International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:7, 2007

Two Approaches to Code Mobility in an Agent-based E-commerce System

Costin Badica, Maria Ganzha, and Marcin Paprzycki

***Abstract*—**Recently, a model multi-agent e-commerce system based on mobile buyer agents and transfer of strategy modules was proposed. In this paper a different approach to code mobility is introduced, where agent mobility is replaced by local agent creation supplemented by similar code mobility as in the original proposal. UML diagrams of agents involved in the new approach to mobility and the augmented system activity diagram are presented and discussed.

***Keywords*—**Agent system, agent mobility, code mobility, e-commerce, UML formalization.

I. Introduction

RECENTLY, we have proposed a model multi-agent e-commerce system that combined adaptability, mobility and intelligence [3, 4]. There, autonomous agents engaged in matchmaking, price negotiations and contracting (including actually "purchasing" products) on behalf of their "owners." Our proposal build on: (i) multi-agent e-commerce skeleton [5], (ii) flexible framework that allows agents to participate in arbitrary negotiations [1, 2], and (iii) lightweight agents migrating to remote markets and engaging in any form of price negotiations via dynamically loadable modules [5]. Finally, we proceed beyond the "act" of price negotiation. While in [6] negotiations were appended to include matchmaking, we conceptualized inside of a complete scenario consisting of: purchase request, matchmaking, price negotiations and completing purchase.

Our original system [5] followed proposal outlined in [7] where negotiating agents consisted of a skeleton and three modules: *communication module* – responsible for messages exchanged between agents, *protocol module* – responsible for enforcing the (FIPA) protocol that governed negotiations, and *strategy module* – responsible for producing protocol-compliant actions necessary to achieve agent goals.

Recently we have started to re-design our system to utilize a more general and flexible agent negotiation framework introduced in [1, 2]. Its authors analyzed FIPA auction protocols and have shown that they do not provide enough structure for the development of agent-based e-commerce systems. They have also conceptualized negotiations in which multiple *buyers* negotiated price with a *host*. Within the *host* (which is an agent, but plays also a role of a negotiation location), the infrastructure for negotiations was provided through a number of sub-agents: *Gatekeeper*, *Proposal Validator*, *Protocol Enforcer*, *Information Updater*, *Negotiation Terminator* and *Agreement Maker*. The proposed negotiation 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) taxonomy of rules applied to enforce these negotiation mechanisms.

Obviously, the two approaches can be easily combined. (1) In [1, 2] it was assumed that *Buyer* agents are mobile and carry with them the *negotiation protocol*. Obviously, our approach based on pluggable modules could have been employed to achieve lightweight mobility. (2) 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. Thus we have removed it from the "negotiation infrastructure" (and put in the system) and made responsible for a number of additional managerial functions. However, this change did not modify the price negotiation framework itself, which was the most important contribution of [1, 2].

When combining the two approaches we had to confront the question: is there any reason for agents to be mobile? In [3] we have argued that *agent and code 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, by the *Client* agents, before they are send to 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 *Gatekeeper* agent that receives it.

Manuscript received July 31, 2005.

C. Badica is with the University of Craiova, Craiova 200440, Romania, (e-mail: c_badica@hotmail.com).

M. Ganzha is with the Elbląg University of Humanities and Economy, 82-300 Elbląg, Poland (e-mail: ganzha@op.pl).

M. Paprzycki is with the SWPS University, 03-815 Warszawa, Poland and the Oklahoma State University, Tulsa, OK 74106, USA (e-mail: marcin.paprzycki@swps.edu, phone: +48-606612166).

The main thrust of research initiated in [7] and extended and summarized in [5] stems from a basic observation that it is practically impossible for agents to be at the same time *effectively mobile* and *intelligent* [8]. Intelligence, however compactly represented, makes agents "heavy" and, moreover, the more "knowledge" they carry with them, the "heavier" they get. As a result, *the more intelligent agents are the less mobile they become*. Therefore it was proposed that only necessary modules are to be sent across the network and loaded by agents preparing to participate in prices negotiations. Recently we have realized that there exists another solution to the problem of combining agent mobility and intelligence. It is based on "proxy agents" that bid for products on behalf of users and that are created within the eBay auctioning system.

