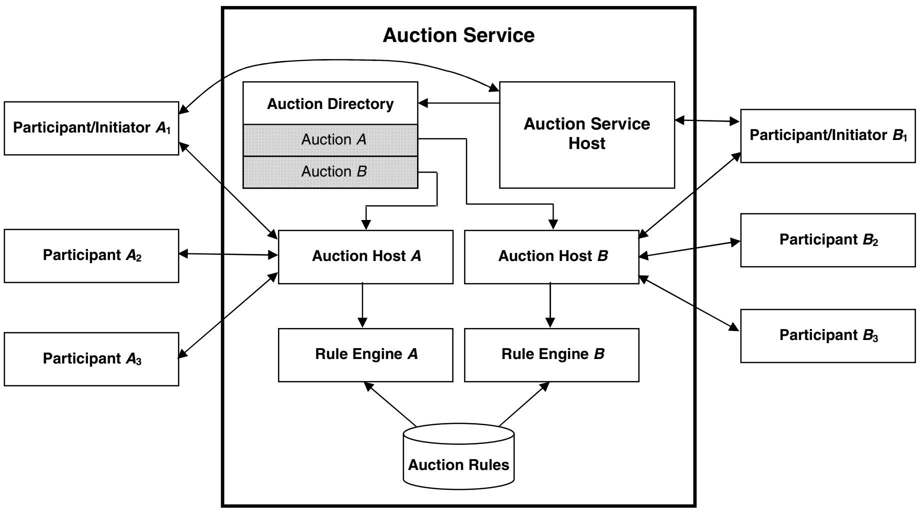

Figure 1. An Auction Service

task is performed manually using a Web browser and a general purpose search engine.

An improvement of this approach is the provision of *auction directories*. An *auction directory* provides an up-to-date list of the most popular auction sites that allows a human to easily navigate or search for an appropriate auction. An advanced feature that is more suitable to automation using software agents, is the *meta-auction* [7]. A meta-auction proposes the integration of several auction sites by means of a common global schema. This facility allows potential buyer agents to automatically search across several auction sites for the items of interest. However, the meta-auction approach suffers from well-known integration schema and protocol problems, thus requiring an increased development effort and an additional complexity associated with the integration middleware.

Our work takes this idea of auctions further from the human-driven Web world to the software agents world by proposing the new concept of a *generic agent-based auction service*. Following [14] we consider a market environment providing one or more types of auction services. Trading partners represented by buyer and seller agents can discover appropriate auction services matching their goals of buying and/or selling goods or more general resources.

Using a service-oriented approach, a market environment can be conceptualized as providing one or more *Auction Services* (*AS*) ([12]). An *AS* (see figure 1) implements a specific type of auction and it has two important functions: i) management of active auctions including activities like auction creation and auction termination; ii) coordination of auction participants, including activities like offer and counter-offer submission (bidding) and clearing (agreement formation), for each active auction within the service. Each *AS* includes a collection of agents that collaborate to support the *AS* to achieve its functions.

An *AS* contains a local directory of active auctions – the *Auction Directory* (*AD*). An active auction – *auction instance* (*AI*) is a process that coordinates behaviors of participants registered with that particular auction. An *AS* is managed by a specialized agent known as *Auction Service Host* or *ASH*. *ASH* serves as *AS* entry point and she is also in charge with creation, deletion and management of *AI*s.

An agent (buyer or seller) that is registered to participate in an auction hosted by an *AS* is called *Auction Participant* or *AP*. Additionally, an auction is initiated by a special *AP* called *Auction Initiator Participant – AIP*. Initiation of an *AI* determines the starting of its associated process.

We have conceptualized an *AI* using the negotiation model inspired from [4] and [9]. An alternative model that uses state-charts to represent negotiation protocols is presented in [5]. However, please note that our model is more flexible and generic, as it allows to easily update the auction mechanism via a declarative representation [10], [12], while preserving most of the software infrastructure including agent types and the generic negotiation protocol.

Within our model, an *AI* is managed by a specialized arbitrator middle-agent known as the *Auction Host* (*AH*). An *AH* is responsible for coordinating the behavior of agents participating to a single *AI*. An *AH* incorporates a rule engine that checks the rules for that auction type. Note that, as we assumed that an *ASH* provides a certain and unique type of auction – eg. English, Dutch, Vickrey, multi-item, a.o., we have a single set of rules for each *ASH*. However, for each active *AH* within the *ASH* and its associated *AI* we must create a separate rule engine that is instantiated with the set of rules for the auction type supported by the *ASH*.

