Developing a Model Agent-based E-commerce System

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

[1] University of Craiova, Software Engineering Department
Bvd. Decebal 107, Craiova, 200440, Romania
badica_costin@software.ucv.ro
[2] Elbląg University of Humanities and Economy,
ul. Lotnicza 2, 82-300 Elbląg, Poland
ganzha@euh-e.edu.pl
[3] Computer Science Institute, SWPS, 03-815 Warsaw, Poland
marcin.paprzycki@swps.edu.pl

Abstract. It is easy to realize that goals set behind a large class of *agent systems* match these put forward for systems defined as *e-service intelligence*. In this chapter we describe a model agent-based e-commerce system that utilizes rule-based approach for price negotiations. Furthermore, the proposed system attempts at mediating the apparent contradiction between agent mobility and intelligence.

1 Introduction and Overview

Recently an increasing interest in combining Internet-based electronic services (e-services) with "intelligent" functions can be observed (these new e-services are often called *e-service intelligence*). While this particular trend is relatively new, creation of intelligent distributed systems in form of software agents can be traced back a least to the seminal paper of P. Maes [29]. While her main concern was development of an infrastructure dealing with information overload, further research concerned applications of software agents in a number of areas including e-government, e-learning, e-shopping, e-marketing, e-banking, e-logistics etc. There, software agents are to facilitate much higher quality information, personalized recommendation, decision support, quasi-direct user participation in organizational planning, knowledge discovery etc. When developed and implemented, agent systems are to be adaptive, personalized, proactive and accessible from a broad variety of devices [40]. It is therefore easy to see how software agents, and agent systems in general, can be viewed as an incarnation of e-service intelligence.

While there exist a large number of attempts at developing agent-based systems, they are mostly small-scale demonstrator systems—later described in academic publications. Separately, some applications utilize the agent metaphor, but not existing agent tools and environments. Finally it is almost impossible to find out if actual agent systems exist in the industry; e.g. the true role of the Concordia agent system within the Mitsubishi Corp. While a number of possible reasons for this situation have been suggested (for instance see [31, 32]), one of them has been recently dispelled. It was shown that modern agent environments (e.g. JADE [21]), even when running on an antiquated

hardware, can scale to 2000 agents and 300,000 messages [14, 15]. Thus it was experimentally established that *it is possible to build and experiment with large-scale agent systems*. Therefore, it is extremely important to follow the positive program put forward by Nwana and Ndumu [31] and focus on developing and implementing such systems.

One of the well-known applications where software agents are to play an important role is e-commerce. Modern agent environments (such as JADE) can support implementation of quasi-realistic model e-commerce scenarios. Moreover, advances in auction theory have produced a general methodology for describing price negotiations. Combination of these factors gave new impetus to research on automating e-commerce. In this context, autonomous, and sometimes mobile, software agents are cited as a potentially fruitful way of approaching e-commerce automation [25].

Since *autonomy* is a broad concept that can be defined in many ways, we would like to narrow it down and focus on *adaptability* viewed as ability to update the negotiation "mechanism" to engage in unknown in advance forms of price negotiations. Obviously, another aspect of autonomy is *decision autonomy* that can be understood as capability to reason over past experiences and domain knowledge in order to maximize "utility" (making it very closely related to "intelligence").

Finally, the notion of agent *mobility* refers to its capacity to migrate from one computer to another. While the goal of such a migration is typically related to acting on behalf of some software or human entity, it does not depend on the intelligence that agents are possibly equipped with. However, to be able to facilitate e-service intelligence, we have to be able to combine the two—as mobile agents have to be able to dynamically adapt to situations found within visited sites. Therefore, agent mobility requires transfer of code, data, process and authority between machines. This makes intelligent mobile agents very heavy [40] and later in this chapter we discuss a partial solution of this problem.

In our work we have been developing a skeleton system in which autonomous agents interact in a way that models realistic scenarios arising in an e-marketplace (for a summary of our early results see [19] and references available there). Here, we have two long-term goals in mind. The first one is to broaden understanding of technical aspects of developing agent systems, such as agent functionalities, their interactions and communication, agent mobility etc. We are also concerned with the fact that without agents systems being actually implemented using tools that are apparently designed to do this, agent research will never be able to reach beyond academia. Success in achieving the first goal will allow utilization of our systems as a tool-box for modeling processes occurring in an e-marketplace. For instance, it will be possible to apply it to study: effects of pricing strategies, of negotiation protocols and strategies, flow of commodities etc. Due to agent flexibility it will be relatively easy to experiment with various e-commerce scenarios.

In this chapter we proceed as follows. In the next section we provide background information and follow with the description of our system formalized through a complete set of UML diagrams. We then discuss in some detail (including implementation specifics) how rule based engine can be used to facilitate autonomous price negotiations.

2 Background

2.1 Agent Systems in E-Commerce

While there exist many definitions of agents, for the purpose of this chapter we will conceptualize them as: encapsulated computer programs, situated in an environment, and capable of flexible, autonomous actions focused on meeting their design objectives [40]. For such agents, e-commerce is considered to be one of the paradigmatic application areas [25].

Proliferation of e-commerce is strongly related to the explosive growth of the Internet. For example, the total number of Internet hosts with domain names was estimated at 150 millions in 2003, while in the same year, Web content was estimated at 8000 millions of Web pages ([26]). At the same time, e-commerce revenue projections were estimated to reach in 2006 up to $0.3 trillions for B2C e-commerce and up to $5.4 trillions for B2B e-commerce ([26]).

E-commerce utilizes (to various degrees) digital technologies to mediate commercial transactions. As a part of our research we have conceptualized a commercial transaction as consisting of four phases:

- *pre-contractual phase* including activities like need identification, product brokering, merchant brokering, and matchmaking;
- *negotiation* where participants negotiate according to the rules of the market mechanism and using their private negotiation strategies;
- *contract execution* including activities like: order submission, logistics, and payment;
- *post-contractual phase* that includes activities like collecting managerial information and product or service evaluation.

While there exist many scenarios of applying agents in e-commerce, automated trading is one of the more promising ones. In particular, we are interested in using agents to support all the four phases of a commercial transaction, outlined above, by addressing questions like: how is an e-shop to negotiate price with selected, what happens before negotiations start and after they are finished, where from the purchase is actually made etc, thus going beyond the phase of negotiation itself.

Unfortunately, our research indicates that most existing automated trading systems are not yet ready to become the foundation of the next generation of e-commerce. For example, the Kasbah Trading System ([12]) supports buying and selling but does not include auctions; SILKROAD ([30]), FENAs ([23]) and Inter-Market ([24]) exist as “frameworks” but lack an implementation (which is typical for most agent systems in general [31]).

2.2 Automated and Agent-based Negotiations

In the context of this chapter we understand negotiations as a process by which agents come to a mutually acceptable agreement on a price ([28]). When designing systems for automated negotiations, we distinguish between *negotiation mechanisms* (*protocols*) and *negotiation strategies*. Protocol defines “rules of encounter” between negotiation

participants by specifying requirements that enable their interaction. The strategy defines the behavior of participants aiming at achieving a desired outcome. This behavior must be consistent with the negotiation protocol, and usually is specified to maximize "gains" of each individual participant.

Auctions are one of the most popular and well-understood forms of automated negotiations ([41]). An increased interest has been manifested recently in attempts to parameterize the auction design space with the goal of facilitating more flexible automated negotiations in multi-agent systems ([41, 28]). One of the first attempts for standardizing negotiation protocols was introduced by the Foundation for Intelligent Physical Agents—FIPA ([17]). FIPA defined a set of standard specifications of agent negotiation protocols including English and Dutch auctions. Authors of [9, 10] analyzed the existing approaches to formalizing negotiations (including FIPA protocols) and argued that they do not provide enough structure for the development of truly portable systems. Consequently, they outlined a complete framework comprising: (1) negotiation infrastructure, (2) a generic negotiation protocol and (3) taxonomy of declarative rules. The *negotiation infrastructure* defines roles of negotiation participants and of a host. Participants negotiate by exchanging proposals and, depending on the negotiations type, the host can also become a participant. The *generic negotiation protocol* defines the three phases of a negotiation: admission, exchange of proposals and formation of an agreement, in terms of how, when and what types of messages should be exchanged between the host and participants. *Negotiation rules* are used for enforcing the negotiation mechanism. Rules are organized into a taxonomy: rules for admission of participants to negotiations, rules for checking the validity of negotiation proposals, rules for protocol enforcement, rules for updating the negotiation status and informing participants, rules for agreement formation and rules for controlling the negotiation termination. Finally, they introduce a *negotiation template* that contains parameters that distinguish one form of negotiations from another, as well as specific values characterizing given negotiation. In this context it should be noted that rule-based approaches have been indicated as a very promising technique for introducing "intelligence" into negotiating agents ([9, 11, 16, 27, 37, 41, 42, 20]). Furthermore, proposals have been put forward to use rules for describing both negotiation mechanisms ([9, 38]) and strategies ([16, 37]).