This paper is devoted to discussing this solution and is organized as follows. In the next section we briefly summarize the design of the original system as well as agents populating it. We follow, in Section III with the description of the modified system and the *Gatekeeper* as well as the *Client* and the *Buyer* agents that have changed their roles vis-à-vis the original system. Finally, we present an action diagram of the "negotiation preparation" stage of the operation of the system.

II. Original System Design

Our e-commerce model mimics a distributed marketplace that hosts shops carrying products for sale, and clients that visit them and attempt at purchasing these products. Clients negotiate prices with one or more shops (through a number of possible mechanisms selected by the shop, dynamically for each product) and choose from which to make a purchase. In the case when shops are approached by multiple clients and when they use auction-type negotiation mechanisms (instead of fixed pricing) they can choose the buyer (auction winner). Note that we consider only situations when price negotiations ended in success (final price was higher than the reserved prices of the client and the shop); otherwise transaction is not possible (however, or system can deal with such a situation). When price negotiation ends successfully we follow the eBay/airline transaction model, where a success in price negotiations does not have to result in an actual purchase. Thus, an item is put "on hold" (reserved) for a limited amount of time. Within this time client has to issue an actual purchase order. If such an order is not delivered to the store in time, the reservation expires and the item is returned to the pool of available goods. In the case of an unsuccessful purchase attempt, client may decide to try again, or to abandon the task.

The top level conceptual architecture of the system depicting the described above system operation, in terms of agents existing in the system and their interactions, is shown in Fig. 1. Let us now describe in more detail each agent appearing in that figure and their respective functionalities.

A *Client* agent (*CA*) acts within the marketplace on behalf of a "user" that seeks a particular product. Similarly, a *Shop* agent (*SA*) represents "user" who plans to sell products within the e-marketplace. After being created the *CA* registers with the *Client Information Center* (*CIC*) agent and awaits orders from its owner. The *SA* creates its supporting agents: *Gatekeeper* (*GA*), *Warehouse* (*WA*) and multiple *Seller* agents (one for each product to be sold) and then registers itself and the *GA* with the *CIC* agent. Note that returning *Client, Shop* and *Gatekeeper* agents will receive their existing *ID*s. In this way we provide support for the future goal of agent behavior adaptability as agents in the system will be be able to recognize their counterparts and differentiate their behavior depending if this is a "returning" or a "new" agent.

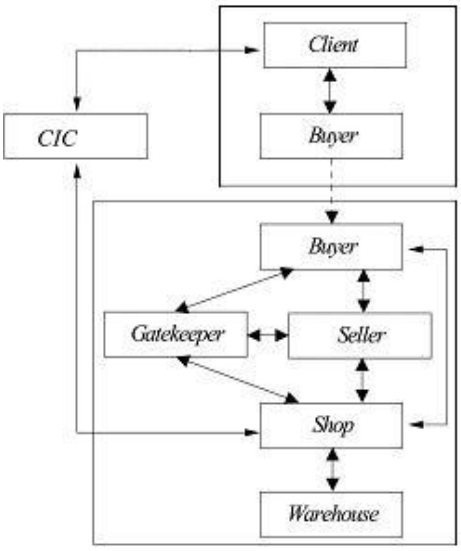

Fig. 1 The original e-commerce environment; solid arrows – communication; dashed arrow – agent movement; rectangular boxes surround buyer and seller systems and agents populating them.

There is only one *Client Information Center* (*CIC*) agent in the system. It is responsible for storing, managing and providing information about all "participants" existing in the system. This information is stored in the *Client Information Database* (*CICDB*). The *CICDB* combines the function of *registry*, by storing information about and unique *ID*s of all "users" and of *yellow pages*, by storing information about all shops known in the marketplace and their offered products [6].

Upon receiving an order from its user the *CA* communicates with the *CIC* agent to obtain list of e-stores that carry the requested product. In the next step it creates one *Buyer* agent (*BA*) for each such store. After *BA*s are released the *CA* awaits messages from its *BA*s and sends them appropriate *negotiation strategy modules*. Then the *CA* awaits messages about results of price negotiations and upon reception performs multicriterial analysis, which store to buy from (factors such as price, history of interactions with a given shop, delivery conditions etc. can be considered). Depending on the success or failure of purchase the *CA* either informs user about success or performs another multicriterial analysis and on the basis of it may decide to retry purchase or abandon the task (and notify its owner about this fact).