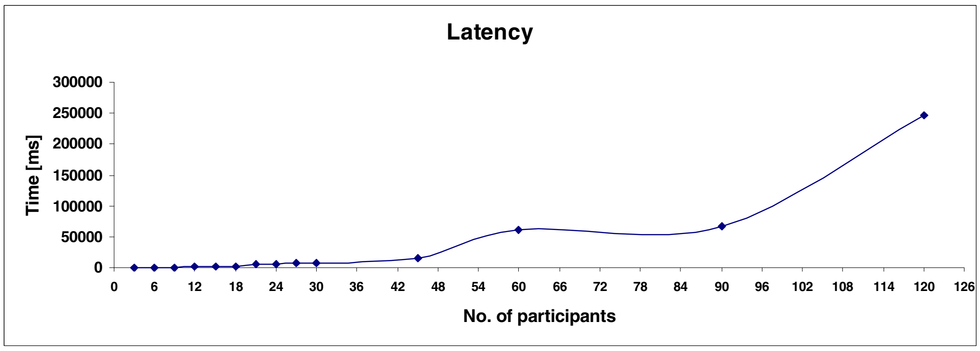

Figure 2. Latency for 3 buyers per negotiation

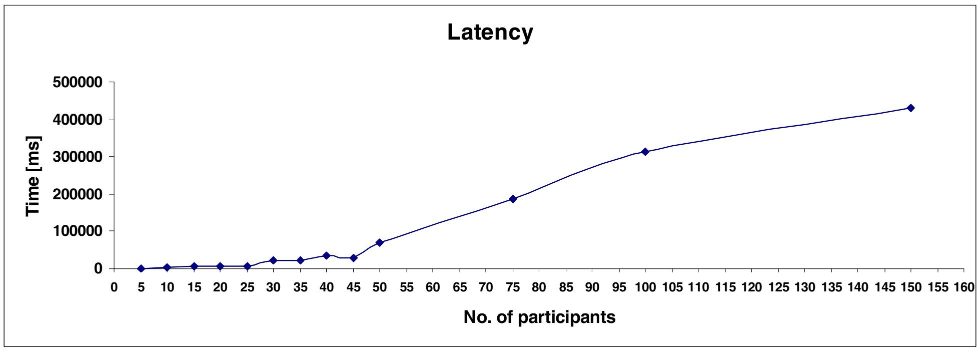

Figure 3. Latency for 5 buyers per negotiation

For each active *AI* inside an *AS*, there is an associated entry in *AD* that contains the identifier of the corresponding *AH* together with an appropriate description of the *AI*. Creation and deletion of *AH*s together with their registration and respectively deregistration with the *AD* are handled by the *ASH* agent.

Figure 1 shows an auction service – let us suppose the service provides English auctions. The service is currently running two *AI*s – *Auction A* and *Auction B*. Note that both *Auction A* and *Auction B* are in fact English auctions, governed by similar negotiation mechanisms, but characterized by different parameters – different auctioned product, different starting time and ending time, different minimum bid increment, different sets of negotiation participants, a.o. Three participants are registered to *Auction A* – *Participant/Initiator* A_1, *Participant* A_2, and *Participant*A_3. Note that *Participant/Initiator* A_1 is the initiator of *Auction A*.

Note that two consequences of borrowing the [4] negotiation model are: i) *AP*s communicate with the *AH* using a *generic negotiation protocol* that is independent of the particular auction type and thus assuring an increased level of interoperability and domain independence. An auction is seen as the process of exchanging proposals (or bids) via a common space that is governed by an authoritative entity – *AH* (having the role of negotiation host from [4]). Status information describing auction state and intermediary information is automatically forwarded by the host to all entitled participants according to the information revealing policy of that particular auction (see [4] for details); ii) *Rules* are used for enforcing the auction mechanism. Rules are organized into a taxonomy: rules for participants admission to negotiations, rules for checking the validity of proposals, rules for protocol enforcement, rules for updating the negotiation status and informing participants, rules for agreement formation and rules for controlling the auction termination (see [9, 10] for details). The use of rules greatly increases the flexibility and the reusability of the approach.

