Introducing Commodity Flow to an Agent-Based Model E-commerce System

Tomasz Serzysko
*Department of Mathematics and Information Science,
Warsaw University of Technology, Poland*

Maria Ganzha, Maciej Gawinecki, Paweł Kobzdej, Marcin Paprzycki
System Research Institute, Polish Academy of Sciences, Poland

Costin Badica
Department of Computer Science, University of Craiova, Romania

Abstract

In our model agent-based e-commerce system [2] we have assumed that a certain number of items of a given product is available for sale. In this note we introduce a model logistics subsystem and discuss how it will be integrated with the system.

1. Introduction

Currently, we are developing and implementing a model agent-based e-commerce system (see [2] and references collected there). In this system multiple buyer agents attempt at making purchase by participating in price negotiations in e-stores and selecting the best offer, while e-stores attempt at maximizing profit resulting from product sales. Thus far our attention was focused on buyer-seller interactions. By assuming that products are in the warehouse we have omitted the question where do they come from. The aim of this work is to describe how our system can be extended to include product restocking processes.

Before proceeding let us make a few observations. First, the proposed logistics subsystem is not "stand alone" (e.g. similar to that considered in [3, 1, 6]). Instead, it has been created within the context of our e-commerce system, which has directly influenced its design. Second, while somewhat similar, processes involved in e-store re-stocking a warehouse differ from client making a purchase in an e-store (e.g. in product demand prediction, interactions with wholesalers, methods of price negotiations that involve more "conditions," offer selection criteria, etc.) Therefore we have created a separate logistics subsystem (instead of re-using already modeled functions; e.g. price negotiations). Third, this note is devoted to the agent structure and agent interactions and, due to the space limitations, we omit important topics like: forecasts derivation, offer evaluation etc. However, these functions can be encapsulated into modules that can become a part of an appropriate agent. Therefore readers should envision that, for instance, when we write that "received offers are evaluated," then their favorite method of offer evaluation has been utilized.

To proceed, first, we briefly describe our e-commerce system. We follow with assumptions that underline the logistics subsystem and description of new agents that were introduced. Finally, we present the sequence diagram of restocking and use it to discuss in detail how this process will take place in our system.

2. System Description

Our system is a distributed marketplace in which software agents perform e-commerce functions (see [2] for details, the Use Case diagram in particular). *User-Client* is represented by the *Client Agent* (*CA*). The *CA* is autonomous and when a purchase order is communicated by the *User-Client*, it works until either it is completed, or purchase is abandoned. The *CA* communicates with the *Client Information Center* (*CIC*), which facilitates information which e-stores sell which products (yellow-page matchmaking). For each store that sells the requested product, the *CA* delegates a *Buyer Agent* (*BA*) to participate in price negotiations and if successful, possibly attempt at making a purchase (successful price negotiations result in a product reservation for a specific time; after which products