With so much work already done in the area of agents and agent systems appearing in the context of autonomous price negotiations, let us underline what makes our approach unique.

- In most, if not all, papers only a "single price negotiation" is considered. Specifically, negotiations of a single item or a single collection of items is contemplated. Once such a negotiation is over, a group of agents (agent system) that participated in it completes its work. We are interested in a different (and a considerably more realistic) scenario when a number of products of a given type are placed for sale one after another. While this situation closely resembles what happens in any Internet store, it is practically omitted from research considerations. In this chapter, for clarity of enclosed UML diagrams, we depict situation where an almost unlimited number of items is to be sold. However, this assumption has only aesthetical reasons.
- Fact that multiple items are to be sold has also an important consequence for the way that price negotiations are organized. In the literature it is very often assumed

that agents join an ongoing negotiation process as soon as they are ready (see for instance [9]), while agent-actions that take place after price negotiation is completed are disregarded. Since we sell multiple items one after another, we have decided to treat price negotiations as a "discrete process." Here, except of a specific case of fixed price mechanism, buyer agents are "collected" and released in a group to participate in a given price negotiation. While the negotiation takes place buyer agents communicate only with the seller agent (they can be envisioned as being placed in a closed negotiation room). At the same time the next group of buyer agents is collected (as they arrive) and will participate in the next negotiation.

- Fact that multiple subsequent auctions (involving the same product) take place allows us to go beyond one more popular "limitation" of known to us agent systems. While sometimes they involve rather complicated price negotiations, e.g. mixed auctions (see for instance [35, 36]), since only a single item or a single collection of items are sold, it is only that given price negotiation mechanism that is taken into account. In our case, since multiple negotiations are used to sell items of the same product we conceptualize situation in which price negotiation mechanism change. For instance, first 25 items may be sold using English Auction, while the next 37 using fixed price with a deep discount.
- Furthermore, we consider the complete e-commerce system, which means that after negotiation is completed we conceptualize subsequent actions that may, or may not result in an actual purchase. In the case when purchase does not take place, we specify what should happen then to all involved agents.
- While agent mobility is often considered to be important in the context of e-commerce, conflict between agent mobility and intelligence is rarely recognized. In our work we address this question by designing modular agents and clearly delineating which modules have to be send, when, by whom and where.
- Finally, the complete system is being implemented using JADE; an actual agent environment.

3 Code Mobility in an Agent-Based E-Commerce System

Code mobility has been recognized as one of key enablers of large scale distributed applications, while its specific technologies, design patterns and applications have been systematically analyzed ([18]). Furthermore, recent research results suggest that blending mobility and intelligence might have important benefits especially in advanced e-commerce by providing application components with automated decision-making capabilities and ubiquity as required in networked environments ([25]). At the same time it has been argued that, as a general feature, agent mobility is unnecessary. Therefore, we asked a basic question: why, in the case of e-commerce, should one use mobile agents instead of messaging? To answer it, let us consider someone who, behind a slow Internet connection (which is not an uncommon situation), tries to participate in an eBay auction. In this case it is almost impossible to assure that this persons bid (1) reaches eBay server in time, (2) is sufficiently large to outbid opponents that have been bidding simultaneously (information about auction progress as well as responses may not be able to reach their destinations sufficiently fast). As a result, network-caused delays

may prevent purchase of the desired product. Obviously, this would not be the case if an autonomous agent representing that user was co-located with the negotiation host. In this context, one can obviously ask about the price of moving buyer agents across the network. Naturally, it may happen that an agent may not be able to participate in an auction because it does not reach the host in time. In response let us observe that: (1) if it is a particular single auction that the user is interested in, then agent not reaching the host has exactly the same effect as not being able to win because of bid(s) being late and/or too small; (2) therefore, it is only an agent that reaches the host in time that gives its user any chance to effectively participate in price negotiations; (3) furthermore, if an agent reaches its destination, it will be able to effectively participate in all subsequent negotiations within that host (and we assume across this paper that multiple negotiations involving items of the same product take place), while delays caused by network traffic may permanently prevent user from effective participation in any of them. For an extended discussion of the need for agent mobility in e-commerce see [7].

Let us now sketch proposed resolution of an above mentioned obvious contradiction between agent mobility and adaptivity. In our work, we utilize the negotiation framework introduced in [9, 10], where the *negotiation protocol* is a generic set of rules that describes all negotiations, while the *negotiation template* is a set of parameters that establishes the form of negotiation and its details. Finally, there is the *negotiation strategy* defining outcome optimizing actions of individual negotiation participants. It should be obvious that the *negotiation protocol* is generic and public—all agents participating in all negotiations have to use it. Therefore buyer agent can receive it upon its arrival at the host; similarly to the negotiation template which has to be "local" as it describes currently used form of negotiations (and which can change over time). It is only the strategy that is "private" and has to be obtained from the client agent (we name *client agent* agents representing User-Clients). At the same time, it has to be assumed that depending on the form of negotiation, different strategies will be used, and thus strategy is not known in advance. Therefore, since the protocol and the template can be obtained within the e-store, carrying them across the network is unnecessary. Unfortunately, it is not possible to establish the negotiation form in advance and send buyers with the negotiation strategy pre-loaded. Recall that in our system we assume that e-stores respond to the flow of commodities by actively changing forms of price negotiations. This being the case, by the time the buyer agent reaches its destination its strategy module may be useless, as the form of negotiations has already changed. We thus propose two network-traffic minimizing approaches to agent mobility. In the first case (named thereafter *agent mobility*) only an agent skeleton is send across the network and upon arrival it obtains the negotiation protocol and the template and then requests the strategy module from the client agent. In the second case (named thereafter *code mobility*) buyer agents are created by the host (on the basis of a request from the client agent) and assembled including (1) protocol, (2) actual template, and (3) information who they represent. Then, again, they request an appropriate strategy module from their client agent. Observe, that since only the strategy module is "secret," while the remaining parts of the buyer are public and open to any scrutiny at any time, this latter solution should not directly result in an increased security risk.

4 Description of the System

4.1 Conceptual Architecture

In our description of the system we utilize its almost complete UML-based formalization. Due to lack of space we have decided to present a *complete* set of UML diagrams of the system, rather than lengthy descriptions of its features and underlying assumptions. Interested readers should consult ([6, 7, 19]) for more details. In Figure 1 we present the use case diagram of our system that depicts all of its agents and their interactions. We can distinguish three major parts of the system: (1) the *information center* where white-page and yellow-page type data is stored—this is our current solution of the matchmaking problem [38], (2) the *purchasing side* where agents and activities representing User-Client reside, and (3) the *seller side* where the same is depicted for the User-Seller. Let us now describe in detail each of the agents (except the CIC agent that plays only an auxiliary role [19]) found in Figure 1.

