Information flow and usage in an e-shop operating within an agent-based e-commerce system

Michał Drozdowicz, Maria Ganzha, Marcin Paprzycki,
Maciej Gawinecki, Alexander Legalov

August 3, 2008

Abstract

Utilization of software agents in e-commerce is a subject of a lot of interest. In our work we are developing a complete agent-based e-commerce system in which agents play all major roles representing both buyers and sellers. The aim of the paper is to describe flow and usage of information in a virtual e-shop operating within the proposed e-commerce system.

1 Introduction

Currently, we are developing a model agent-based e-commerce system. This system varies from other work found in the literature at least in the following ways:

1. Typically, only price negotiation of a single item (or collection of items treated as subject of a "single transaction") is contemplated (even though the negotiation itself may follow a very complicated set of rules; e.g. two stage negotiation found in [22]). Once the negotiation is over, agents that participated in it complete their work and the process ends. We are interested in a more realistic scenario when a number of items of a given product are placed for sale one after another, e.g. 90 Canon EOS cameras are to be sold by an e-store.

2. Since a collection (sequence) of items is sold we treat price negotiations as a "discrete process" in which buyers are "collected" and released together in a group to participate in a price negotiation. While the negotiation takes place buyer(s) are allowed to communicate only with seller(s). At the same time the next group of buyers is collected (as they arrive) for the next negotiation. This process is similar to such forms of real-life auctions where auctioneers gather in a room and stay in it until the auction ends.

3. Since multiple subsequent price negotiations (involving the same product; e.g. Canon EOS cameras) take place, price negotiation mechanisms can be dynamically changed. For instance, first 55 items may be sold using English Auction, next 22 using iterative bargaining, while the remaining 13 may be sold using fixed (bargain) price.

From this setup follows that we assume that shops in the system have to adapt to changing market conditions; and change of price negotiation mechanism is an example of such adaptation. Obviously, to achieve this goal they have to collect, store and later analyze information about various events taking place in the shop. In this context, the aim of this paper is to describe flow of information in the system, with particular attention paid to processes taking place within the e-shop. Note that we concentrate on that part of the system due to the fact that information flow within the client-side is much less involved and encompasses only three entities (*Buyer Agent*s, *Client Agent* and *Client Decision Agent*). Finally, we outline how collected data can be used in sales forecasting.

To this effect we proceed as follows. In the next section we summarize main features of the system. Next we follow with the description of the sources and flow of information within the system. Finally, we briefly illustrate how stored information can be utilized by the e-shop.

Note that this work is complementary to [10], where we describe in detail *how* information is to be efficiently stored within the system. Thus readers may want to consider that source for additional details.

2 System description

The system under construction is a model agent-based virtual marketplace, where agents representing *Buyers* engage in price negotiations with agents representing *Sellers*. Here, instead of focusing only on a single feature of e-commerce (e.g. price negotiations), which is often the case in the literature [4, 24, 1, 20, 22, 8, 21] we consider the complete processes that involves both the *Buyer* and the *Seller* (as well as *Wholesalers* that provide products to e-stores). Thus, we start from the moment when the *User-Client* expresses desire to purchase a product, and follow the chain of events until the purchase is made, or deemed impossible. Conceptualization of the proposed system has been represented as a use case diagram in Figure 1. Since the detailed description of the system can be found in [5, 15, 7, 23], and is out of scope of this paper, here, we only briefly describe pertinent processes taking place in the system (and entities participating in them).

2.1 Client subsystem

First, let us consider agents supporting the *User-Client* in her/his shopping needs. The *Client Agent* (*CA*) is responsible for the direct support of the *User-Client*. Here, one can envision the *Client Agent* as a role within, or a part of functionality of, a *Personal Agent* as conceptualized by P. Maes in [21].