Buyer agents (*BA*) arrive at e-stores and communicate with the *Gatekeeper* agents to be admitted to the negotiations. Upon entry, they obtain (from the *GA*) the *negotiation protocol* and the *negotiation template* [1, 2]. In the next step, they request an appropriate strategy module (that is dependent on the negotiation template) and upon its reception inform the *GA* that they are ready to participate in price negotiations. When negotiations are complete *BA*s inform their *CA*s and either (i) attempt at making a purchase, (ii) re-enter negotiations, or (iii) self-destruct. The exist as long as attempts at making purchase of a given product are repeated.

On the supply side, a single *Shop* agent (*SA*) is created for each "merchant" in the system and is responsible for creating *Seller* agents for each product sold. These agents represent the negotiation infrastructure introduced in [1, 2]. The *SA* should be understood as a "store manager" that controls the flow of commodities, on the basis of multicriterial analysis adjusts the negotiation templates and strategies used by *Seller* agents etc.

The *Seller* agent (*SeA*) embodies functions of the *host* (sans the *GA*) introduced in [1, 2]. After being created by the *SA* its only role is to facilitate price negotiations.

The *Warehouse* agent (*WA*) is created to manage the stocks of available commodities. After being created it is informed by the *SA* about available products and their quantities. When a reservation is made (as a result of successful negotiations) the *WA* keeps it available for a pre-specified time. It also controls the quantity of available products and informs the *SA* when any of the goods is sold-out. In the future, the *WA* may become the interface to the product supply subsystem.

Finally, the *Gatekeeper* agent (*GA*) is the only agent that has substantially changed its role vis-à-vis that described in [1, 2]. It is created by the *SA* as a full-fledged agent of the system. Its main role is to be an interface between *BA*s and *SeA*s. First, it admits *BA*s to negotiations and provides then with the negotiation protocol and the current negotiation template. Second, in suitable moment it releases *BA*s to appropriate *SeA*s. Finally, it manages updates of negotiation templates. Note that the *GS* admits to negotiations only "complete" *BA*s – that have all modules installed and confirmed that are ready.

III. Modified System

When analyzing the above described system one can realize that there is another way of attempting at balancing mobility and intelligence. Let us consider what happens during negotiation "preparations." The *GA* communicates with incoming *BA*s and after admitting them to the e-store it outfits them with the generic negotiation protocol and the current negotiation template. Subsequently *BA*s request negotiation strategy module from their *CA*s. It is easy to see that only actions that involve the *CA* are (1) sending the *BA* to the store, and (2) sending it the *negotiation strategy module*. Let us combine these observations with the notion of a user proxy agent that is bidding on behalf of an eBay client, but is created locally, and we can construct a different scenario. Here, the *CA* communicates with the *GA* and requests that the *GA* creates a *BA* that will act on its behalf. When a decision is made that a representative of that *CA* can be admitted to the negotiations (which is exactly the same admission procedure as before), the *GA* creates a generic *BA* consisting of all the same modules as before: the skeleton, the communication module, the negotiation protocol and the negotiation template. In addition this newly created *BA* obtains information about its *CA* is and proceeds to request the negotiation strategy module. At the end of this process we have obtained exactly the same situation, where a *BA* representing a given *CA* through a link to it and its negotiation strategy module is ready to get involved in price negotiations on its behalf. Agent communication in the modified system is depicted in Fig. 2 (compare with Fig. 1).

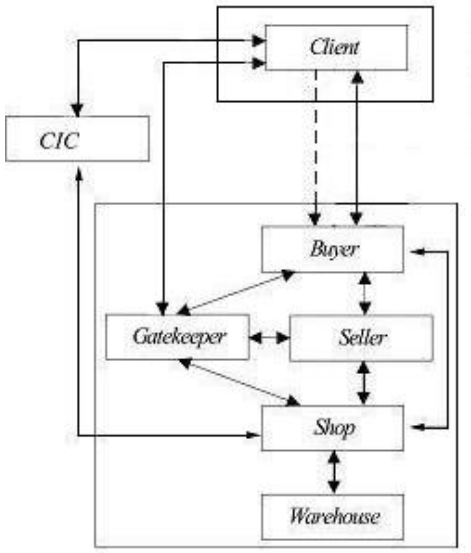

Fig. 2 The modified e-commerce environment; solid arrows – communication; dashed arrow – strategy module migration; rectangular boxes surround buyer and seller systems and agents populating them.

