UML Models of Agents in a Multi-Agent E-Commerce System

Costin Bădică
Software Engineering Department
University of Craiova
Bvd.Decebal 107, Craiova, 200440, Romania
c_badica@hotmail.com

Maria Ganzha
Department of Informatics
Giżycko Private Higher Educational Institute
ul. Daszyńskiego 9, 11-500 Gizycko, Poland
ganzha@pwsz.net

Marcin Paprzycki
Computer Science, OSU
Tulsa, OK, 74106, USA
and
Computer Science, SWPS
03-815 Warsaw, Poland
marcin@cs.okstate.edu

Abstract

Recently a new model agent-based e-commerce system was proposed, in which rule-based mechanism representation was combined with lightweight modular mobile agent design. Furthermore, need for agent mobility as the optimal solution to satisfy user needs was discussed. The aim of this paper is to introduce UML formalizations of most important agents that appear in the model system as well as presentation of its complete action diagram.

1 Introduction

Recent advances in developing agent environments that support agent mobility (e.g. [4]) combined with advances in general methodology of price negotiations ([9]) resulted in renewed interest in applying software agents in e-commerce. Surprisingly, such interest does not seem to result in actual agent systems being implemented *using agent environments* (we are aware of agent-methodology-based systems implemented in C++, but this is not what we are interested in). Developing such a model system is a goal of our research and we have already designed and implemented a multi-agent e-commerce system in which autonomous agents are engaged in matchmaking, negotiations and contracting on behalf of their users: humans or businesses [6] (see also references to our earlier work collected there). In this way we proceed beyond what is typically discussed in the agent-literature: the "act" of price negotiation itself; and consider a more complete e-commerce scenario consisting of: requesting purchase, matchmaking, negotiating price and completing purchase. Interestingly, the stage between completion of price negotiations and the actual purchase, while involving a number of possibilities, is practically forgotten in the literature.

Our original model system followed proposal outlined in [8] to implement agents capable of negotiation adaptation via dynamically loadable modules. Negotiating agents consisted of three main components: (i) *communication module* – responsible for messages exchanged between agents, ii) *protocol module* – responsible for enforcing the (FIPA) protocol that governed negotiations, and (iii) *strategy module* – responsible for producing protocol-compliant actions necessary to achieve agent goals. Recently we have decided to re-design our system and utilize a more general and flexible agent negotiation framework introduced in [2, 3]. Its authors analyzed the FIPA-standardized auction protocols and have shown that they do not provide enough structure for the development of portable agent-based e-commerce systems. They also outlined a negotiations framework, consisting of multiple *buyers* and a *host* where the negotiations take place. Within the host, the infrastructure for negotiations was provided through a number of sub-agents: *Gatekeeper*, *Proposal Validator*, *Protocol Enforcer*, *Information Updater*, *Negotiation Terminator* and *Agreement Maker* that interact with each-other by direct messaging and via a blackboard. The central point of their framework consisted of (a) a generic negotiation protocol, (b) a negotiation template – a structure that defined all negotiation parameters and thus its mechanisms, and (c) a taxonomy of JESS [5] rules used for enforcing these specific negotiation mechanisms. Obviously, these two approaches can be easily combined. The framework introduced in [2, 3] assumes implicitly that *Buyer* agents are mobile and carry with them the *generic negotiation protocol* thus making them rather heavy. Obviously, our approach based on pluggable modules can be employed here to achieve lightweight mobility. Separately, the *Gatekeeper* sub-agent does not participate in actual price negotiations as it only allows buyers into the negotiation space and provides them with the negotiation protocol and template. After careful analysis we have decided to remove it from

the "negotiation infrastructure" and place in the system as a full-fledged agent. In this way we were able to add to it a number of additional managerial functions. However, this change does not modify the price negotiation framework itself, which was the most important contribution of [2, 3].

When combining the two approaches we had to confront the question: is there any reason for agents to be mobile? In [1] we have responded to it in the context of our proposed model system. We have argued that *agent mobility is the most optimal solution* for the e-commerce model considered there. Then we have discussed why it can be expected that in the future e-stores will provide an infrastructure robust enough for mobile agents to frequent them and negotiate prices. We have followed by arguments why the proposed solution, based on dynamically loadable modules, helps reduce auction-server resource utilization and why *Buyer* agents should not be assembled before they reach their destination. Finally we have discussed why there is no simple solution to the problem of finding the optimal offer when multiple agents negotiate prices within multiple e-stores and thus why our solution is as optimal as any other. Our arguments were supported through an analysis of UML diagrams of two agents directly involved in agent mobility, the mobile *Buyer* agent and the receiving it *Gatekeeper* agent.