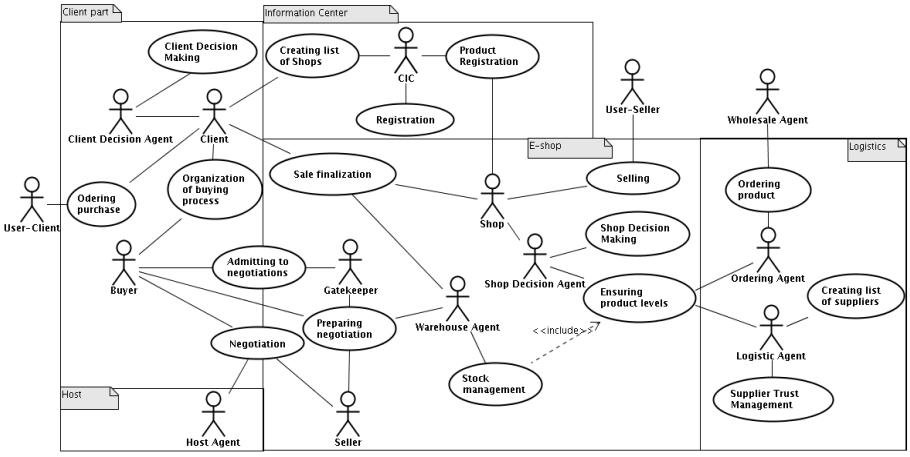

Figure 1: Use Case

When the *User-Client* interacts with its *CA* (s)he describes the product (s)he would like to buy, as well as conditions of purchase (e.g. the price and/or delivery time). In order to fulfill the order the *CA* obtains a list of shops to be contacted. This is done by querying a yellow-page (infomediary) agent called the *Client Information Center* (*CIC*; see, [24, 19, 9] for more details about the role and implementation of the *CIC*). Obtained list of shops is adjusted on the basis of trust considerations (see, [6, 12]). As a result, the *CA* interacts with the *Gatekeeper Agents* (*GA*), representing selected shops, to ensure that its *Buyer Agents* (*BA*) participate in price negotiations. After receiving offers from winning *Buyer Agents*, the *CA* communicates with the *Client Decision Agent* (*CDA*) to determine which offers are acceptable and, among them, which is "the best." Note, that offers arrive asynchronously and have varying expiration dates ([15]), thus the decision making within the *CDA* has to include not only product-features like price, but also reservation expiration and trust [12, 6]. If there exists a satisfactory offer and the *CDA* decides to accept it, the *CA* finalizes the purchase. If no appropriate offer can be found, the *CDA* decides course of further action, which may involve trying to negotiate for a better price, or informing the *User-Client* that purchase under specified conditions is impossible.

2.2 Shop subsystem

Activities that take place within the e-shop can be divided into four major parts:

1. selling products,
2. managing negotiation mechanism(s),
3. managing trust toward customers, and

4. managing product inventory.

- Selling products involves price negotiations and is handled by two agents:
  - the *Gatekeeper Agent* (*GA*) that facilitates entrance of *Buyer Agents* (by inviting agents to be sent in, or by creating them), registers incoming *BA*s (interested in purchasing products), and starts the negotiation; as well as
  - the *Seller Agent* (*SeA*) that represents the shop during the negotiation (for details of the negotiation process see, [5, 3]).
  - the *Host Agent* (*HA*) which supervises the negotiation and enforces protocol of the specific negotiation that parties are engaged in; it also collects data about the negotiation process (see below)

  Finalization of purchase as well as registering (and de-registering) products with the *CIC*, and coordinating work of other agents in the e-shop is the duty of the *Shop Agent* (*SA*), which should be viewed as the *Manager* of the e-store. In this role the *SA* is often passing messages between agents that do not know each other (one of the important features of agent systems is reduction of number of agents that know each-other directly to facilitate agents system maintenance; see [26]).

- Managing price negotiation mechanisms is performed by the *Shop Decision Agent* (*SDA*), and involves selecting the most appropriate negotiation mechanisms and their parameters, as well as strategies for *Seller Agents* representing the store. Furthermore, the *SDA* is responsible for adapting price negotiation mechanisms and their parameters to changing market conditions.