Before we proceed, let us make a few comments. First, let us observe that in the proposed scenario we are able to substantially further reduce the total network utilization. Instead of sending a complete mobile agent, we send only a request for one to be created (independently, ability to remotely create agents exists already in some agent platforms). The only large data packet that is to be transferred is the strategy module. Second, this approach provides an extended e-store security, as the *BA*s are created locally and thus can be assured that they are safe (they do not carry with the malicious code that can attack the server). Third, the only possible problem with the proposed approach is the question of user trust – can the *BA* created within an unknown e-store be trusted. However, since in the proposed approach all *BA*s are to be generic, it can be assumed that they can be verified if they have been tampered with.

Let us no look into details of the three agents that have been changed to accommodate the new approach: the *Client*, the

Buyer and the *Gatekeeper*. Let us note that the remaining agents have not been changed and thus their descriptions are omitted (they can be found in [3, 4]).

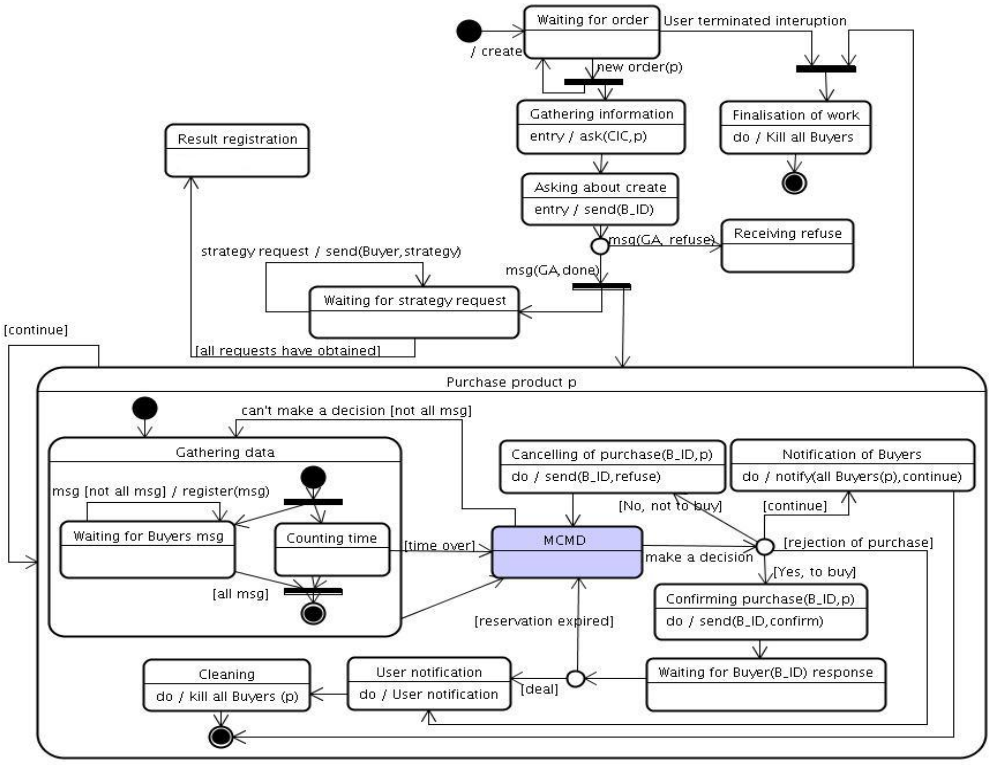

Fig. 3 UML statechart diagram of the *Client* agent

A. Client Agent

The UML statechart diagram of the *Client* agent is presented in Fig. 3. After creation the *CA* waits for an order from the user. When such order is received the *CA* communicates with the *CIC* agent to obtain addresses of *Gatekeeper* agents representing shops that sell thought products. In this paper we assume that all *GA*s can create *Buyer* agents; this assumption can be relaxed and a mixed environment consisting of *GA*s of two types (that can and cannot create *BA*s) can be created. Upon reception of the list of addresses of e-stores of interest the *CA* sends messages (containing *ID* number of new *Buyer* agents to be created) to all pertinent *GA*s. After all requests to create *BA*s have been sent, the *CA* awaits messages: (a) rejections by the *GA*, (b) requests for *strategy modules*, and (c) results of price negotiation (within the "Purchase product p" state – bottom panel in Fig. 3). During a specified time period, the *CA* gathers messages send by its *BA*s, containing results of negotiations. If messages from all *BA*s have been received or the wait-time is over the *CA* analyzes the situation (the *MCDM* state-box) and makes a decision about product purchase. If not all messages were received and/or the *Client* can not make a decision it comes back to the "Gathering data" state. If all messages have been received and *CA* cannot make a decision to buy on the basis of obtained "offers" and at the same time is not ready to abandon the purchase, it orders *Buyer* agents to return to negotiations.