The aim of this paper is to present UML-based specifications (statecharts) of agents existing in our system as well as a complete system action diagram.

2 Agents in the System

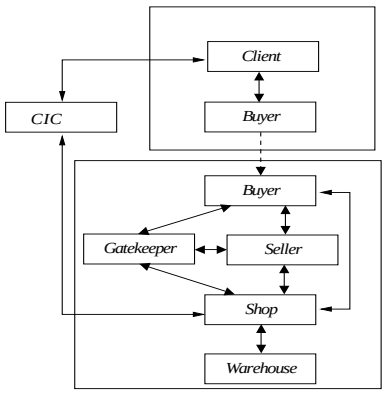

Figure 1. Agent interactions and mobility

Proposed model system represents a distributed marketplace that hosts e-stores and allows e-clients to visit them to purchase products. Stores are approached by multiple customers and, through various price negotiation mechanisms, select the buyer (e.g. winner of negotiations). Figure 1 introduces agents existing in our system and specifies which agents are in direct contact. Furthermore, migration of the *Buyer* agent from the "buyer system" to the "seller system" is denoted by a dashed arrow. Let us now describe each type of agents appearing in Figure 1. Here, we omit detailed description of an auxiliary *CIC* agent, which combines the function of white pages, by storing information (addresses and identifiers) about all *Gatekeeper*, *Shop* and *Client* agents existing in the system (since only agents known to the *CIC* are able to utilize its services we have to register *Shop* agents), and of yellow pages, by storing information about all available products (more information about its role and implementation can be found in [6] and papers referenced there).

2.1 Shop agent

The *Shop* agent acts as the representative of a "user-seller." In our current implementation a predefined number of such agents are created during system initialization (and registered with the *CIC* agent) and persist in the system. The UML diagram representing the *Shop* agent is presented in Figure 2. After being created, each *Shop* agent creates and initializes its *Gatekeeper* and *Warehouse* agents, as well as *Seller* agents (one for each product sold). Initialization of the *Warehouse* agent involves passing information about products available for sale, (see Figure 3), while initialization of the *Gatekeeper* and *Seller* agents involves providing them with initial negotiation protocol and template. Finally, the *Gatekeeper* agent and the list of available products are registered with the *CIC* agent.

After initialization, the *Shop* agent exists in complex state where it supervises negotiations and product flow. As a direct supervisor of price negotiations, the *Shop* agent awaits finish of any of them. As a result of negotiation completion, one of *Seller* agents sends a message containing negotiation details. In case of successful negotiations such a message includes, the final price and the ID of the winning *Buyer*. (Information about completed negotiations is stored in a database for further analysis.) In response, the *Shop* agent sends a message to the *Warehouse* agent and asks it to reserve a given quantity of a particular product (for a specific amount of time). There are now three possible courses of action. **Purchase:** when the winning *Buyer* confirms interest in purchase the *Shop* agent asks the *Warehouse* agent to check the status of the reservation. If it did not expire then the *Shop* informs the *Buyer* that transaction will take place. This event starts the final stage (box "Sale finalization") which includes, among others, payment and delivery. **Shop-rejection:** if the reservation has expired, the *Shop* agent sends a rejection message to the *Buyer* agent. **Buyer-rejection:** if the *Client* agent decides to not to pursue purchase at a given e-store, the *Buyer* agent sends a rejection message to the *Shop* agent, which