- Managing of trust toward customers is also one of duties of the *SDA*. Trust is the main parameter in (i) decision of admitting *Buyer Agents* to negotiations, and (ii) establishing time of product reservation (to minimize potential losses caused by unreliable or malicious customers; see, [6, 12]).

- Overseeing product stock levels is a task handled by a group of agents, supervised by the *Warehouse Agent* (*WA*) that stores and manages information about current inventory and about product reservations. Furthermore, the *WA* uses sales forecasts prepared by the *SDA* to proactively ensure adequate supply of all products. Supply orders are carried out by the *Logistic Agent* (*LA*) that interacts with a wholesaler-oriented equivalent of the *CIC* to receive a list of wholesalers that sell specific products and dispatches requests for product sale and delivery proposals using workers from a pool of *Ordering Agents* (*OA*). More details about the logistics subsystem can be found in [23, 13].

3 Gathering and utilizing information

Let us now focus our attention on processes involved in generating, managing and utilizing flow of information during the work of the system. Let us recall, that we are concerned primarily with processes taking place within the e-shop. Obviously, generation of information is a result of monitoring of the state of various entities within the system. Subsequently, generated information has to be delivered to the *SDA* for storage and processing. Monitoring and delivery of resulting data can be performed in several ways (see [10] for more details). In the process of selecting the right approach for our system, we have considered the fact that as far as the *SDA* is concerned, there are two sources of information:

1. Information originating from the *inside* of the e-shop—received either from/through the *SA* or the *WA* agents, describing specific events related to the functioning of the e-shop; e.g. buyer registration, negotiation closing, transaction finalization, restocking deliveries etc. These messages have to provide a complete picture of activities of individual agents within the system and thus should be promptly delivered to the *SDA*. Let us note that most messages passed within the e-shop are rather small in size (even though there may be a relatively large number of them; scaling with the size of activities in the shop), while the *SDA* is the only receiver of collected data. Therefore, for the time being, we have decided to utilize an active information source-based approach, i.e. change in the state of the system or occurrence of an event result in a message, containing necessary information about this event, sent to the monitoring module (the *SDA*).

2. Information originating *outside* of the e-shop—concerning number of shops selling a specific product, and possibly, the number of queries concerning specific products received from clients by the *CIC*. Since the *CIC* is not an active source of information, the *SDA* has to be. Therefore it requests periodically the needed data from the *CIC*.

Keeping this in mind we can now discuss where in the system information is generated and how it is used. Let us start from initial interactions with clients interested in making a purchase.

3.1 Before negotiations

3.1.1 *Gatekeeper Agent* interacting with *Client* and *Buyer* Agents

After the *User-Client* specifies its need, the *Client Agent* obtains from the *CIC* list of shops that sell given product and adjusts it on the basis of its trust in them (here, trust information provided by the *CDA* is utilized, see [6, 12]). As a result the *CA* interacts with representatives of selected shops—their *Gatekeeper Agents*. The aim of this interaction is first, to find out if the needed product is still available, and second, to establish if and how the *Buyer Agent* can become involved in price negotiations. Note that the *Buyer Agent* is a lightweight,

mobile agent that is delegated by the *Client Agent* and migrates to the e-shop to represent the *User-Client* in price negotiations. In the case when the *CA* is not able (not allowed by *GA*) to create its own *BA* to take part in price negotiations, a *BA* could be created by the shop in response to a client request. In both cases the *Client Agent* communicates with the *Gatekeeper Agent*. The interaction is summarized as a sequence diagram in Figure 2.