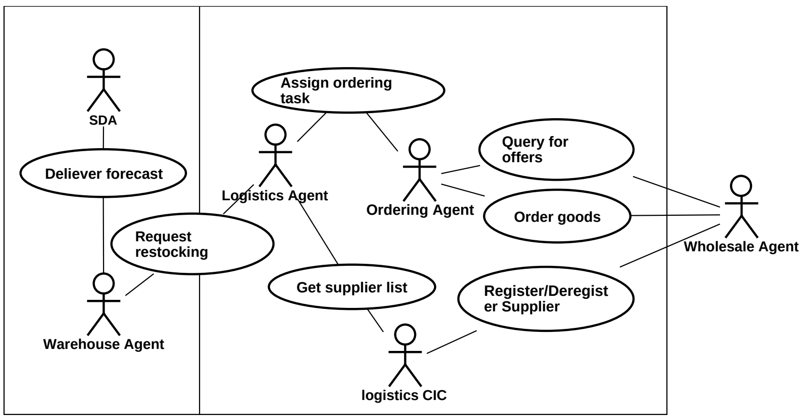

Figure 1. Use Case of the logistics subsystem

that have not been purchased are available for sale again). Since multiple *BA*s representing the same *CA* can win price negotiations the *CA* makes the decision if either of available offers is good enough to make a purchase. *Buyer Agents* can participate in negotiations only if the *Gatekeeper Agent* (*GA*) admits them (if they are trusted; e.g. *BA*s that win price negotiations but do not make a purchase may be barred from subsequent negotiations). The *GA* represents a given e-store and is created by the *Shop Agent* (*SA*). The *SA* is the central manager and facilitating the selling process it utilizes the *GA*, and a set of *Seller Agents* (*SeA*) that negotiate price with incoming *BA*s, as well as a *Warehouse Agent* (*WA*) that is responsible for inventory and reservation management. Thus far, the *WA* was responsible for managing product reservations and the inventory. Specifically, (1) before a new price negotiation the *WA* "reserved" a given product—so that if negotiation ended successfully there was an item that could be sold; (2) when a reservation ended in purchase, it adjusted product counters; and (3) when a reservation expired it also adjusted products counters. However, the *WA* was always envisioned as the "gateway" between the store and suppliers, which is one of the foci of this note.

3. Assumptions behind the logistics system

Let us now specify assumptions that underline design of the logistics subsystem:

1. Nowadays, except of largest store-chains (e.g. TESCO, WalMart), companies outsource transportation activities (considered non-core business activities) to specialists (e.g. UPS, Mayflower). However, we assume here that suppliers are still responsible for facilitating transportation. Therefore, we omit transportation related processes and focus only in interactions between e-stores and suppliers.
2. As a result of (1), transportation costs are assumed to be paid by the supplier and included directly in product price (e.g. discount on delivery costs, will manifest itself in the total price).
3. While in the original system auction-based price negotiations were used, here we opted for the simplicity of the *FIPA Contract Net Protocol* [4]. Therefore, in the logistics subsystem, a single round of negotiations consisting of a call for proposals and evaluation of responses, is used.

4. New functions and agents

In order to perform logistics-related tasks, several new roles were introduced; some of them have been delegated to agents existing in the system, while others warranted adding new agents. Specifically:

- *demand estimation*—to draw information from sales data and/or external premises to predict future sales of products,
- *warehouse monitoring*—to observe supply levels and react in case of a risk of dropping below values considered sufficient to satisfy estimated demand,
- *order management*—to coordinate issuing orders for goods, to assist in evaluating received offers,
- *ordering goods*—to contact suppliers for their offers and to select the best offer,
- *selling goods*—in the system suppliers were also modeled; while goods are acquired without extensive price negotiations, "someone" has to deliver proposals to ordering components,

- *logistics yellow pages*—the role of the "logistics CIC" is very similar to the original *CIC* ([2]); it has to provide lists of potential suppliers of products; obviously, it is possible for a shop to become a supplier for another shop and to suggest that the two *CIC*s could be joined, but we decided to clearly separate the client-side from the shop and from the supply-side. Another reason for this separation was that while some shops may not be interested in becoming wholesaler, we would have to make changes to the original CIC data structure (e.g. wholesaler—yes/no). Finally, since the logistics subsystem does not involve auctions, the separation is even more warranted.

Let us now see how these tasks/roles could be placed in our system. The *demand estimation* role was attributed to the existing *Shop Decision Agent* (*SDA*), responsible for the "knowledge management" functions (e.g. trust management, sales trend data mining, etc.) in the shop.

The *warehouse monitoring* role is already a part of the existing *WA*. The difference is that now *WA* becomes a proactive manager of supplies; acting on predictions supplied by the *SDA*.

Fulfillment of the *order management* role required introduction of the *Logistics Agent* (*LA*), which became the "central manager" of the logistics subsystem. It is responsible for contacting the *logistics CIC* for the list of potential suppliers and managing a pool of agents responsible for ordering goods from "wholesalers." Finally, it collects and manages data related to supplier reliability. This data, in turn, will be one of factors in selecting the supplier.

The *ordering goods* process is facilitated by the *Ordering Agent* (*OA*), which is also a new agent. Its task is to issue a call for proposals, collect responses and select the best offer taking into account factors such as: price, delivery time, reliability etc. Let us recall that due to the modularity of agent design ([2]), our system is flexible enough so that *any* method of selecting an offer can be applied (it can be encapsulated in a module and plugged into the *OA*).