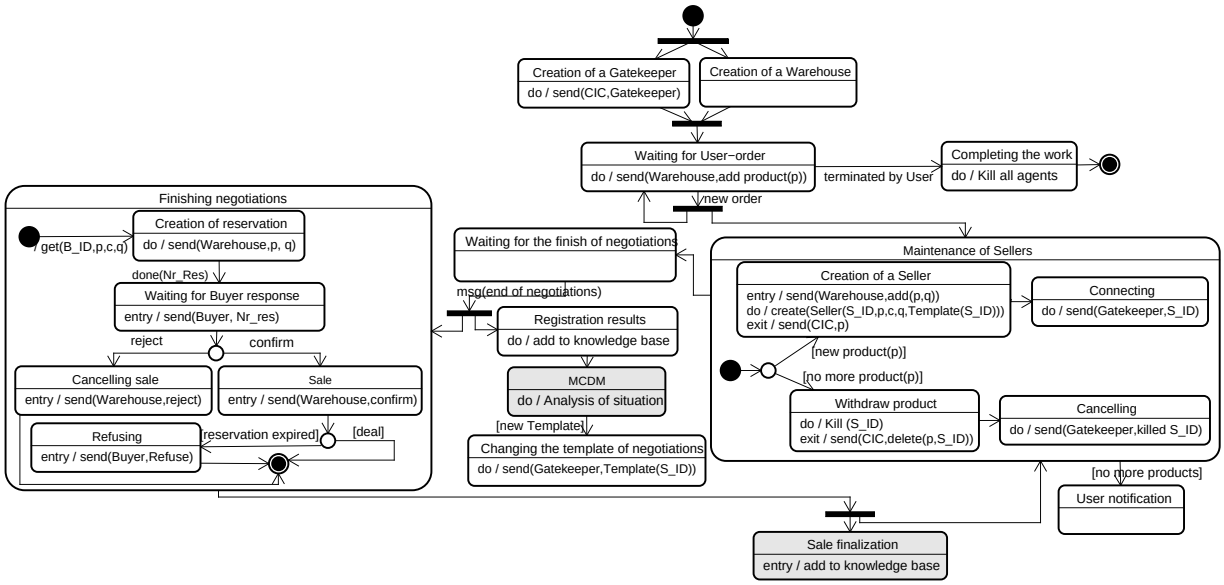

Figure 2. Statechart of the Shop agent

in turn asks the *Warehouse* agent to cancel the reservation. Completing of either branch concludes an instance of price negotiation monitoring.

Separately, the *Shop* acts as the "shop manager" and periodically performs multicriterial analysis (the *MCDM* box) of stored information about, among others, completed negotiations, resulting transactions, and quantities of available products. The *MCDM* module starts during initialization of the *Shop* agent. The *MCDM* analysis may result in changes in the negotiation template (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 [1] figures 2, 3, 4).

2.2 Warehouse Agent

The *Shop* agent interacts directly with the *Warehouse* agent presented in Figure 3. After being initialized the *Warehouse* agent is supplied (by the *Shop* agent) with information about available products and their quantities. Subsequently the *Warehouse* agententers a complex state consisting of two threads: (1) it awaits notifications from the *Shop* agent and (2) manages time triggered events. The *Shop* agent notifies the *Warehouse* agent about: (i) registration of new products made available for sale, (ii) product reservations, (iii) purchase confirmations, and (iv) purchase terminations. Furthermore, it requests checking status of reservations. Timer within the *Warehouse* agent results in periodical checking of existing reservations. Expired reservation are canceled and the total number of available items is appropriately increased, while the *Shop* agent is informed about a new amount of available product. Finally, if some product is sold-off, the *Warehouse* agent informs the *Shop* agent, which may terminate the corresponding *Seller* agent. If the *Seller* agent is terminated, then both the *CIC* and the *Gatekeeper* agents are informed by the *Shop* agent.

2.3 Gatekeeper agent

Gatekeeper agents are created by *Shop* agents and are responsible for: (1) admitting *Buyer* agents to the negotiations (some agents may not be admitted, for instance if they are known to represent a fraudulent and/or disruptive *Client*); (2) in a specific moment releasing *Buyer* agents to an appropriate *Seller* agent so that they can participate in price negotiations (how *Sellers* are released depends on the negotiation mechanism, e.g. fixed price mechanisms allow immediate release, while in the case of auctions *Buyer* agents are gathered for some time and released in groups), and (3) acting on behalf of its *Shop* agent, the *Gatekeeper* manages the process of negotiation mechanism change, i.e. provides *Seller* agents with modified negotiation templates and oversees the template exchange with regards to the waiting and incoming *Buyer* agents. Due to the lack of space, detailed description of this agent can be found in [1].