B. Buyer Agent

Obviously, since in the proposed system the *BA* is created by the *Gatekeeper* (instead of the *CA*), the statechart diagram of the *Buyer* agent had to change in comparison with that presented in [3]. The modified statechart diagram is presented in Fig. 4. When comparing it with Fig. 4 of [3] it is easy to see that, as expected, changes concern only initial parts of *Buyer* agent operation. Since all *BA*s are instantiated by the *GA*, they are created already "inside" of the negotiating *host* [1, 2] and thus they do not need to ask for permission to enter. Furthermore, as a part of their initialization, they receive the *generic protocol* and *current negotiation template*. Therefore, they just have only to request, form their *CA*, an appropriate strategy module and from that moment on their actions follow the same rules of previously conceptualized *Buyer* agents [3]. Note that negotiations (box "Negotiation Process") match these proposed and UML represented in [1, 2].

C. Gatekeeper Agent

In [3, 4] we have moved the *Gatekeeper* agent from the negotiation infrastructure, where it was a sub-agent within the *host*, into the system and made it a full-fledged agent with a number of managerial functions (see Section 4 and Fig. 2 in

[3]). Here we proceed even further, by making the *GA* responsible for creating *Buyer* agents (on the request of *Client* agents). Complete statechart diagram of the *Gatekeeper* agent can be found in Fig. 5.