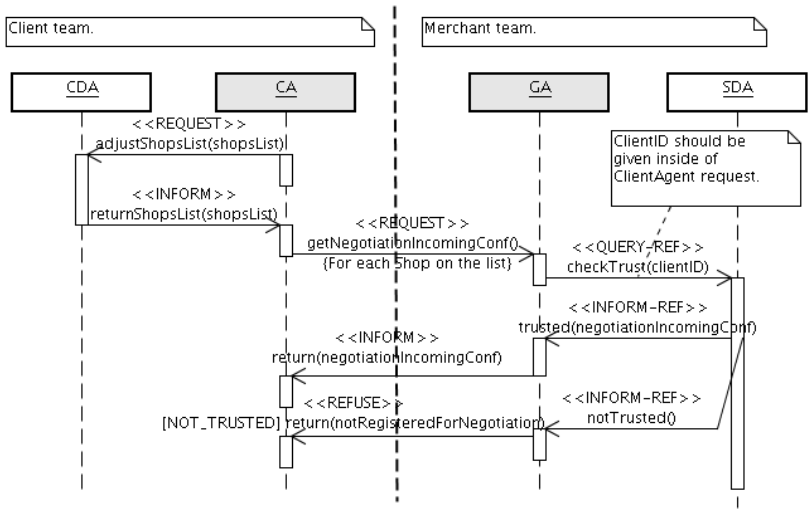

Figure 2: Sequence Diagram of trust checking process

Below, we present code preparing message sent by the *CA* to initiate interactions with the *GA* (process involves also checking hostility of the *GA*):

```
protected Vector prepareRequests(ACLMessage request) {
    request = new ACLMessage(ACLMessage.REQUEST);
    request.addReceiver(gatekeeper);
    request.setOntology(NegotiationOntology.NAME);
    request.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
    request.setLanguage(FIPANames.ContentLanguage.FIPA_SL2);

    CheckHostility check = new CheckHostility();
    Action act = new Action(gatekeeper, check);
    try {
         cm.fillContent(request, act);
        }
    catch (Exception e) {
         if (logger.isLoggable(Level.SEVERE))
         logger.log(Level.SEVERE, ''Cannot prepare request'', e);
         request = null;
        }
    return super.prepareRequests(request);
}
```

What we can see in this snippet is:

1. *CA* defines the type of communicative massage—`REQUEST`
2. *CA* defines a receiver of this message—`Gatekeeper`
3. the receiver can "understand" this message based on the "`Negotiation-Ontology`" ontology
4. *CA* also defines a protocol of this communication, namely the `FIPA_REQUEST` protocol
5. the content language is based on the `FIPA_SL2` specification
6. the request contains a `CheckHostility` action which denotes the operation of checking the trust level of a given e-shop

Upon being contacted by the *CA*, the *GA* checks if that *CA* is trustful. Since the *SDA* has access to history of interactions with a given *CA*, it can utilize it to determine if: (a) to reject *CA*s request to enter, (b) approve such request and allow the *CA* to send a *BA*, or (c) approve request under condition that the *BA* will be created locally (if such option is available). On the basis of assessment provided by the *SDA*, the *GA* sends a response to the *CA*. The following code snippet represents actions undertaken by the *GA*.

```
protected ACLMessage handleRequest(ACLMessage request)
   throws RefuseException, FailureException, NotUnderstoodException
{
   ACLMessage response;
   String team = helper.getSenderTeamId(request);
   if (team == null)
   throw new UnknownTeamException(null).fillACLMessage(cm, request);

          // Checking trust

   HostilityConfiguration hostility=buyerMgr.getHostility();
   response = request.createReply();
   response.setPerformative(ACLMessage.INFORM);

   Result r = new Result(requestAct, hostility);
   try {
            cm.fillContent(response, r);
       }
    catch (Exception e) {
        throw new FailureException(''Cannot prepare response'');
        }
    return response;
}
```

Here, we can see that the *Gatekeeper Agent* retrieves, from the request message, the *id* of the *Client*. Next, it acquires its trust value and on its basis prepares an `INFORM` response message by appropriately filling its contents. The decision specifies if a *Buyer* created by the *Client* can migrate to the shop platform (the `immigrantsAllowed` flag) and if the platform allows creation of buyers on behalf of the client (the `creatingBuyersAllowed` flag). In the future, such message will also contain information about the *life-timeout* of the *Buyer Agent*

(determined basis on trust). Life-timeout is the length of a period of time after which an inactive *BA* will be removed from the system.