We have extended the prototype implementation of the generic auction service initially presented in [12] following the design outlined in this paper using JADE 3.5 agent platform [1] and JESS [2]. The service was configured to run single-item English auctions. For this purpose we have reused parts of the implementation discussed in [9] including infrastructure and rules for describing an English auction, and we have extended it with functions for management of multiple active auctions within the service. We have

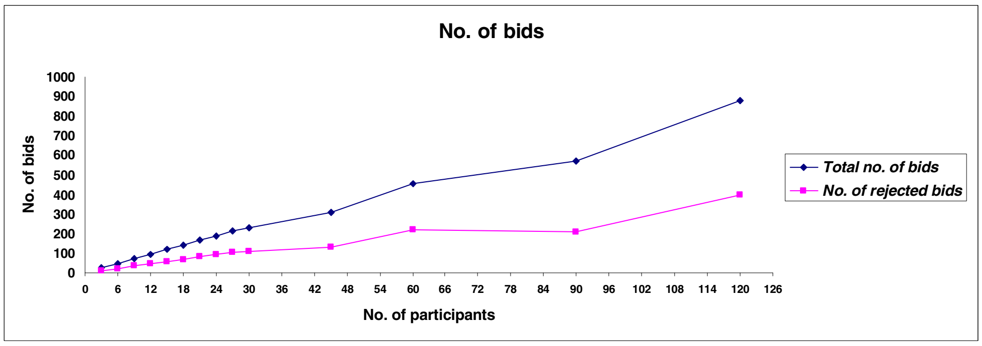

Figure 4. No. of bids for 3 buyers per negotiation

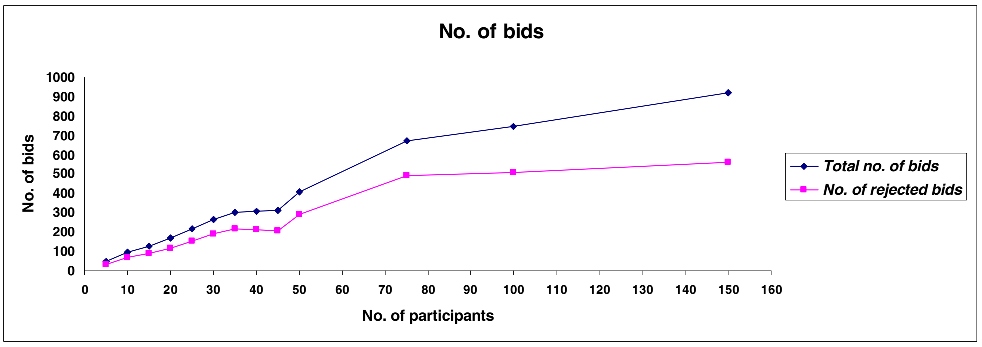

Figure 5. No. of bids for 5 buyers per negotiation

implemented the following types of agents: *ASH*, *AH*, *AP* (in the role of buyer) and *AIP* (in the role of seller). Currently, *AD* is implemented as a list data structure that is located internally to the *ASH*, rather than as a separate agent. Following [9], each *AH* incorporates a Jess rule engine [2] that is initialized with rules for describing English auctions.

3. Experiments and Discussions

We have performed several experiments using this implementation to asses the performance of the service as a function of the number of auction participants and the number of simultaneously active auctions. We measured: i) the *latency* of the service i.e. the time elapsed since a participant (in this case a buyer) submitted a bid until it receives a response from the service; ii) *number of bids* as a function of the number of participants and iii) *throughput* as the ratio indicating the number of bids processed by the service per unit of time. There was considered a single *AS* with its corresponding *ASH*, but with multiple auctions running in parallel, each driven by its own *AH* agent. Note that each *AH* agent also incorporated a JESS rule engine, initialized with the rules for describing English auctions.

The experiments were setup on 4 ordinary machines, with 1 machine running the *AS* and its associated infrastructure, and the remaining 3 machines running *AP*s – buyers and sellers. The machine running the service was a PC laptop AMD Turion 64x2 Mobile at 1.6 GHz with 2x512MB of RAM DDRII. The other 3 machines were: (i) PC laptop Intel(R) Core(TM)2 Duo 1.66 GHz with 1GB of RAM (ii) PC laptop Intel(R) Core(TM)2 Duo 1.66 GHz with 2 x 1GB of RAM and (iii) PC desktop Intel Pentium4 3.0 GHz with 512MB of RAM. The communication between machines was based on an Internet connection using a wireless network realized using a router ASUS 240 MIMO WL-566gM with a data transfer rate of 36.0Mbps.

After the creation and initialization of the *AS*, seller agents were created with the task to initiate the auctions. Buyers were then created and started almost simultaneously on 3 machines using an additional dummy agent. Its purpose was to send a message to all buyers created in the system to signal their starting. Upon receival of this message, buyers registered to auctions and started bidding. All buyers were using the same bidding strategy (see below).