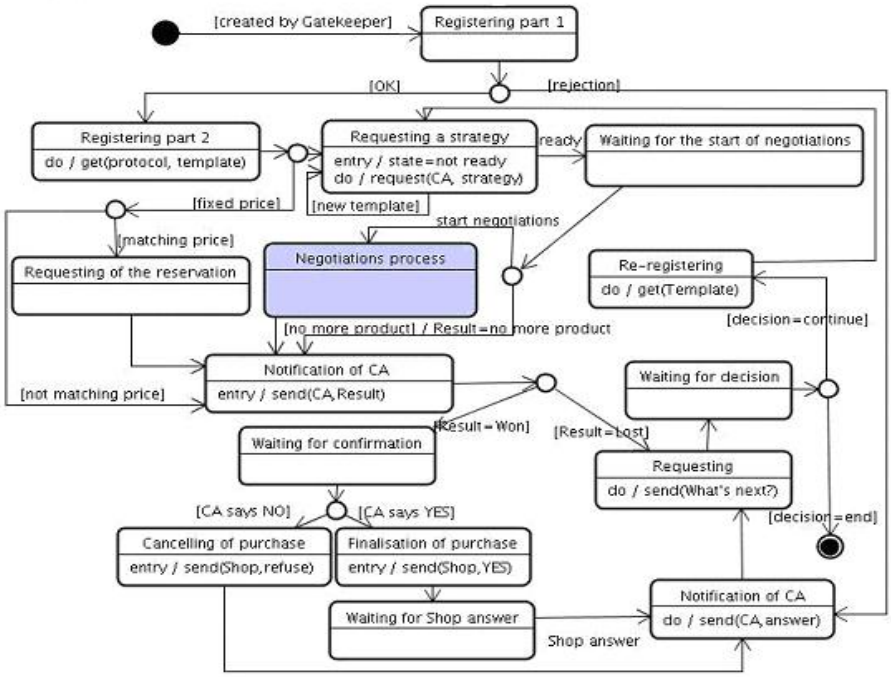

Fig. 4 UML statechart diagram of the *Buyer* agent

A simple comparison shows that the "new" *GA* differs from the previous version only in the part related to creation of *Buyer* agents. After its creation, the *GA* awaits messages from *SA*s and *CA*s. Messages from *SA*s are exactly the same as before and may concern (i) creation of a new *Seller* – *GA* has to get ready to support negotiations for it (state "Preparing negotiations"), (ii) killing of a *SeA* (in case of product being exhausted and the *SA* deciding that it will not be replenished), and (iii) changing a negotiation template. *Client* agent messages request creation of a new *Buyer* agent and contain its expected *ID*. The *GA* checks if a representative of a given *CA* should be admitted to negotiations – can it be trusted? (here, trust is understood very broadly) – state "Checking CA." If this *CA* is considered worthy business relationship, the *GA* creates a *Buyer* agent. It is assumed that *BA*s created this way will be exactly the same as in the original system (when they were created by the *CA*). If the *CA* cannot be trusted, the *GA* sends a rejection message. In the case of changing the negotiation template, the *GA* "suspends" the *Buyer* agent creation process. If a list of agents ready to participate in negotiations is non-empty and the *Seller* agent is not busy, all *BA*s are send to the *SeA* to complete negotiations utilizing their current templates (and the *BA* creation process re-starts). If the *SeA* is busy, the registration list is deposited in a *Buffer* and awaits for the *Seller* to be free while the *BA* creation process re-starts. All newly created *Buyers* will receive the new template and will be processed only when the *Buffer* is empty. Lastly, all *BA*s that did not receive their strategy module have to request a new one to match the new negotiation template.

Finally, to combine what we have discussed thus far, we present in Fig. 6 an activity diagram of the stage crucial for our considerations: negotiation preparations. The remaining two stages (system initialization and finalization of purchase) remain practically the same as in [3, 4] and are thus omitted.

IV. Concluding Remarks

In this paper we have introduced a different approach to resolving the conflict between agent mobility and intelligence. By carefully analyzing the process of and agent participating in negotiations described in [1, 2] we were able to delineate the private and the public parts of *Buyer* agents and use this information to redesign the system. As a result we were able to further reduce network utilization. We are in the process of implementing the proposed framework, as specified by the UML diagrams presented here and in [3, 4]. We will report on our progress in subsequent papers.

References

[1] C. Bartolini, C. Preist, N. R. Jennings, "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

[2] C. Bartolini, C. Preist, N. R. Jennings, "A Software Framework for Automated Negotiation," in: Proceedings of SELMAS'2004, LNCS 3390, Springer Verlag, 2005, 213-235.
[3] C. Badica, M. Ganzha, M. Paprzycki, "Mobile Agents in a Multi-Agent E-Commerce System," submitted for publication.
[4] C. Badica, M. Ganzha, M. Paprzycki, "UML Models of Agents in a Multi-Agent E-Commerce System" submitted for publication.
[5] M. Ganzha, M. Paprzycki, A. Pirvanescu, C. Badica, A. Abraham, "JADE-based Multi-agent E-commerce Environment: Initial Implementation," Analele Universitatii din Timisoara, Seria Matematica – Informatica, to appear.
[6] D. Trastour, C. Bartolini, C. Preist, "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 (2002) 89–98.
[7] M. T. Tu, F. Griffel, M. Merz, W. Lamersdorf, "A Plug-in Architecture Providing Dynamic Negotiation Capabilities for Mobile Agents," in: Proceedings MA'98: Mobile Agents, Stuttgart, Germany, 1998 222-236.
[8] Wooldridge, M.:An Introduction to MultiAgent Systems}, John Wiley & Sons, 2002.