Obviously, behavior of the *CA* depends on the received answer. First, let us note that information contained in the answer is forwarded by the *CA* to the *CDA* for storage and further processing. For instance, refusal of admission is an indicator of the level of trust used in the given shop and may be used in the future to evaluate incoming proposals (see also [6, 12]). If the *CA* is informed that it can send a *BA* to the store, it may do it (note that, based on trust considerations (see [6, 12]), the *CDA* may decide to not to send an agent to a store if a very large number of e-shops, selling a given product, is available. However, if the message informs that only locally prepared *BA* can be used, the *CA* will confirm (or not) that it wants to utilize this form of negotiation participation. Below you will find a snippet of code in which *CA* prepares a message that asks for the *BA* to be created by the *GA*:

```
protected Vector prepareRequests(ACLMessage request)
{
  request = new ACLMessage(ACLMessage.REQUEST);
  request.addReceiver(gatekeeper);
  request.setOntology(NegotiationOntology.NAME);
  request.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
  request.setLanguage(FIPANames.ContentLanguage.FIPA_SL2);

  CreateBuyer create = new CreateBuyer();
  Action act = new Action(gatekeeper, create);
  try {
        cm.fillContent(request, act);
      }
  catch (Exception e) {
       }
  return super.prepareRequests(request);
}
```

In the case when the *BA* is send, when it arrives at a shop, it communicates with the *GA* and the *GA* checks (again) trust towards this *BA*. This is necessary because multiple representatives of the *CA* can participate in negotiations (for different products) and their actions may negatively affect the trust toward their owner. If the trust check is not successful, the *BA* receives a message prepared as follows:

```
/**
 * Prepare REFUSE message and leave in {@link DataStore}
 *  object to be sent.
 * @param reason
 * is the reason to refuse admitting for a negotiation
 */
protected void prepareRefuseResponse(RefuseException reason)
{
 ACLMessage request =
              (ACLMessage) getDataStore().get(REQUEST_KEY);
  ACLMessage response = request.createReply();

  response.setPerformative(ACLMessage.REFUSE);
  try {
```

```
        cm.fillContent(response, reason);
    } catch (Exception e) {
        response.setPerformative(ACLMessage.FAILURE);
    }

    getDataStore().put(REPLY_KEY, response);
}
```

We can see here that the *GA* creates a new `REFUSE` message and fills it with a reason for refusal based on the parameter passed to the method.

If the *BA* receives a refusal to be admitted to the negotiation, it finishes its operation, thus removing itself from the shop's platform. At the same time, positive trust check results in *BA* being provided with the current *negotiation protocol* and *template*. The protocol module defines the general set of rules and flow of the process of a specific negotiation. It is common for all *BA*s and, therefore, can be passed to incoming (and created) *BA*(s) by the *Gatekeeper Agent*. Along with the protocol, *BA*s receive negotiation parameters (a negotiation template) that customize the protocol to the specific negotiation. The template contains values such as the starting price, bid increment, etc. (see [5, 3] for more details). At this moment both the incoming and locally created *BA*s are in the same "stage of development" and request from their correspondent *CA*s an appropriate negotiation strategy (matching the obtained protocol and parameters of the negotiation).

In the following snippet we see how the *BA* handles receiving the negotiation template from the *GA*.

```
private final ACLMessage
        handleProductInformationChanged(ACLMessage inform)
{
  ACLMessage request = inform.createReply();
  Result r;

  NegotiationTemplate template =
              (NegotiationTemplate) r.getValue();
  strategy = getStrategy(template);
  if (strategy != null) {
      request.setPerformative(ACLMessage.REQUEST);
      ConfirmReady confirm = new ConfirmReady(template.getId());
      Action act = new Action(receiver, confirm);
       try {
            cm.fillContent(request, act);
           }
       catch (Exception e) {
         logger.log(Level.SEVERE,''Cannot prepare request'',e);
         throw new SystemException(e);
           }

    if (logger.isLoggable(Level.FINE))
      logger.fine(''Confirming of readiness
       to negotiate about product <'' + globalProductId + ">");
       }
  else {
       if (logger.isLoggable(Level.FINE))
         logger.fine(''Cancelling admission process, product <''
```

```
                                + globalProductId + ''>'');
                    request.setPerformative(ACLMessage.CANCEL);
            }
            return request;
}
```