2.4 Seller agent

Finally, the last agent working within the "supply infrastructure" is the *Seller* agent represented in Figure

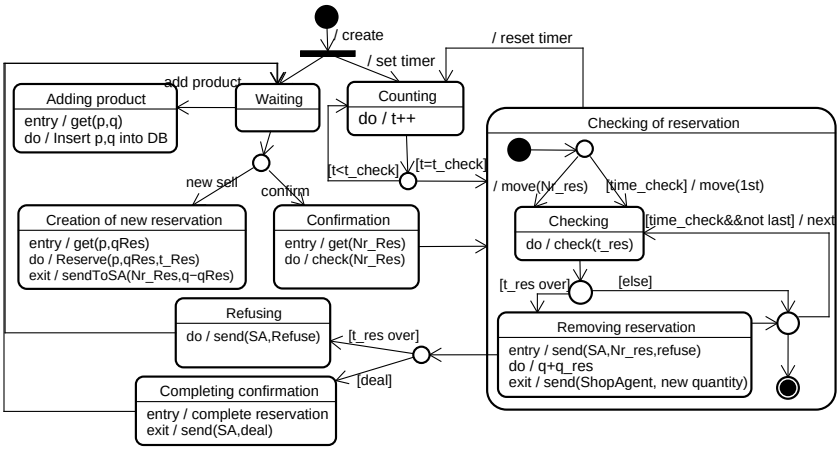

Figure 3. Statechart of the Warehouse agent

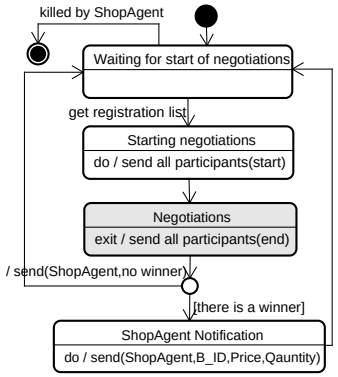

Figure 4. Statechart of the Seller agent

4. Its apparent simplicity is a result of the "Negotiations" box encompassing the *complete* negotiation framework proposed in [2, 3]. Observe that not all negotiations have to end in finding a winner. Separately, all data about negotiations are sent to the *Shop* agent that collects them for further analysis (the "MCDM" box in Figure 2). For instance, a sequence of negotiation failures may result in changes in the negotiation template.

2.5 Client agent

As soon as initialized, the *Client* agent (see Figure 5) enters a complex state consisting of two threads. In one of them it listens for orders from the customer and, after receiving them it starts the purchasing process by: (1) querying the *CIC* agent which stores sell the requested product, and (2) dispatching *Buyer* agents to these e-stores (identified by their *Gatekeeper* agents). In the second thread, it supervises the purchasing process, where decisions are made on the basis of information received from *Buyer* agents. For each purchase order, *Client* agent accumulates messages from *Buyer* agents (each report is also stored in a database for further information extraction). When the wait-time is over (or when all *Buyer* agents have reported), the *Client* agent enters another complex state. In one of its threads it continues listening for messages from *Buyer* agents (obviously, if all have reported then there will be none). In the second thread it goes through a multicriterial decision procedure (the "MCDM" box) that lead to separate execution paths: (1) attempt to complete a selected purchase, (2) cancel existing reservations and request that *Buyer* agents re-engage in price negotiations, or (3) declare the purchase impossible, for specified conditions, and notify the customer. When attempt at completing a purchase is successful, the *Client* agent requests that all *Buyer* agents, responsible for purchasing a given product, self-destruct. When the attempt was unsuccessful then control is returned to the *MCDM* module. As a result, the *Client* agent may pick another offer (including these that arrived in the meantime) and attempt to make a purchase (return to (1) above). It may also decide to switch to path (2) and informs all *Buyer* agents that have reported thus far to cancel reservations and return to price negotiations. Then it resets timer specifying when to start the next *MCDM* analysis. Observe that it is possible that the first *MCDM* analysis was undertaken before all *Buyer* agents completed their "first round" of price negotiations. Some could have contacted the *Client* after it already requested that agents return to price negotiations. In this way, some agents make second attempt at negotiating prices, while some agents have just finished the first. As this procedure continues in an asynchronous fashion *Buyer* agents make different number of attempts at price negotiations. This process continues until purchase is made or abandoned.