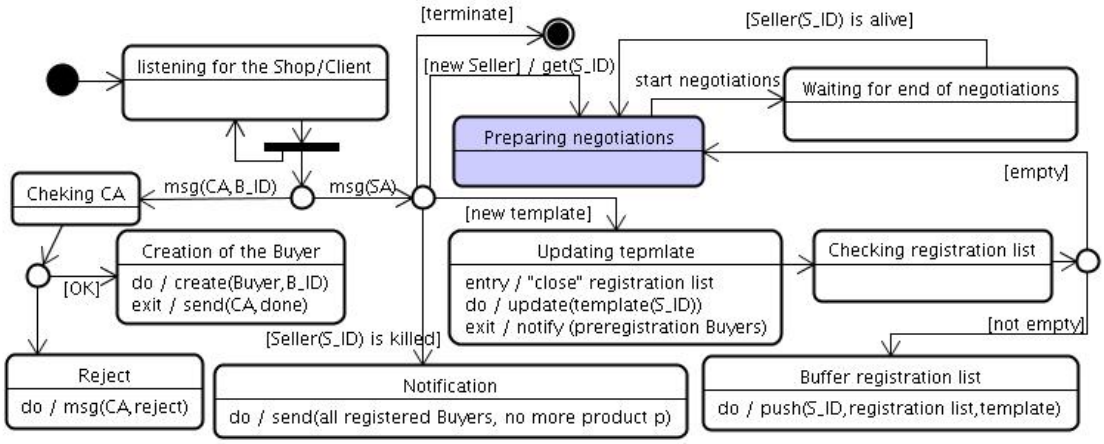

Fig. 5 UML statechart diagram of the *Gatekeeper* agent

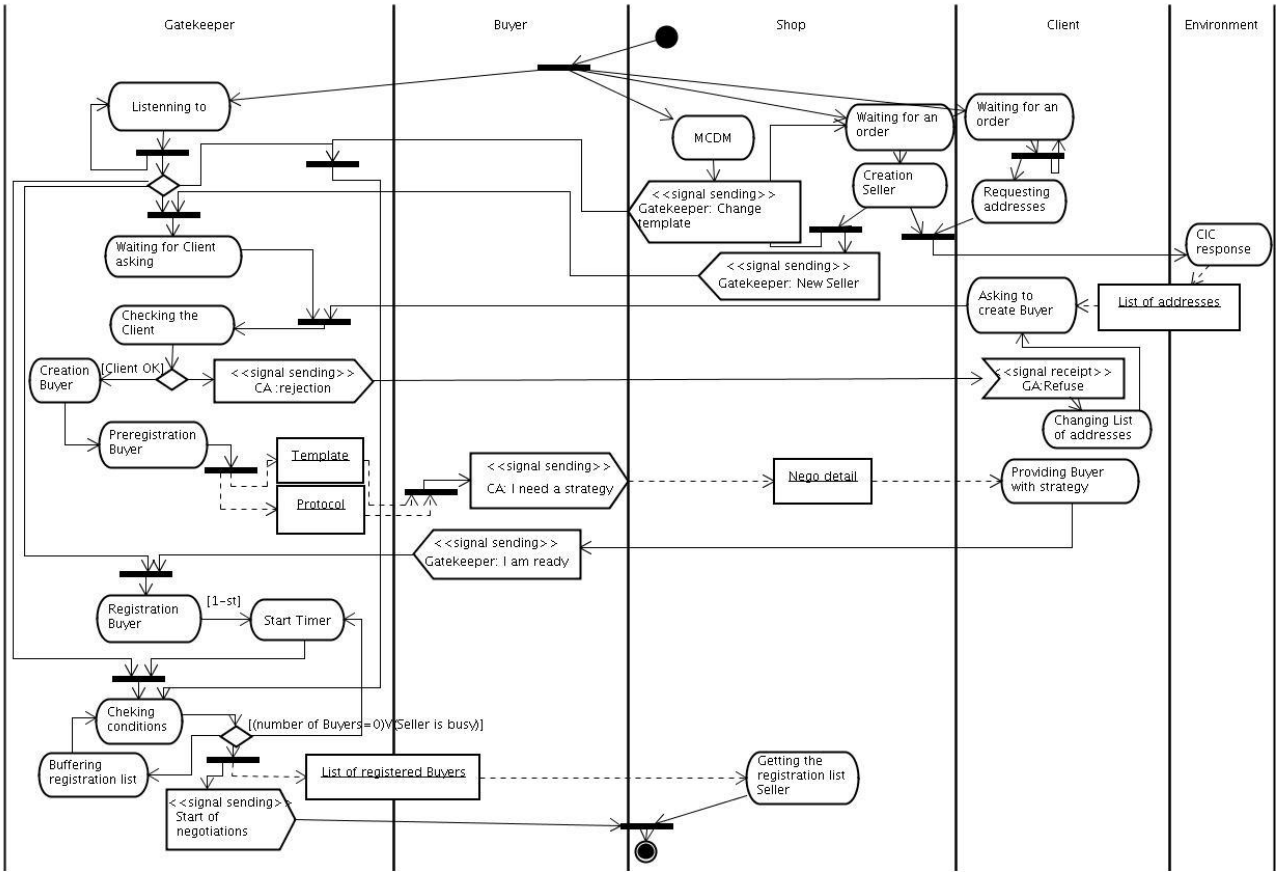

Fig. 6 Activity diagram of negotiation preparations