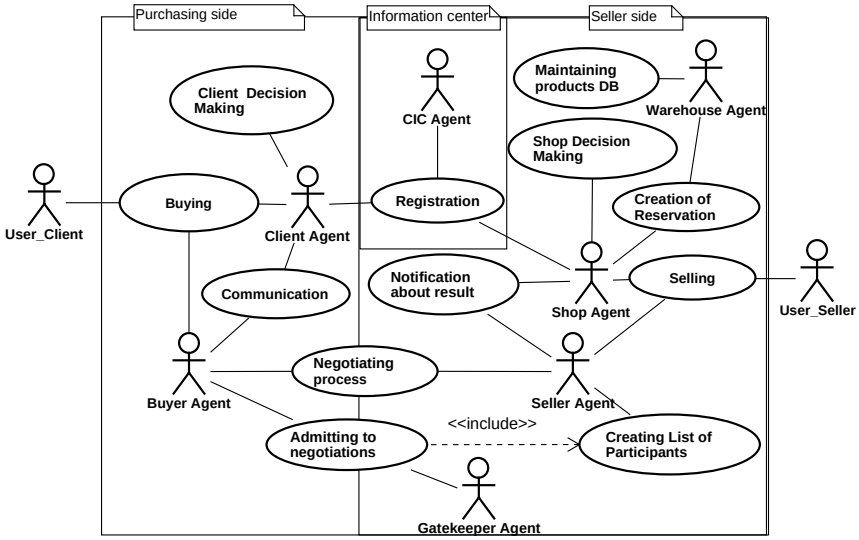

Fig. 1. Use case diagram

4.2 UML Models of Agents in the System

Client agent On the purchasing side, we have two agents. The *Client* agent, represented in Figures 2 and 3 exists in a complex state. On the one hand it listens for orders from the User-Client and, to fulfill them: (1) queries the *CIC* agent which has access to information which stores sell the requested product and if they create *Buyer* agents locally (or if such agent has to be send to them), (2) then it dispatches or requests creation of *Buyer* agents to / by each such e-store (identified by its *Gatekeeper* agent). At the

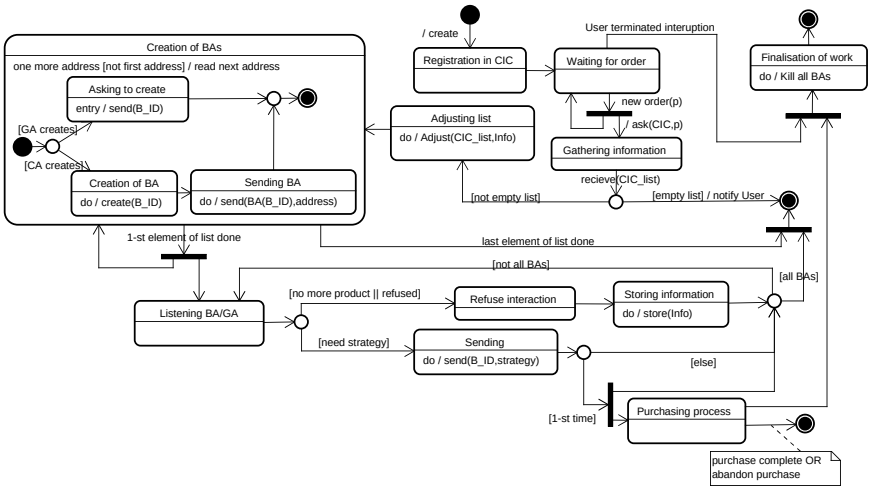

Fig. 2. Client Agent Statechart diagram

same time, it directly manages the process of making purchases on behalf of the User-Client (Figure 3), on the basis of *Buyer* agent messages informing about results of price negotiations (let us note that in the case of multiple orders separate groups of *Buyer* agents—corresponding to separate products—will be managed in the same fashion). For a certain amount of time the *Client* collects reports send by *Buyer* agents. When the wait-time is over (or when all *Buyer* agents have reported back), *Client* agent enters a complex state. On the one hand it continues listening for messages from *Buyer* agents (obviously, if all have reported already then none will be coming). On the other hand it goes through a multi-criteria decision making procedure (the "MCDM" box) that has one of three possible outcomes: (i) to attempt at completing a selected purchase, (ii) to await better opportunity, or (iii) to declare the purchase impossible and notify the User-Client accordingly. Note that, in a realistic system, the *MCDM* analysis should be truly multi-criteria and include factors such as: price, history of dealing with a given e-shop, delivery conditions etc.

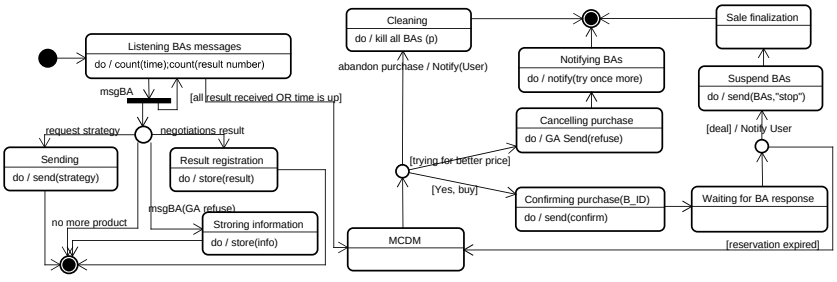

Fig. 3. Client Agent Statechart diagram

When attempt at completing a purchase is successful, the *Client* agent sends messages to all *Buyer* agents to cease to exist. The situation is slightly more complicated when the attempt was unsuccessful. Note that it is quite possible that the first *MCDM* analysis was undertaken before all *Buyer* agents have complete their "first round" of price negotiations. They could have contacted the *Client* while it was "thinking" which of the existing offers to choose. Therefore, when the *Client* agent analyses available reservations, they include not only reservations that have been already considered, but also possibly new ones that have arrived in the meantime. As a result another attempt at making a purchase can be made. If none of the available offers is acceptable, but purchase was not declared impossible, the *Client* agent undertakes the following actions: (1) informs all *Buyer* agents that have already reported to cancel current reservations and return to price negotiations (or just to return to price negotiations if they previously failed) and (2) resets timer establishing how long it will wait before the next *MCDM* analysis. Observe that in this way, in the proposed system it is possible that some agents make their second attempt at negotiating prices, while some agents have just finished the first. As this process continues in an asynchronous fashion *Buyer* agents will make different number of attempts at negotiating price that is acceptable to the *Client* agent. This process and the *Client* agent will terminate when all orders submitted by the customer have been either honored or abandoned. For the time being we assume that the "Sale finalization" process seen in Figure 3 is always successful. In the future we plan to remove this somewhat artificial restriction.

Let us note that it is possible that since it is the *Client* agent that makes the final determination which offer to accept, and that it has to communicate with one of its remotely located *Buyer* agents to actually complete a purchase, the request to attempt at making that purchase could be network-delayed resulting in an expired reservation and inability to complete the task. Unfortunately, this problem does not seem to have a simple solution, since price comparison requires communication between agents participating in price negotiations. In our system we have selected a central point—*Client* agent—that will collect all offers, instead of all-to-all communication. Since not all sites will conduct their price negotiations at the same time, and with the same urgency, it is impossible to assure that the best offer will still be available, when the "remaining" agents complete their negotiations. Therefore, our solution remains optimal in terms of reducing total network congestion by sending only minimal-size agents and minimizing the total number of messages send over the network.

Finally, let us note that a complete information about all events taking place during servicing User-Client request (such as: refusal to admit to negotiations, results of price negotiations, length of reservation, etc.) is stored for further information extraction. For instance, as a result of data analysis, store that is constantly selling good at very high prices may be simply avoided.

Buyer agent *Buyer* agent (see Figure 4) is the only agent in the system that involves mobility. It is either dispatched by the *Client* agent or created by the *Gatekeeper* agent on request of the *Client*. If it is dispatched, then upon arrival at the store it communicates with the *Gatekeeper* (see Figure 6,7) to obtain entry to negotiations (in case when entry is not granted it informs its *Client* agent and is killed). In the case of *Client* re-