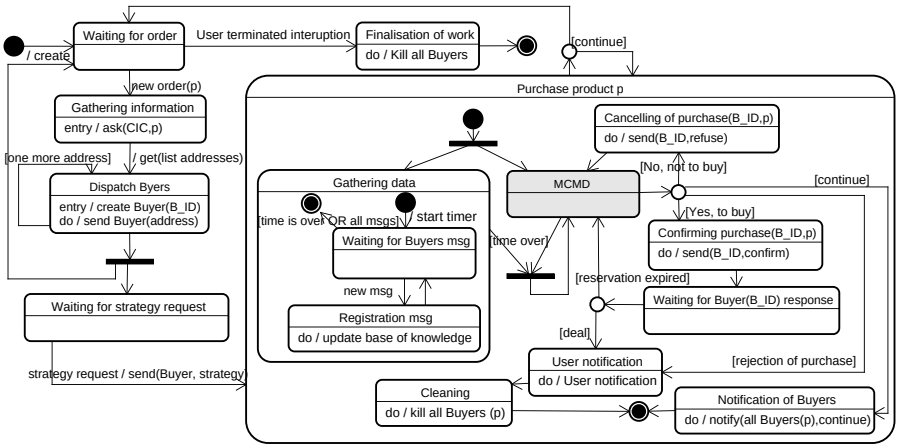

Figure 5. Statechart of the Client agent

2.6 Buyer agent

Finally, *Buyer* agents are the only mobile agents in the system. They are dispatched to all stores that carry product desired by the customer and communicate with the *Gatekeeper* agent to obtain entry to negotiations (if entry is not granted they inform their *Client* agents and are killed). When entry is granted the *Buyer* obtains, from the *Gatekeeper*, the negotiation protocol and template. Then it requests and obtains an appropriate strategy module from the *Client* agent (see Figure 5). When all three modules are installed, the *Buyer* informs the *Gatekeeper* that it is ready and when prompted proceeds to negotiate with an appropriate *Seller* (see Figure 4). 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 *Client* and *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 (after updating negotiation template and strategy modules). This process continues until the *Buyer* agent is "killed" by the *Client* agent. Due to the lack of space, detailed description of this agent can be found in [1].

In summary, in Figures 6 and 7, we present the complete flow of actions in the system.

3 Concluding Remarks

In this paper we have discussed a multi-agent e-commerce system that combines rule-based and mobile agent technologies for implementing flexible automated negotiations. We have focused on formal, UML-based descriptions of agents in the system as well as its complete action diagram. System described here is currently being re-implemented on the basis of these UML formalizations (the previous version of the system, while fully functional [6], did not involve the negotiation framework introduced in [2, 3]). We will report on our progress in subsequent papers.

References

[1] Bădică, C., Ganzha, M., Paprzycki, M.: Mobile Agents in a Multi-Agent E-Commerce System, submitted for publication.

[2] Bartolini, C., Preist, C., Jennings, N.R.: Architecting for Reuse: A Software Framework for Automated Negotiation. In: Proceedings of AOSE'2002: International Workshop on Agent-Oriented Software Engineering, Bologna, Italy, LNCS 2585, Springer Verlag (2002) 88–100.

[3] Bartolini, C., Preist, C., Jennings, N.R.: A Software Framework for Automated Negotiation. In: Proceedings of SELMAS'2004, LNCS 3390, Springer Verlag (2005) 213–235.

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

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

[6] Maria Ganzha, Marcin Paprzycki, Amalia Pîrvănescu, Costin Bădică, Ajith Abraham (2005) JADE-based Multi-agent E-commerce Environment: Initial Implementation, Analele Universităţii din Timişoara, Seria Matematică–Informatică(to appear)

[7] Tamma, V., Wooldridge, M., Dickinson, I: An Ontology Based Approach to Automated Negotiation. In: *Proceedings AMEC'02: Agent Mediated Electronic Commerce*, LNAI 2531, Springer-Verlag (2002) 219–237.

[8] Tu, M.T., Griffel, F., Merz, M., Lamersdorf, W.: A Plug-in Architecture Providing Dynamic Negotiation Capabilities for Mobile Agents. In: *Proceedings MA'98: Mobile Agents*, Stuttgart, Germany, (1998) 222–236.

[9] Wurman, P, Wellman, M., Walsh W.: A Parameterization of the Auction Design Space. In: *Games and Economic Behavior*, 35, Vol. 1/2 (2001), 271–303.