Experimental data were collected during running of the experiments. For example, to evaluate the latency of the ser-

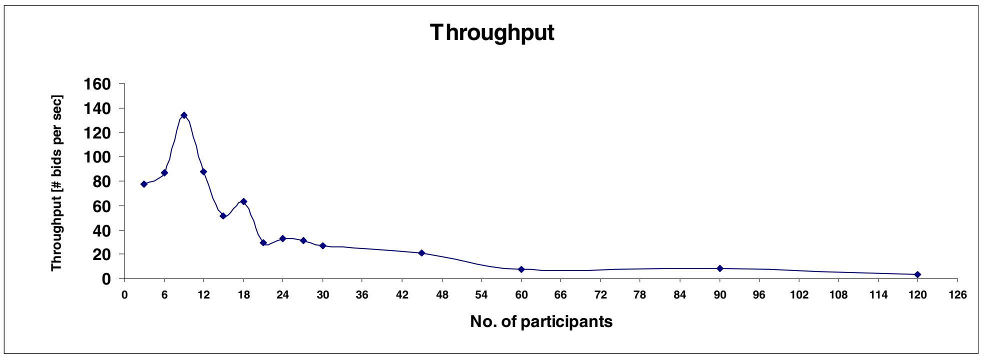

Figure 6. Throughput for 3 buyers per negotiation

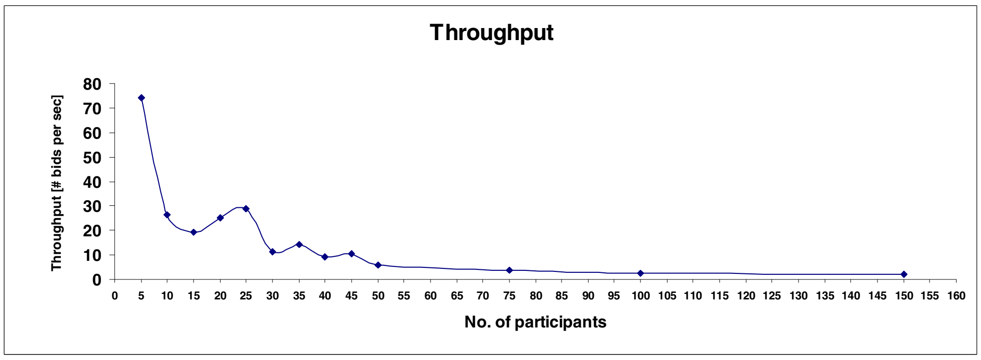

Figure 7. Throughput for 5 buyers per negotiation

vice, each buyer had to record both the time when a bid was submitted to the service and the time when the corresponding response was received from the service. At the end of an auction, each buyer averaged all the recorded elapsed times between submission of a bid and receival of a response and sent this result to a logger agent. The logger agent centralized the results obtained from all the buyers that participated to the auction and logged them onto a text file, so they could be further analyzed.

In our experiments we have varied the load in number of participants registered to an active auction. We carried out two sets of experiments: (i) 3 buyers were registered to each active auction, and (ii) 5 buyers were registered per auction. For each set the number of active auctions was varied from 1 to 40, and respectively from 1 to 50. So the number of participants was from 3 to 120 in the first case, and respectively from 5 to 150 in the second case.

We have chosen a simple strategy that actually simulates a sniping scenario: each participant agent submits a first bid immediately after it is granted admission to the auction and subsequently, whenever it gets a notification that another participant issued a bid that was accepted by the host. Note that under this sniping scenario the service is heavily overloaded – participants bid as fast as they can with the hope to overbid their opponents. The value of the bid is determined by incrementing the currently highest bid with a fixed bid increment that is part of each participant strategy. Additionally, each participant has its own valuation of the negotiated product. If the current value that the buyer decided to bid exceeds its private valuation then the proposal submission is canceled – product became "too expensive".

For each conducted experiment we measured: i) the average response time between submitting a bid and receiving a response, ii) the total number of submitted bids and iii) the total number of rejected bids. All the buyers had the same parameters: the same reservation price and the same bid increment. Because of this, the total number of bids accepted for an auction was constant and equal with the ratio between the reservation price and the bid increment (we assumed that the auction starting price was set to 0). In the experiments we have chosen a bid increment of 5 and a reserve price of 60.

Figures 2 and 3 illustrate the latency of the service as a function of number of participants. The graphics shows an increase of the latency with the number of participants. An explanation could be that, according to the generic negotiation protocol, whenever a bid is accepted by *AH*, all registered participants must be notified and thus the notification time will increase with the number of participants.

Figures 4 and 5 illustrate the total number of bids submitted as a function of number of participants. Note that because of the sniping scenario many bids were rejected. The difference between the total number of bids placed and the total number of bids rejected per auction is always the same, and equal in this case to 12 (60/5). In general, this difference will be a multiple of 12, more exactly, it will be equal to 12 multiplied by the number of active parallel auctions. This is the reason why the two charts show a linear growth of the total number of bids and the total number of rejected bids with the number of active buyers.