The *selling goods* role is realized by a very simple *Wholesale Agent* (*WhA*). Its role is to respond to CFP's incoming from *OA*s. Currently we assume that *WhA*s receive instructions in what way to generate a stream of responses to the CFPs.

Finally, *logistics yellow pages* are facilitated by the *logistics CIC Agent*. Its role is to store a complete list of suppliers and products that they sell. Obviously, the *logistics CIC* uses the original product ontology ([2]), extended by the logistics ontology. When the system is initialized, each *WhA* registers with the *logistics CIC* and provides it with a list of products for sale.

What was described thus far is summarized in an UML use case diagram presented in Figure 1.

5. Typical Product Restocking Process

Let us now describe the processes involved in restocking the warehouse. Here, we skip the description of system initialization, and start with the *Shop Decision Agent* sending a forecast to the *Warehouse Agent*. The sequence of actions resulting form such a forecast is depicted, as a UML sequence diagram in Figure 2.

The *SDA* communicates the forecast to the *WA* by sending a *FIPA Inform* message containing the *PredictionDescription* (which contains all necessary data such as: product ID, amount of predicted sales, standard deviation of sales, expected purchase price, period for which this forecast is valid, etc.). We assume that the *SDA* forecasts are of the type: until a new forecast, weekly sales are expected to be 45 items of a given product. Forecasts can be issued at specific times (e.g. once a week or once a month) and their frequency depends on the information found in data analyzed by the *SDA* to derive forecast(s).

The *WA* starts by examining current stock of a given product, and if current supplies are sufficient, it sets up to check their levels at the end of the time unit specified in the forecast (i.e. forecasts specified on weekly basis are checked once a week). If stocks are insufficient, the *WA* utilizes the *FIPA Request Protocol* (FIPA specification SC00026) and FIPA SL language [4] (used in all agent interactions), to communicate with the *Logistics Agent*. The initial message from the *WA* is the *FIPA Request* message sent to the *LA* and it contains *OrderRequest* action with the *OrderRequestDescription*. The *OrderRequestDescription* contains the necessary information specifying the order to be made: product ID, preferred delivery time, amount and maximum price. Delivery time and amount are computed based on the current product level, predicted delivery time and an overall inventory strategy.

Upon receiving the request, the *LA* dispatches a query to the *logistics CIC* to obtain a list of suppliers of a given product. Ensuing conversation conforms to the *FIPA Query Protocol*, starting with the *FIPA Query-Ref message* containing the *CICQuery* action with the *Product ID*. The *logistics CIC* responds with the *FIPA Inform-Ref* message containing the *CICResponse* with a list (possibly empty) of suppliers. Empty list results in a *FIPA Failure* message (with *OrderRequestResult* set to failure) send by the *LA* to the *WA*. Similar response is sent when the *logistics CIC* cannot be contacted.