First, the *BA* extracts the `NegotiationTemplate` content object from the message, second, it retrieves a strategy for the template it received. Afterwards, it creates a confirmation response using a `REQUEST` message containing a `ConfirmReady` action and sends it back to the *GA*. In the case when the *CA* cannot prepare a strategy *BA* decides to cancel its admission to negotiation (the *CANCEL* message).

The *GA* informs the *SDA* (via the *SA*; as the *SDA* and the *GA* do not know each-other directly) about all incoming/created *BA*s. Appropriate message includes also information about the product that the *BA* was interested in. This information is stored in the data mart and can be used for demand prediction.

Note that when a product is asked for the first time (e.g. a *BA* arrives interested in an Olympus E-520 camera) then such product has to be reserved for a negotiation (to assure that negotiation can take place and after it is successful, there will be product available for sale). This is achieved by the *GA* communicating with the *Warehouse Agent*. Preparing a message to the *WA* is achieved as follows:

```
protected final Vector prepareRequests(ACLMessage request) {
  request = new ACLMessage(ACLMessage.REQUEST);
  request.addReceiver(warehouse);
  request.setLanguage(FIPANames.ContentLanguage.FIPA_SL0);
  request.setOntology(NegotiationOntology.NAME);
  request.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);

  ReserveProduct action = new ReserveProduct();
  action.setGlobalProductId(globalProductId);
  action.setReleaseTime(releaseTime);

  Action act = new Action(warehouse, action);

  try {
        cm.fillContent(request, act);
   }
  catch (Exception e) {
   logger.log(Logger.SEVERE,
                ''Problem while preparing request msg'',e);
        throw new SystemException(e);
   }
  if (logger.isLoggable(Level.FINE))
      logger.fine(''Request for product <''+
        globalProductId + ''> reservation prepared'');

  return super.prepareRequests(request);
}
```

Here, we see that the *GA* creates a new message of the `REQUEST` communicative type, specifies the receiver of the message to be the *Warehouse Agent*. The language of the content is set to `FIPA_SL0` and the ontology describing it is

the "NegotiationOntology" ontology. The sender specifies that the message is a part of a conversation adhering to the FIPA_REQUEST protocol. The message is filled with the ReserveProduct action with the GlobalProductId parameter set to the identifier of the product to be reserved and ReleaseTime set to the period of time long enough to perform the negotiation.

After receiving the message the *WA* checks if this product is available. If yes, the *WA* reserves the product and sends to the *GA* the Negotition Template and the Negotiation Protocol. It is also possible that the *WA* informs the *GA* that the product is not available (it was sold out in the meantime). For extended discussion of this and other cases, involving for example products temporarily unavailable, see [5].

3.1.2 *Seller* agent

Let us briefly note that a similar (modular) approach has been used in the case of the *Seller Agent* (*SeA*). First, the *SeA* receives the same protocol and negotiation template as other negotiation participants (since all of them are to participate in the same price negotiation). Second, a private strategy module that defines the way it should handle the negotiation, as well as other private data, such as the reserve price is provided. These modules are sent to the *SeA* by the *SDA* (via the *SA* and the *GA*, as the *SDA* does not know the *SeA* directly).

Overall, we can say that, in terms of receiving and consuming information, *BA*s and *SeA*s are the end destinations of strategy and template parameters, originating at the *CDA* (for client strategy), the *SDA* and the *WA* (for the template and shop strategy).

3.1.3 Gatekeeper Agent

As it was shown, the *Gatekeeper Agent* plays one of crucial roles in information management within the shop. Therefore, to further clarify its functions, in Figure 3 we present its use case. This figure can be treated as a partial summary of material presented thus far.