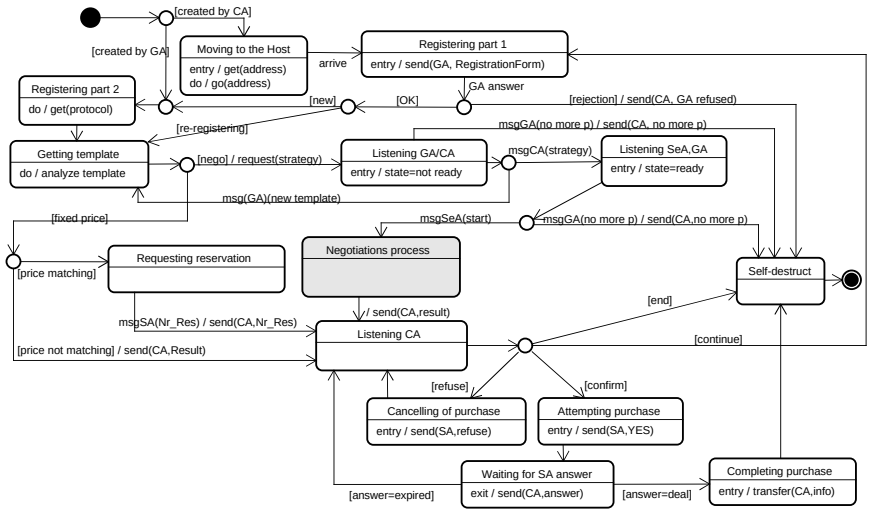

Fig. 4. Buyer Agent Statechart diagram

questing creation of the *Buyer*, the *Gatkeeper* may deny such request. In both cases, information about refusal to cooperate is stored by the *Client* agent (e.g. e-stores that do not want to cooperate may be removed from the list obtained from the *CIC*; see box "Adjusting list" in Figure 2). Next, the preregistered *Buyer* obtains from the *Gatekeeper* the negotiation protocol and the current negotiation template and requests (and obtains) an appropriate strategy module from the *Client* agent (see Figure 2). When all these modules are installed *Buyer* informs the *Gatekeeper* that it is ready to negotiate (it is then registered as such). Price negotiations start when the *Buyer* receives a start message from the *Seller* agent (see Figure 9; note that the "Negotiations box" appearing there is "the same" for both the *Seller* and the *Buyer* agents); note also special treatment of fixed-price negotiations by both the *Buyer* and the *Gatekeeper* agents. Upon completion of negotiations, *Buyer* informs the *Client* about their result and, if necessary (when an attempt at completing purchase is made), acts as an intermediary between the *Client* and the *Shop* agents. In the case when purchase was not attempted or was not successful, *Buyer* agent awaits the decision of the *Client* and if requested proceeds back to participate in price negotiations (before doing so it requests permission to re-enter that may be granted or denied; updates its negotiation template and possibly the strategy module—if the template has changed). This process continues until the *Buyer* agent self-destructs on the request of the *Client* agent.

Shop agent On the "selling side" of the system, the *Shop* agent acts as the representative of the User-Seller. We assume that after it is created, it persistently exists in the system until the user decides that it is no longer needed. The UML diagram representing the *Shop* agent is presented in Figure 5. Upon its instantiation, the *Shop* agent creates and initializes its co-workers: a *Gatekeeper* agent, a *Warehouse* agent and *Seller* agents (one for each product sold). Initialization of the *Warehouse* agent involves pass-

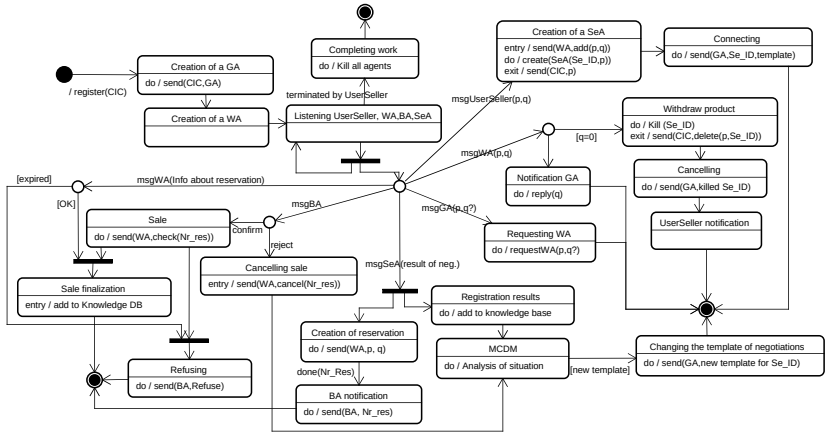

Fig. 5. Shop Agent Statechart diagram

ing information about goods that are initially available for sale (see Figure 8), while initialization of the *Gatekeeper* agent involves providing it with templates that are to be used initially in price negotiations of each product sold. Furthermore, the *Gatekeeper* agent and the list of products available in the store are registered with the *CIC* agent.

After the initialization stage, the *Shop* agent enters a complex state where it supervises negotiations and product flow. First, it waits for finish of any price negotiation. If the negotiation was successful, a given *Seller* informs the *Shop* agent, which is asking the *Warehouse* agent to reserve a given quantity of a particular product for a specific amount of time. (Currently we assume that a single item of a given product is sold each time, but this, somewhat limiting, assumption will be removed in the future.) Events can then proceed according to following scenarios.

1. If the winning *Buyer* confirms purchase then the *Shop* asks the *Warehouse* agent to check the reservation.
   - If the reservation did not expire then the *Shop* informs the *Buyer* agent about acceptance of transaction. This event starts the final stage, named ”Sale finalization” which includes such actions as payment and delivery.
   - In opposite case, the *Shop* agent sends rejection to the *Buyer* agent
2. If the *Client* agent rejects purchase (and informs the *Shop* agent about it through the *Buyer* agent) then the *Shop* agent asks the *Warehouse* agent to cancel the reservation.

Completing one of these scenarios ”closes” this branch of *Shop* agent execution. Separately, the *Shop* agent keeps track of all negotiations and transactions and periodically performs multi-criteria analysis (the *MCDM* module) that may result in changes in the negotiation template for one or more products (e.g. minimal price, type of price negotiation mechanism, etc.). For instance, when only a few items are left they may be deeply-discounted, or put on sale through an auction. In this case a new template is generated and send to the *Gatekeeper* agent that switches it in an appropriate moment (see below, Figures 6, 7).

Let us also note that, similarly to the *Client* agent, the *Shop* agent stores complete information about all events taking place in the e-store (such as: results of price negotiation, information about agents that actually purchased reserved product, information of agents that canceled reservations, etc.). This information, when analyzed, may result for instance in a given *Client* agent being barred from entering negotiations.

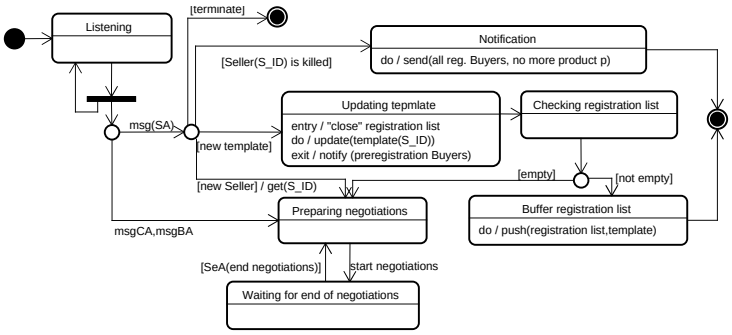

Fig. 6. Gatekeeper Agent Statechart diagram

Gatekeeper agent *Shop* agents cooperate directly with their *Gatekeeper* agents that (1) either interact with incoming *Buyer* agents, and admit them to the negotiations (or reject their attempt at entering the host), or interact with *Client* agents and, on their request, create *Buyer* agents (or reject such a request), and provide admitted / created *Buyer* agents with the protocol and the current negotiation template (2) in appropriate moments release *Buyer* agents to appropriate *Seller*s and (3) manage updates of template modules. The statechart diagram of the *Gatekeeper* agent is presented in Figure 6 (the top level description of *Gatekeeper* functionality) and continued in Figure 7 (depicting negotiation related activities). Each created or allowed to enter *Buyer* agent is put on a list of preregistered agents and provided with protocol and current template. *Buyer* agents remain on that list until they receive their strategy module and complete self-assembling. Assembled *Buyer* agents are put on a list of registered agents that await start of price negotiations. When a minimum number of *Buyer* agents have registered (minimum for a given form of negotiations) and the wait-time has passed, the *Gatekeeper* passes their identifiers and the current negotiation template to the *Seller* agent. Then it cleans the current list of registered *Buyer* agents and the admission/monitoring process is restarted (assuming that the *Seller* agent is still alive).

As stated above, our system allows *Buyer* agents that lost negotiations or that decided not to make a purchase to stay at the host and try to re-enter negotiations. They have to ask permission to be re-admitted and if allowed back they receive an updated template (”old Buyer” path). When a new template module is delivered by the *Shop* agent, a list of currently registered *Buyer* agents is put into a buffer (”Buffer registration list” box). These agents have to be serviced first, using the current template that they have been provided with upon entering the e-store. At the same time the new in-

coming agents will then be given the new template. Finally, in a special case, when a given product has been sold-off and the *Shop* agent terminates the *Seller* responsible for selling it, the *Gatekeeper* informs awaiting *Buyer* agents about this fact.