Figures 6 and 7 illustrate the service throughput measured under the sniping conditions as the average number of bids submitted and processed by an *AH* per unit of time. Note that the throughput decreases as the number of participants is increasing.

This experimental analysis shows that the service performance becomes worse as the total number of buyers and the total number of simultaneously active auctions are increasing. Moreover, we have noticed that situation is even worse when the number of participants registered per one auction is increasing. This can be noticed in our experiments by observing that the latency increases and throughput decreases as the number of buyers registered per auction is increasing. For example, for a total number of 45 simultaneously registered buyers, if the number of buyers per auction is 3 then the throughput was 20.85 bids/s and latency was 14.8 s, while if the number of buyers per auction is 5 then the throughput was 10.68 bids/s and latency was 29.3 s.

We plan to address these performance issues by improving the architecture of the auction service. One path of investigation could be the use of active networks, as suggested by the modeling and simulation results reported in [13].

4. Concluding Remarks

In this paper we have presented the main ideas concerning design and implementation of an agent-based auction service, together with its experimental evaluation on an English auction scenario. Currently we are in the process of extending the implementation across several directions: i) incorporating ontologies into our design. Ontologies will be used to add semantics to the negotiation process and also to describe the service at the semantic level; ii) implementation of other auction mechanisms. This will include the implementation of the corresponding auction rules and of additional participant agents capable of using these rules; iii) enhancing the internal representation of the service by improving the architecture, replacing *AD* with a software agent, rather than an internal data structure and incorporation of other rule engines – eg. Drools [3].

References

[1] JADE: Java Agent Development Framework. `http://jade.cselt.it`.

[2] JESS: Java Expert System Shell. `http://herzberg.ca.sandia.gov/jess/`.

[3] Drools: Business Rules Management System. `http://labs.jboss.com/drools/`.

[4] C. Bartolini, C. Preist, and N. R. Jennings. A software framework for automated negotiation. In R. Choren, A. F. Garcia, C. J. P. de Lucena, and A. B. Romanovsky, editors, *Software Engineering for Multi-Agent Systems III, Research Issues and Practical Applications [the book is a result of SELMAS 2004].*, volume 3390 of *Lecture Notes in Computer Science*, pages 213–235. Springer, 2005.

[5] M. Benyoucef and S. Rinderle. Modeling e-negotiation processes for a service oriented architecture. *Group Decision and Negotiation*, 15(5):449–467, 2006.

[6] M. B. Blake. Agent-oriented approaches to b2b interoperability. *Knowledge Engineering Review*, 16(4):383–388, 2001.

[7] C. Bornhövd, M. Cilia, C. Liebig, and A. Buchmann. An infrastructure for meta-auctions. In *WECWIS '00: Proceedings of the Second International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS 2000)*, page 21, Washington, DC, USA, 2000. IEEE Computer Society.

[8] A. Bădică and C. Bădică. Formalizing agent-based english auctions using finite state process algebra. *Journal of Universal Computer Science*, 14(7):1118–1135, 2008.

[9] C. Bădică, M. Ganzha, and M. Paprzycki. Implementing rule-based automated price negotiation in an agent system. *Journal of Universal Computer Science*, 13(2):244–266, 2007.

[10] C. Bădică, A. Giurca, and G. Wagner. Using rules and r2ml for modeling negotiation mechanisms in e-commerce agent systems. In *Proc. 2^{nd} Int. Conf. Trends in Enterprise Application Architecture, TEAA'2006*, 2007.

[11] A. Dobriceanu, L. Bîscu, and C. Bădică. Adding a declarative representation of negotiation mechanisms to an agent-based negotiation service. In *Web Intelligence/IAT Workshops*, pages 471–474. IEEE Computer Society, 2007.

[12] A. Dobriceanu, L. Bîscu, C. Bădică, and E. Popescu. Considerations on the design and implementation of an agent-based auction service. In C. Bădică and M. Paprzycki, editors, *Advances in Intelligent and Distributed Computing*, volume 78 of *Studies in Computational Intelligence*, pages 75–84, Berlin, Germany, 2008. Springer.

[13] J. Hillston and L. Kloul. Performance investigation of an on-line auction system. *Concurrency and Computation: Practice and Experience*, 13(1):23–41, 2001.

[14] D. Rolli, S. Luckner, H. Gimpel, and C. Weinhardt. A descriptive auction language. *Electronic Markets – The International Journal*, 16(1):51–62, 2005.