As we can see, the *Gatekeeper Agent*: (1) interacts with incoming *Buyer Agents*, and admits them to the negotiations (or rejects their attempt at entering the host), or interact with *Client Agents* and, on their request, creates *Buyer Agents* (or reject such requests), and provides admitted / created *Buyer Agents* with the protocol and the current negotiation template; (2) in appropriate moments releases selected *Buyer* and *Seller* agents to participate in price negotiations, and (3) manages updates of form (and specific details) of negotiations. To further formalize the description of the *Gatekeeper Agent* in Figure 4 we present its statechart diagram of activities related specifically to supporting negotiations. For additional details concerning actions of the *GA* see also [5, 11].

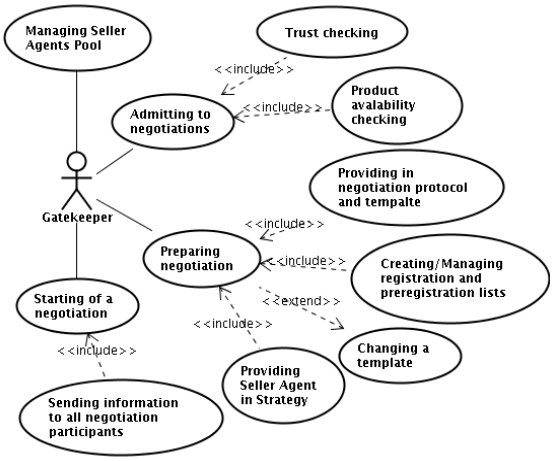

Figure 3: Use case diagram of the *Gatekeeper Agent*

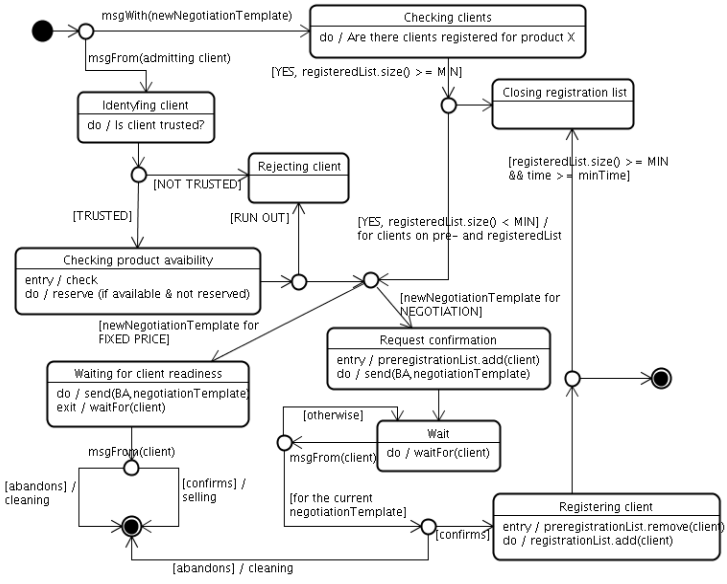

Figure 4: *Gatekeeper Agent* negotiation related activity—statechart diagram

3.2 Negotiation process

It should be obvious that during price negotiations *BA*(s) and/or *SeA*(s) post bids and as such are key generators of information about the negotiation process. As described in [14, 16], the negotiation process is managed by the *Host Agent*, which obtains from the *Seller Agent* a list of *Buyer Agents* that are to participate in negotiations and assures that they proceed according to the specified protocol (see also [4]). Since all information pertinent to a given negotiation is available to the *Host Agent* (e.g. posted on its blackboard, see [25]), once the negotiation is over it can pass it to the *GA* to be stored in, and utilized by, the e-shop. In [10] we have provided a complete description of information related to negotiations stored by the e-shop. A brief summary can be found in the next section and in section 4, where the list of selected tables from the data-mart is presented.