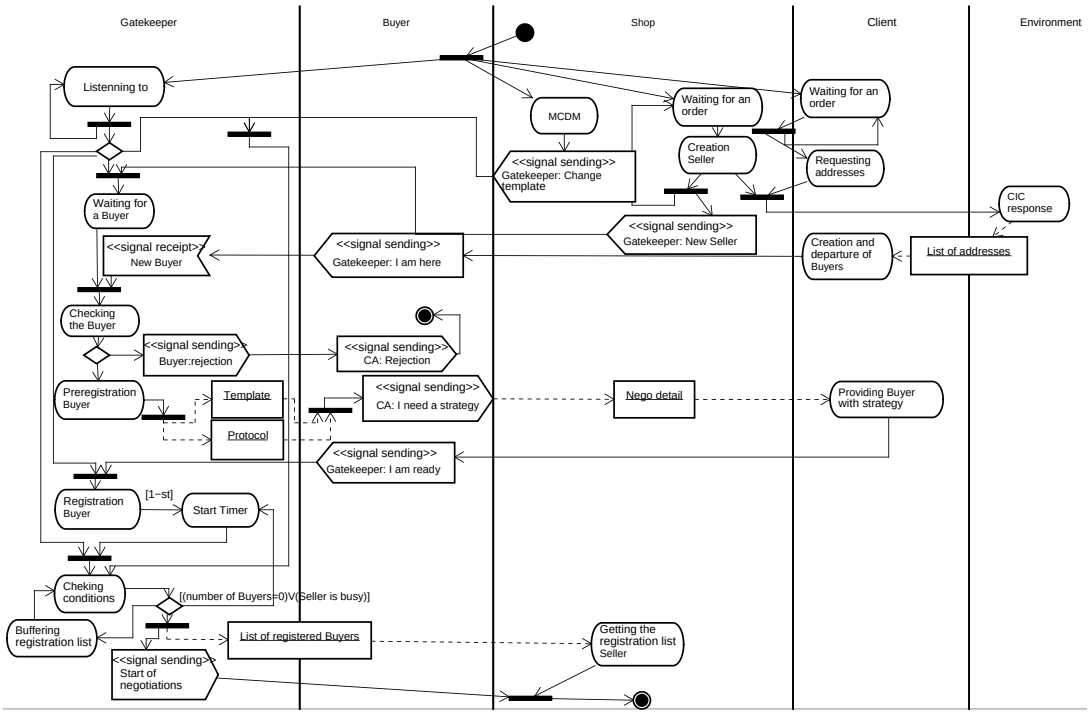

Figure 6. Activity diagram – actions during preparations to negotiations

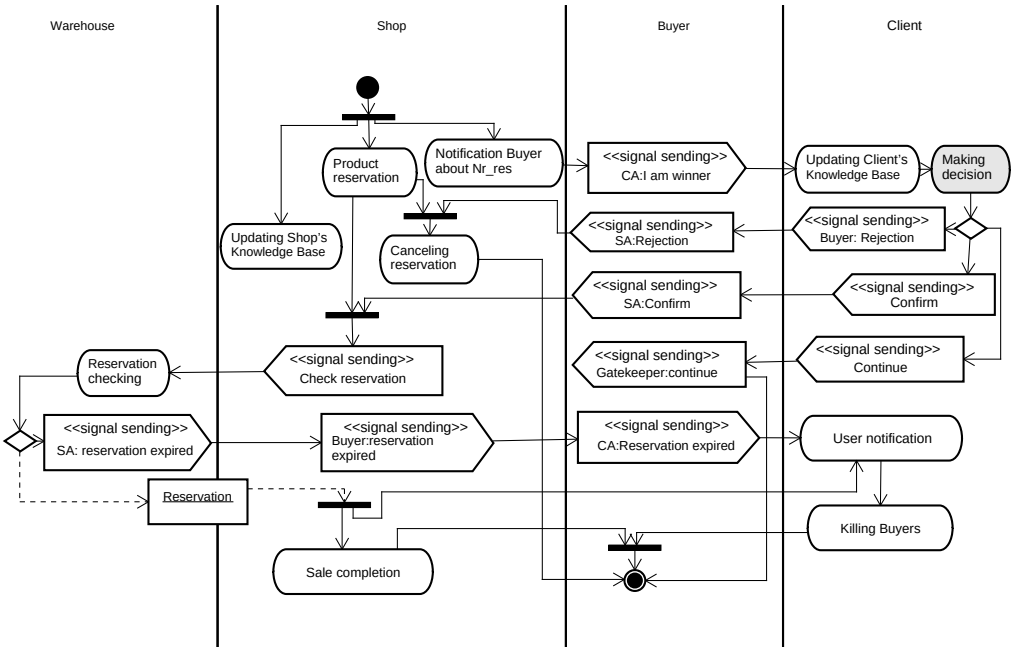

Figure 7. Activity diagram – actions after negotiations are complete