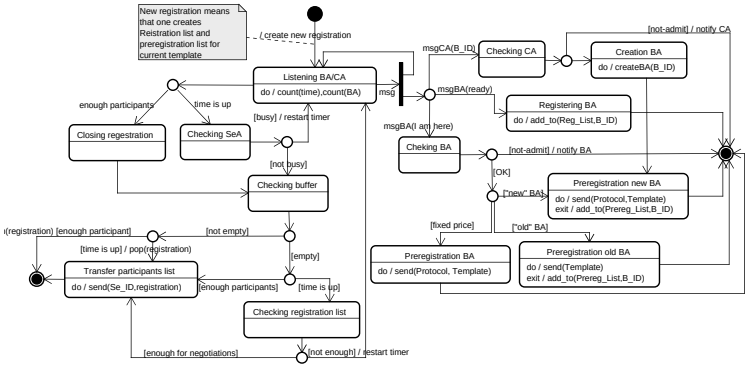

Fig. 7. Statechart diagram for Preparing Negotiations State

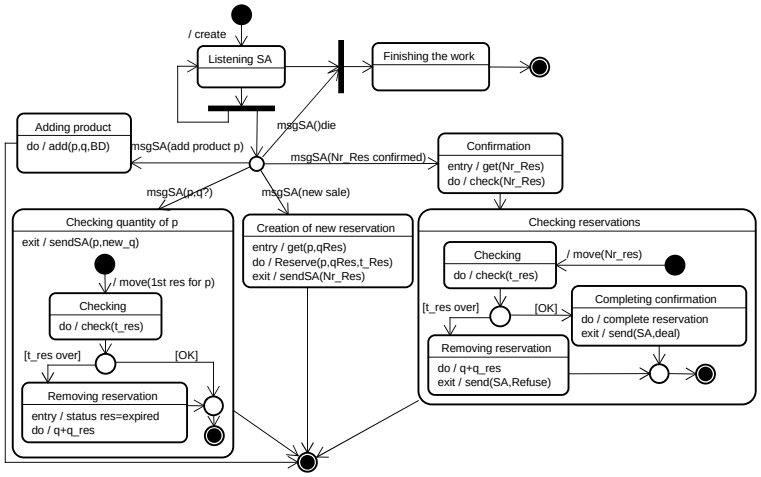

Fig. 8. Warehouse Agent Statechart diagram

Warehouse agent *Shop* agent interacts also directly with the *Warehouse* agent (presented in Figure 8). In the early stages of its functioning the *Warehouse* agent is supplied (by the *Shop* agent) with information about products and their quantities (to be saved in a database). Then it enters a complex state where it (a) awaits notifications from the *Shop* agent and (b) acts on them. The *Shop* agent notifies the *Warehouse* agent

about: (i) registration of new products for sale, (ii) product reservations, (iii) purchase confirmations, and (iv) purchase terminations. Each of these notifications is followed by an appropriate response: (i) product registration, (ii) product reservation, (iii) checking status of a reservation, (iv) cancellation of a reservation. Finally, if quantity of some product becomes 0, the *Warehouse* agent informs about it the *Shop* agent, which (in the current state of our system) terminates the corresponding *Seller* agent, and informs about it both the *CIC* and the *Gatekeeper* agents.

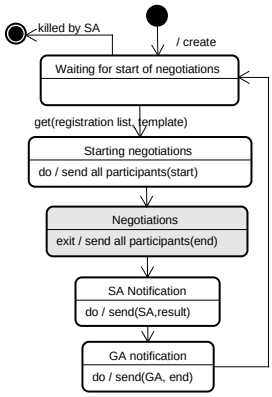

Fig. 9. Seller Agent Statechart diagram

Seller agent Finally, the last agent working on the "selling side" of the system is the *Seller* agent. It is characterized by a rather simple statechart diagram (see Figure 9). The simplicity comes form the fact that, in the "Negotiations box," the *complete* negotiation framework proposed in [9, 10] in enclosed. Observe that not all negotiations have to end in finding a winner and our system is able to handle such an event. At the same time, all data about negotiations is collected and analyzed by the *Shop* agent and, for instance, a sequence of failures could result in a change of the negotiation template.

System activity diagram Let us now combine activities of all agents in the system into one diagram (see Figure 10, 11). This diagram represents flow of actions presented from the perspective of the two main agents in the system: the *Shop* and the *Client*. Obviously, to keep that diagram readable, we had to omit large number of details that have been represented within statechart diagrams of individual agents that should be "co-viewed" with the activity diagram.

4.3 Rule-Based Mechanism Representation

Let us now describe how we have implemented in our system rule-based mechanisms. We start by summarizing the framework for automated negotiation introduced in [9, 10]