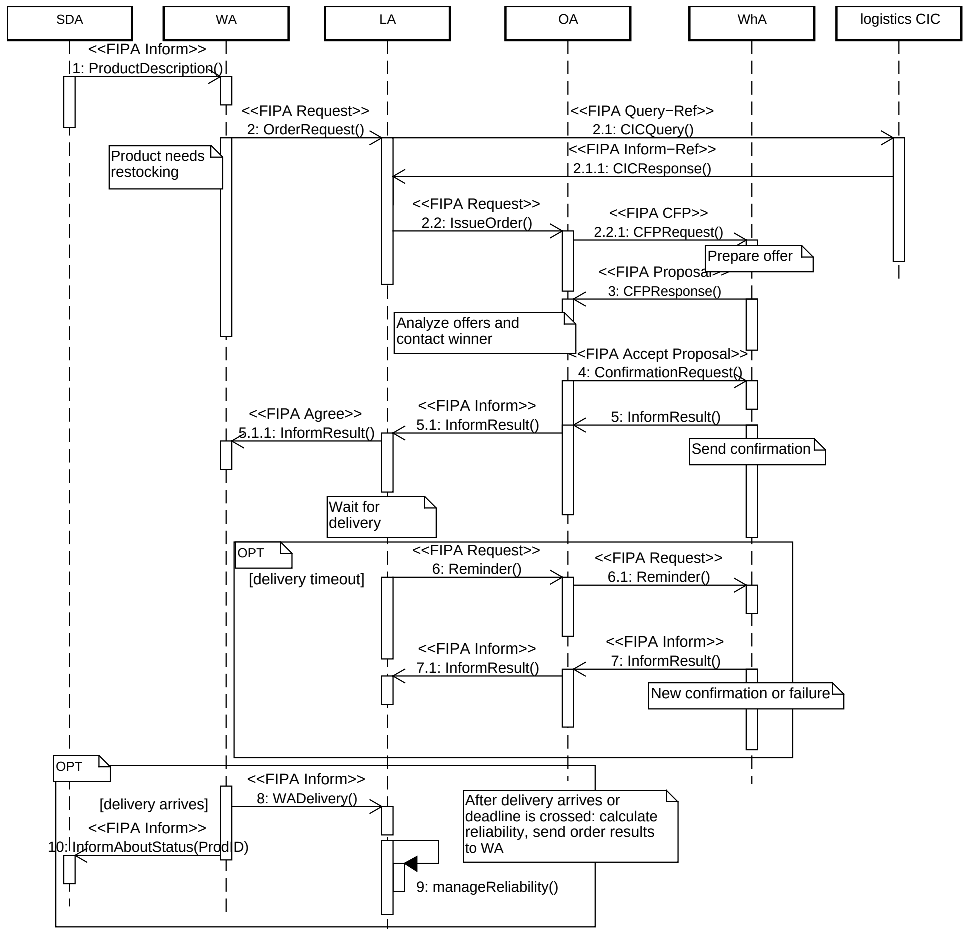

Figure 2. Restocking process: sequence diagram

When the non-empty list was received, the *LA* removes these suppliers that have their reliability value below a certain threshold. Then the *LAAgentDescriptions* list is formed by supplementing each *CICAgentDescription* received from the *logistics CIC* with the reliability information. If a given *WhA*'s is not known a default trust value is used.

After preparing the list, the *LA* utilizes the *FIPA Request Protocol* to find a free *OA*. Busy agents will respond with *FIPA Refuse* messages. If all agents respond in such a way, this process may need to be repeated until a free *OA* is found and responds with the *FIPA Agree* message. The *LA* then sends the ordering request to the selected *OA* and awaits for the result of the ordering process. *LA*'s message contains the *IssueOrder* action with the *OrderDescription* and the *LAAgentDescriptions*.

After obtaining the request from the *LA*, the *OA* engages in the *FIPA ContractNet Protocol* interactions with *WhA*s from the list. It sends the *FIPA CallForProposal* message, containing *CFPRequest* with *OrderDescription* to the *WhA*s. *WhA*s evaluate the CFP and submit their offers by sending *FIPA Propose* messages containing the *CFPResponse* action with *OfferDescription* or, if terms contained in the CFP are unacceptable/not interesting, respond using the *FIPA Refuse* message.

Responses must arrive within a *timeframe* specified by the *OA*, after which the *OA* proceeds to evaluate them. First, it filters unacceptable offers. Note that it is possible that some *WhA*s may respond knowingly with proposals that violate some of the conditions and in special circumstances—when no better offers were found—the *OA* may need to accept such offers. In the next step, offers are ranked and the winner is determined. Winner is sent a *FIPA AcceptProposal* message containing the *ConfirmationRequest* action with its offer quoted. The winning *WhA* must in turn reply with the *FIPA Inform* message containing *ConfirmationResponse* action with the *OrderConfirmation* which has unique *orderID* generated by the supplier. This successfully completes the ordering process. The winner can also withdraw the offer by sending a *FIPA Failure* message. In this case, runner-ups are contacted in an iterative manner. In case when there are no more offers left or there were no offers to begin with, the *OA* sends a *FIPA Failure* message to the *LA*, which, in turn, forwards it to the *WA*. When the winner confirms the order, the *OA* sends to the *LA* a *FIPA Inform* message containing the *InformResult* action with the *WhA*-received *OrderConfirmation*, thus completing the protocol. At this time the *LA* sends information to the *WA*, inside a message of the *FIPA Agree* type. This performative is used in compliance with the protocol to indicate that the *LA* is performing the desired task (ordering), but its efforts do not guarantee success (ordering success ≠ order success), and thus sending the final response (*FIPA Inform*) is inappropriate at this stage. Meanwhile the *OA* returns to the pool of available *Ordering Agent*s.