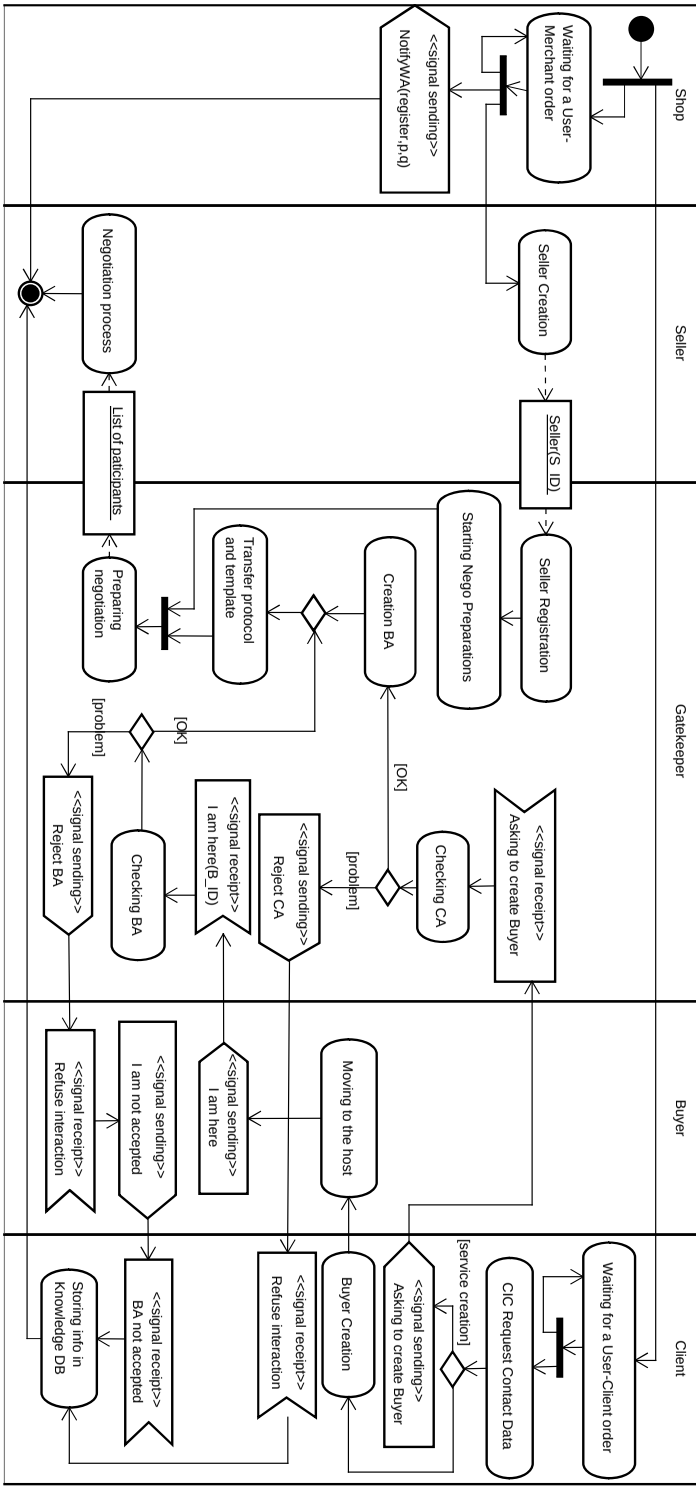

Fig. 10. Activity Diagram—before negotiation process

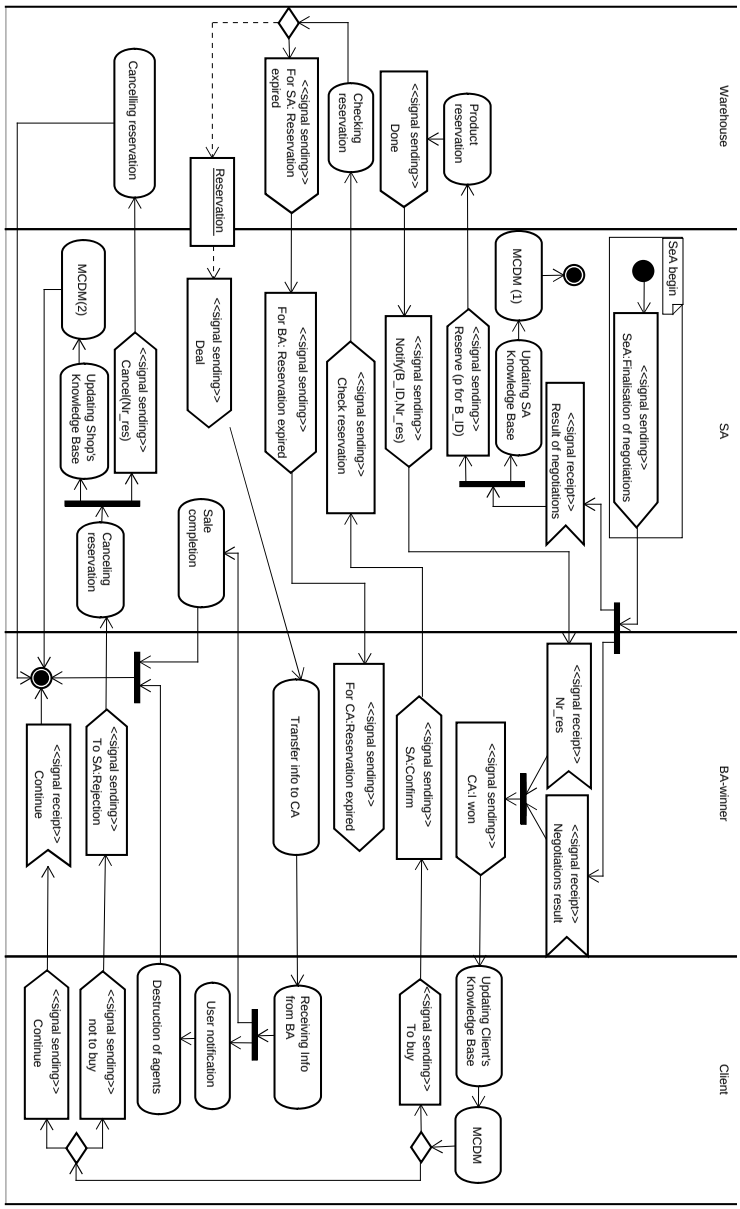

Fig. 11. Activity Diagram—after negotiation process

which is based on an abstract negotiation process that comprises: a negotiation infrastructure, a generic negotiation protocol and a taxonomy of declarative rules. Here, the *negotiation infrastructure* defines roles involved in the negotiation process: participants and a host. Participants negotiate by exchanging proposals within a negotiation locale that is managed by the negotiation host. Depending on the type of negotiations, the host can also play the role of a participant (for example in an iterative bargaining scenario). The *generic negotiation protocol* defines, in terms of how and when messages should be exchanged between the host and negotiation participants, the three main phases of negotiations: (1) admission, (2) exchange of proposals and (3) formation of an agreement. *Negotiation rules* are needed for enforcing a specific negotiation mechanism. Rules are organized into a taxonomy that contains the following categories: (a) rules for participants admission to negotiations, (b) rules for checking the validity of negotiation proposals, (c) rules for protocol enforcement, (d) rules for updating the negotiation status and informing participants, (e) rules for agreement formation and (f) rules for controlling the negotiation termination. Based on the categories of rules identified as necessary to facilitate negotiations, in [9, 10] it is suggested to partition the negotiation host into a number of corresponding components: *Gatekeeper, Proposal Validator*, *Protocol Enforcer*, *Information Updater*, *Negotiation Terminator* and *Agreement Maker* (that are called sub-agents). Each component is responsible for enforcing a specific category of rules. Host components interact with each-other via a blackboard and with negotiation participants by direct messaging. Note that these components are conceptualized as a part of the host (sub-agents), not as stand-alone agents. This fact will have consequences as to how they are to be implemented.

Before proceeding let us note that we have modified the proposed framework and upgraded the *Gatekeeper* to become a full-fledged agent [19]. In its new role, the *Gatekeeper* agent has also an increased scope of responsibilities (described above). Let us now show: (i) how the negotiation host agent (*Seller*) is structured into components (sub-agents); (ii) how rules are executed by the negotiation host in response to various messages received from negotiation participants and how rule firing control is switched between various components of the negotiation host, and (iii) how the generic negotiation protocol was implemented using JADE agent behaviors and ACL message exchanges between host and participants.

The Negotiation Host — *Seller* agent Let us note that what was defined in [9, 10] as negotiation *host* became a *Seller* agent in our system. We will thus use these two terms interchangingly. Host and negotiation participant agents are implemented as ordinary JADE agents and thus they extend the *jade.core.Agent* class. The *Seller* agent encapsulates the negotiation controlling sub-agents that are implemented as ordinary Java classes (see Figure 12): *Proposal Validator*, *Protocol Enforcer*, *Information Updater*, *Negotiation Terminator* and *Agreement Maker*. Each host component defines a *handle()* method that is activated whenever the component must react to check the category of rules it is responsible for. Note, again, that these components are not full-blown JADE agents, but ordinary member objects within the *Seller* agent.

In addition to sub-agents responsible for protocol enforcement, the host encapsulates two member objects representing the negotiation locale and the blackboard (see

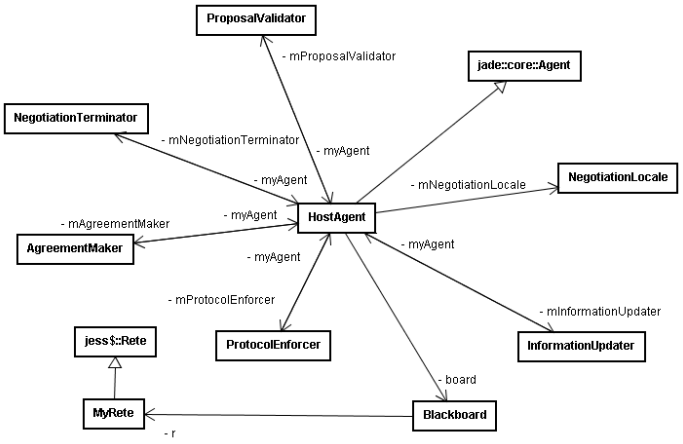

Fig. 12. The class diagram showing the structure of the *Seller* agent

Figure 12): *Negotiation Locale* and *Blackboard* "boxes". The *Negotiation Locale* object stores the *negotiation template* (a structure that defines negotiation parameters; see [9]) and the list of participants that were admitted to a given negotiation (obtained from the *Gatekeeper* agent—see above). The *Blackboard* object is a JESS rule engine (class *jess.Rete*) that is initialized with negotiation rules. Whenever the category of negotiation rules is checked, the rule engine is activated.

The negotiation host contains handler methods that are activated by *action*() methods of agent behaviors. Each handler method delegates the call to the responsible component. Finally, that component activates the rule engine via the *myAgent* member object that points to the parent host agent (see Figure 12).

Controlling Rule Execution Rather then implementing each component of the negotiation host as a separate rule engine, we are using a single JESS rule engine that is shared by all host components. This rule engine is implemented using class *jess.Rete*. The advantage is that we now have a single rule engine per negotiation host rather than 6 engines as suggested in [9]. Furthermore, this means that in the case of m products sold, we will utilize m instances of the JESS rule engine, instead of $6m$ instances necessary in [9, 10].

Rules and facts managed by the rule engine are partitioned into JESS modules. Currently we are using one JESS module for storing the blackboard facts and a separate JESS module for storing rules used by each component. Blackboard facts are instances of JESS *deftemplate* statements and they can represent: (1) the negotiation template; (2) the active proposal that was validated by the *Proposal Validator* and the *Proposal Enforcer* components; (3) a withdrawn proposal; (4) seller reservation price (not visible to participants); (5) negotiation participants; (6) the negotiation agreement that is eventually generated at the end of a negotiation; (7) the information digest that is visible to the negotiation participants; (8) the maximum time interval for submitting a new bid before

the negotiation is declared complete; or (9) the value of the current highest bid. Note that these facts have been currently adapted to represent English auctions (and will be appropriately modified to represent other price negotiation mechanisms).

Each category of rules for mechanism enforcement is stored in a separate JESS module. This module is controlled by the corresponding component of the *Seller* agent. Whenever the component handles a message it activates the rules for enforcing the negotiation mechanism. Taking into account that all rules pertinent to a given host are stored internally in a single JESS rule-base (attached to a single JESS rule engine), the JESS *focus* statement is used to control the firing of rules located only in the focus module. This way, the JESS facility for partitioning the rule-base into disjoint JESS modules proves very useful to efficiently control the separate activation of each category of rules. Note also that JADE behaviors are scheduled for execution in a non-preemptive way and this implies that firings of rule categories are correctly serialized and thus they do not cause any synchronization problems. This fact also supports our decision to utilize a single rule engine for each host.

Generic Negotiation Protocol and Agent Behaviors The *generic negotiation protocol* specifies a minimal set of constraints on sequences of messages exchanged between the host and participants. As specified in [9], the negotiation process has three phases: (1) admission, (2) proposal submission and (3) agreement formation. The admission phase has been removed from the negotiation process described in [9], but it was implemented in exactly the same way as suggested there. For instance, in the case of *agent mobility* it starts when a new participant (*Buyer* agent) requires admission to the negotiation, by sending an ACL PROPOSE message to the *Gatekeeper* agent. The *Gatekeeper* grants (or not) the admission of the participant to the negotiation and responds accordingly with either an ACL ACCEPT-PROPOSAL or an ACL REJECT-PROPOSAL message (currently admission is granted by default). In the way that the system is currently implemented, the PROPOSE message is sent by the participant agent immediately after its initialization stage, just before its *setup()* method returns. The task of receiving the admission proposal and issuing an appropriate response is implemented as a separate behavior of the negotiation host.

When a *Buyer* agent is accepted to the negotiation, it also receives from the host the negotiation protocol and template (representing parameters of auctions: auction type, auctioned product, minimum bid increment, termination time window, currently highest bid). *Buyer* will enter the phase of submitting proposals after it was dispatched to the negotiation (here, a number of *Buyer* agents that were granted admission is ”simultaneously” released by the *Seller* (that sends them a start message) and they—possibly immediately—start submitting bids according to their strategies [19]). The generic negotiation protocol states also that a participant will be notified by the negotiation host if its proposal was either accepted (with an ACL ACCEPT-PROPOSAL) or rejected (with an ACL REJECT-PROPOSAL). In the case when a proposal was accepted, the protocol requires that the remaining participants will be notified accordingly with ACL INFORM messages.

Strategies of participant agents must be defined in accordance with the constraints stated by the *generic negotiation protocol*. Basically, the strategy defines when a ne-

gotiation participant will submit a proposal and what are the values of the proposal parameters. In our system, for the time being, we opted for an extremely simple solution: the participant will submit a first bid immediately after it was released to the negotiation and subsequently, whenever it gets a notification that another participant issued a proposal that was accepted by the host. The value of the bid is equal to the sum of the currently highest bid and an increment value that is private to the participant. Additionally, each participant has its own valuation of the negotiated product in terms of a reservation price. If the value of the new bid exceeds this reservation price then the proposal submission is canceled. The implementation of the participant agent defines two JADE agent behaviors for dealing with situations stated above. Obviously, as the system matures, we plan to develop, implement and experiment with a number of negotiation strategies that can be found in the literature (e.g. see [20]).

Finally, the agreement formation phase can be triggered at any time. When the agreement formation rules signal that an agreement was reached, the protocol states that all participants involved in the agreement will be notified by the host with ACL INFORM messages. The agreement formation check is implemented as a timer task (class *java.util.TimerTask*) that is executed in the background thread of a *java.util.Timer* object.

5 Consluding Remarks

In this chapter we have described an agent-based model e-commerce system that is currently being developed and implemented in our group. This system as it is being extended is slowly converging toward the main ideas underlying e-service intelligence systems. After presenting background information on software agents and automatic negotiations we have provided a description of the system, illustrated by its complete formal UML-based definition. We have also argued that the proposed solution is able to mediate the existing contradiction between agent mobility and intelligence, by precisely delineating which components, and when, have to be pushed across the network. Furthermore, we have discussed in detail how the negotiation framework, utilizing a rule-based engine is implemented in the system.

Currently, the proposed system is systematically being implemented and extended. We have experimented with its earlier versions and were able to see that it scales well (on a network consisting of 22 computers). Furthermore, we were able to successfully run it in a heterogeneous environment consisting of Windows and Linux workstations. The results have been reported in [4]. More recently we have implemented and successfully experimented with the above described rule-based engine applied to the English auction mechanism. Additional information can be found in [1, 2].

As the next steps we envision, among others: (1) completion of integration of the original system skeleton with the rule-based engine, (2) addition of rules for a number of additional price negotiation protocols (e.g. Vickery auction, Dutch auction etc.), (3) implementation of an initial set of non-trivial negotiation strategies for both buyers and sellers, (4) conceptualization of MCDM processes, starting from the ways in which data concerning results of price negotiations has to be stored so that it can be effectively

utilized in support of decision making in the system etc. We will report on the results in subsequent publications.

References

1. Bădică, C., Ganzha, M., Paprzycki M.: Rule-Based Automated Price Negotiation: an Overview and an Experiment. In: *Proceedings of the International Conference on Artificial Intelligence and Sost Computing, ICAISC'2006*, Zakopane, Poland. Springer LNAI, 2006 (in press)
2. Bădică, C., Bădiţă, A., Ganzha, M., Iordache, A., Paprzycki M.: Implementing Rule-based Mechanisms for Agent-based Price Negotiations. In: *Proceedings of the ACM Symposyum on Applied Computing, SAC'2006*, Dijon, France. ACM Press, 2006 (in press)
3. Bădică, C., Ganzha, M., Paprzycki, M., Pîrvănescu, A.: Combining Rule-Based and Plug-in Components in Agents for Flexible Dynamic Negotiations. In: M. Pěchouček, P. Petta, and L.Z. Varga (Eds.): *Proceedings of CEEMAS'05*, Budapest, Hungary. LNAI 3690, Springer-Verlag, pp.555-558, 2005.
4. Bădică, C., Ganzha, M., Paprzycki, M., Pîrvănescu, A.: Experimenting With a Multi-Agent E-Commerce Environment. In: V. Malyshkin (Ed.): *Proceedings of PaCT'2005*, Krasnoyarsk, Russia. LNCS 3606, Springer-Verlag, pp.393-402, 2005.
5. Bădică, C., Ganzha, M., Paprzycki, M.: Mobile Agents in a Multi-Agent E-Commerce System. In: *Proceedings 7th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC'05*, Timişoara, Romania. IEEE Computer Society Press, Los Alamitos, CA, pp.207-214, 2005.
6. Bădică, C., Ganzha, M., Paprzycki, M.: UML Models of Agents in a Multi-Agent E-Commerce System. In: *Proceedings of the IEEE Conference of E-Business Engineering, ICEBE 2005*, Beijing, China. IEEE Computer Society Press, Los Alamitos, CA, pp.56-61, 2005.
7. Bădică, C., Ganzha, M., Paprzycki, M.: Two Approaches to Code Mobility in an Agent-based E-commerce System. In: C. Ardil (ed.): *Enformatika*, Volume 7, pp.101-107, 2005.
8. Bădică, C., Bădiţă, A., Ganzha, M., Iordache, A., Parzycki, M.: Rule-Based Framework for Automated Negotiation: Initial Implementation. In: *Proceedings 1st Conference on Rules and Rule Markup Languages for the Semantic Web, RuleML'2005*, Galway, Ireland. Lecture Notes in Computer Science 3791, Springer-Verlag, pp.193-198, 2005.
9. Bartolini, C., Preist, C., Jennings, N.R.: Architecting for Reuse: A Software Framework for Automated Negotiation. In: *Proceedings of AOSE'2002: Int. Workshop on Agent-Oriented Software Engineering*, Bologna, Italy, LNCS 2585, Springer Verlag, pp.88-100, 2002.
10. Bartolini, C., Preist, C., Jennings, N.R.: A Software Framework for Automated Negotiation. In: *Proceedings of SELMAS'2004*. LNCS 3390, Springer-Verlag, pp.213-235, 2005.
11. Benyoucef, M., Alj, H., Levy, K., Keller, R.K.: A Rule-Driven Approach for Defining the Behaviour of Negotiating Software Agents. In: J.Plaice et al. (eds.): *Proceedings of DCW'2002*, LNCS 2468. Springer-Verlag, pp.165-181, 2002.
12. Chavez, V., Maes, P.: Kasbah: An Agent Marketplace for Buying and Selling Goods. In: *Proc. of the First Int. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology*. London, UK, 1996.
13. Chmiel, K., Czech, D., Paprzycki, M.: Agent Technology in Modelling E-commerce Process; Sample Implementation. In: C. Danilowicz (ed.): *Multimedia and Network Information Systems*, Volume 2, Wroclaw University of Technology Press, pp.13-22, 2004.
14. Chmiel, K., Tomiak, D., Gawinecki, M., Karczmarek, P., Szymczak, Paprzycki, M.: Testing the Efficiency of JADE Agent Platform. In: *Proceedings of the 3rd International Symposium on Parallel and Distributed Computing*, Cork, Ireland. IEEE Computer Society Press, Los Alamitos, CA, USA, pp.49-57, 2004.