Now the purchase enters the *delivery monitoring* stage. Here, the *LA* waits for the delivery from the *WhA* to be registered with the *WA*. When a delivery arrives the *WA* sends (to the *LA*) a plain *FIPA Inform* message containing the *WADelivery* action with the *DeliveryDescription*, which has supplier's *AgentID* and the already mentioned *orderId*. The *LA* does not need to respond to this message, but it checks the messages to see if it is currently awaiting a delivery with the given *orderId* coming from a supplier *AgentID*. If it finds a match, the ordering process is completed. As a result, the reliability value of supplied *AgentID* is increased. If a delivery notification does not come within time agreed in the *OrderConfirm*, actions must be undertaken (recall, that receiving supplies is vital to the e-Shop as its warehouse is likely to run out of stock Those actions are: (1) retry the ordering (sending reminder to the *WhA* / choosing new *WhA*), if there is still time, and (2) marking that a retry has been made.

If there is still time before the deadline (established by the *WA*), then order can be retried. If it is the first time an attempt to retry the order is made, a reminder is sent to the *WhA*. To this end, *LA* contacts a free *OA* with a *FIPA Request Protocol* message with a *Reminder* action containing *AgentID* of the *WhA* and the *orderId*. The *OA* accepts the job (the *FIPA Agree*) and contacts the *WhA* (also using *FIPA Request* Protocol), sending it the exact same action. The *WhA* is expected to reply within a *timeframe* using either a *FIPA Failure* (offer is withdrawn) or a *FIPA Inform* providing new *OrderConfirmation* with a new delivery time, which is forwarded to the *LA* unchanged. In the case of an agreement, the *LA* returns to awaiting delivery, in the case of failure, the *LA* removes this *WhA* from the *LAAgentDescription* list and locates an *OA* to perform entirely new search for a supplier. New search is also ordered if a reminder to the supplier whose delivery we were waiting resulted in a failure. The monitoring stage ends when: (1) delivery is received, or (2) reminder to the supplier was made, but it was refused, while deadline has already passed. Note that we assume that the actual order failure occurs only when the delivery deadline has passed *and* the reminder failed. This is because it is possible that there is an order delay and goods may arrive late. This information can be obtained from the *WhA*, and thus the need for the reminder.

When the monitoring stage ends, the *WA* is notified about the result by the *FIPA Inform* or the *FIPA Failure* message to complete the *FIPA Request* protocol. The message will contain the *OrderRequest* action with the *OrderRequestResult* set appropriately. Furthermore, at this stage the reliability bonuses and/or penalties are calculated and applied. Finally, in the case of a successful order, the *WA* sends to the *SDA* a *FIPA Inform* message containing status information about the re-stocking of the warehouse.

6. Concluding remarks

In this note we have discussed the way in which the logistics subsystem is being introduced into our model agent-based e-commerce system. We have presented used UML's use case and sequence diagrams to formally depict and discuss the most important features of our approach. Due to the lack of space, we have focused our attention on agents and their interactions. The proposed system has been implemented and is in the final testing phase.

References

[1] Agentis Software, 2007. `http://www.agentissoftware.com/`.

[2] C. Bádicá, A. Báditá, M. Ganzha, M. Paprzycki, Developing a Model Agent-based E-commerce System. In: Jie Lu et. al. (eds.) E-Service Intelligence—Methodologies, Technologies and Applications, Springer, Berlin, 2007, 555–578

[3] C. A. Butler and J. T. Eanes. Software agent technology for large scale, real-time logistics decision support. *US Army Research Report ADA392670*, 2001. 23 pages.

[4] FIPA: Foundation for Physical Agents. `http://www.fipa.org`

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

[6] W. Ying and S. Dayong. Multi-agent framework for third party logistics in e-commercestar. *Expert Systems with Applications*, 29(2):431–436, August 2005.