15. Chmiel, K., Gawinecki, M., Kaczmarek, P., Szymczak, M., Marcin Paprzycki: Efficiency of JADE Agent Platform, Scientific Programming, 2005 (to appear).
16. Dumas, M., Governatori, G., ter Hofstede, A.H.M., Oaks, P.: A Formal Approach to Negotiating Agents Development. In: *Electronic Commerce Research and Applications*, Vol.1, Issue 2 Summer, Elsevier Science, pp.193-207, 2002.
17. FIPA: Foundation for Physical Agents. See `http://www.fipa.org`.
18. Fuggetta, A., Picco, G.P., Vigna, G.: Understanding Code Mobility. In: *IEEE Transactions on Software Engineering*, vol.24, no.5, IEEE Computer Science Press, pp.342-361, 1998.
19. Ganzha, M., Paprzycki, M., Pîrvănescu, A., Bădică, C, Abraham, A.: JADE-based Multi-Agent E-commerce Environment: Initial Implementation, In: *Analele Universităţii din Timişoara, Seria Matematică-Informatică*, Vol. XLII, Fasc. special, pp.79-100. 2004.
20. Governatori, G., Dumas, M., ter Hofstede, A.H.M., and Oaks, P.: A formal approach to protocols and strategies for (legal) negotiation. In: Henry Prakken (ed.): *Procedings of the* 8^{th} *Int. Conference on Artificial Intelligence and Law*, IAAIL, ACM Press, pp.168-177, 2001.
21. JADE: Java Agent Development Framework. See `http://jade.cselt.it`.
22. JESS: Java Expert System Shell. See `http://herzberg.ca.sandia.gov/jess/`.
23. Kowalczyk, R.: On Fuzzy e-Negotiation Agents: Autonomous negotiation with incomplete and imprecise information, In: *Proc.DEXA'2000*, London, UK, pp.1034-1038, 2000.
24. Kowalczyk, R., Franczyk, B., Speck, A.: Inter-Market, towards intelligent mobile agent E-Market places. In: *Proc.* 9^{th} *Annual IEEE Internmational Conference and Workshop on the Engineering of Computer-Based Systems, ECBS'2002*, Lund, Sweden, pp.268–276, 2002.
25. Kowalczyk, R., Ulieru, M., Unland, R.: Integrating Mobile and Intelligent Agents in Advanced E-commerce: A Survey. In: *Agent Technologies, Infrastructures, Tools, and Applications for E-Services, Proceedings NODe'2002 Agent-Related Workshops*, Erfurt, Germany. LNAI 2592, Springer-Verlag, pp.295-313, 2002.
26. Laudon, K.C., Traver, C.G.: *E-commerce. business. technology. society* (2^{nd} ed.). Pearson Addison-Wesley, 2004.
27. Lochner, K.M., Wellman, M.P.: Rule-Based Specification of Auction Mechanisms. In: *Proc. AAMAS'04*, ACM Press, New York, USA, 2004.
28. Lomuscio, A.R., Wooldridge, M., Jennings, N.R.: A classification scheme for negotiation in electronic commerce. In: F. Dignum, C. Sierra (Eds.): *Agent Mediated Electronic Commerce: The European AgentLink Perspective*. LNCS 1991, Springer-Verlag, 19-33, 2002.
29. Maes, P., Guttman, R.H., Moukas, A.G.: Agents that Buy and Sell: Transforming Commerce as we Know It. In *Communications of the ACM*, Vol.42, No.3, pp.81-91, 1999.
30. Michael, S.: Design of Roles and Protocols for Electronic Negotiations. In: *Electronic Commerce Research Journal*, Vol.1 No.3, pp.335-353, 2001.
31. Nwana, H., Ndumu, D.: A Perspective on Software Agents Research. In: *The Knowledge Engineering Review*, 14(2), pp.1-18, 1999.
32. Paprzycki, M., Abraham, A.: Agent Systems Today; Methodological Considerations. In: *Proceedings of 2003 International Conference on Management of e-Commerce and e-Government*, Nanchang, China. Jangxi Science and Technology Press, China, pp.416-421, 2003.
33. Pîrvănescu, A., Bădică, C., Ghanza, M., Paprzycki, M.: Developing a JADE-based Multi-Agent E-Commerce Environment. In: Nuno Guimares and Pedro Isaias (eds.): *Proceedings IADIS International Conference on Applied Computing, AC'05*, Algarve, Portugal. IADIS Press, Lisbon, pp.425-432, 2005.
34. Pîrvănescu, A., Bădică, C., Ghanza, M., Paprzycki, M.: Conceptual Architecture and Sample Implementation of a Multi-Agent E-Commerce System. In: Ion Dumitrache, Catalin Buiu, (Eds.): *Proceedings of the* 15^{th} *International Conference on Control Systems and Computer Science CSCS'15*). "Politehnica Press" Publishing House, Bucharest, 2005, Vol.2, pp.620-625
35. Rolli, D., Eberhart, A.: An Auction Reference Model for Describing and Running Auctions, Wirtschaftsinformatik 2005, Physica-Verlag, pp.289-308.

36. Rolli, D., Luckner, S., Gimpel, H., Weinhardt, C.: A Descriptive Auction Language. In: *International Journal of Electronic Markets*, 2006, 16(1), pp. 51-62.
37. Skylogiannis, T., Antoniou, G., Bassiliades, N.: A System for Automated Agent Negotiation with Defeasible Logic-Based Strategies – Preliminary Report. In: Boley, H., Antoniou, G. (eds): *Proceedings RuleML'04*, Hiroshima, Japan. LNCS 3323, Springer-Verlag, pp.205-213, 2004.
38. Tamma, V., Wooldridge, M., Dickinson, I: An Ontology Based Approach to Automated Negotiation. In: *Proceedings Agent Mediated Electronic Commerce, AMEC'02*. LNAI 2531, Springer-Verlag, pp.219-237, 2002.
39. Trastour, D., Bartolini, C., Preist, C.: Semantic Web Support for the Business-to-Business E-Commerce Lifecycle. In: *Proceedings of the WWW'02: International World Wide Web Conference*, Hawaii, USA. ACM Press, New York, USA, pp.89-98, 2002.
40. Wooldridge, M.: *An Introduction to MultiAgent Systems*, John Wiley & Sons, 2002.
41. Wurman, P.R., Wellman, M.P., Walsh, W.E.: A Parameterization of the Auction Design Space. In: *Games and Economic Behavior*, 35, Vol.1/2, pp.271-303, 2001.
42. Wurman, P.R., Wellman, M.P., Walsh, W.E.: Specifying Rules for Electronic Auctions. In: *AI Magazine* 23(3), pp.15-23, 2002.