3.3 After a negotiation

One of the more "information rich" events happening in the e-shop is the end of a negotiation. At this moment, the *GA* obtains from the *Negotiation Host* information about just completed negotiation and sends it to the *SA*. The *SA*, in turn, forwards it to *SDA* for storing and processing. The negotiation description contains the following information:

1. The unique negotiation identifier—generated by the *WA* during the negotiation preparation (when product is reserved for negotiation; see above)
2. Identifier of the strategy passed to the *Seller Agent*—generated by the *SDA* when preparing the negotiation template and strategy.
3. Identifier of the template passed to the negotiation participants.
4. Date and time of the beginning and the end of the negotiation
5. The amount of products offered
6. The list of *BA*s (and thus *CA*s) registered for the negotiation—including those who received the negotiation template but rejected it or failed to accept it. For each *BA* the following information is also specified:
   (a) Date and time of providing the *BA* with the negotiation template.
   (b) Date and time of the *BA* accepting the template if it occurred.
   (c) A list of bids made by the client described by the time of the offer, amount offered to buy, the price and a flag specifying if the bid was a winning offer.

Note that, due to performance reasons we have decided to include in this message the list of bids (instead of recording them one by one). It is also important to stress that all information which the *Negotiation Host* forwards to the *GA* should be marked as **visible** in the negotiation protocol.

When the negotiation is successfully completed the *SDA* establishes length of reservation—time during which the negotiated conditions, such as the price and amount of offered product, hold (here, the *SA* sends to the *SDA* the `QUERY-REF` about reservation duration—`getReservationDuration(ClientID)`, and awaits an answer). The reservation length is to be calculated based on the level of trust towards the client represented by the *BA*. Currently, all *BA*s receive reservation of the same length.

The reservation can end in three ways: (1) with confirmation of the purchase—*SA* receives a message in which the *CA* confirms the purchase; (2) cancellation of the purchase—*SA* receives a message about cancellation of the purchase; or (3) expiration of reservation—*SA* did not receive the confirmation/cancellation message in time. Information about expiration of the reservation the *SA* receives from the *WA*. The *SA* forwards to the *SDA* information about (either) end of reservation, to create/update the trust profile of a given *Client*. The message describing the finalization of the transaction contains the following information:

1. Identifier of the transaction
2. Transaction outcome i.e. confirmation, cancellation or expiration
3. Date and time of the event

4 Information storing

The *Shop Decision Agent* is the decision-making entity within the shop subsystem. As already mentioned, its main aims are: (1) management of trust toward individual customers, including the decision whether to admit (and possibly in what way) their representatives to negotiations; (2) setting the product reservation deadline; (3) managing forms and parameters of price negotiations; and, (4) preparing sales forecasts for the automatic stock management. To fulfill these goals, the *SDA* gathers, stores and processes data and knowledge generated within the e-store.

Note that the information flow in the system has been designed to allow the *SDA* to gather the most detailed information about events occurring in the shop and "remember" their complete history. While this enables the *SDA* to make informed decisions, it also poses the danger of running into a situation when the amount of collected data makes it impossible to effectively process it. Therefore, we have decided to store data in a multi-dimensional data mart built according to the star / fact constellation schema proposed in [18]. The detailed description of this solution can be found in [10], in the following sections we only summarize selected tables that the schema consists of.

4.1 Bid Fact Table

The *Bid Fact Table* contains information about bids made during negotiations. Each row in the table depicts a single bid and holds information about amount

of the bid along with links to the dimensions of: the client who made the bid, the product, the date and time of the offer, the negotiation template and the ID of strategy used in the negotiation and the bid status, i.e. whether the bid was above or below the negotiation minimum price and if it was a winning offer.

4.2 Transaction Fact Table

The *Transaction Fact Table* is the largest fact table in the schema and contains information about the complete sale process flow, for every transaction in the system (with the exception of the bidding process persisted in the *Bid Fact Table*). This table is built on the basis of the *Accumulating Snapshot* pattern ([18]) and describes transactions using series of boolean flags as well as date and time dimension links that answer questions if and when the transaction passed consecutive steps of the sales process.

4.3 Negotiation Fact Table

Every row in the *Negotiation Fact Table* holds data describing a single negotiation such as the negotiation end date and time, product, used strategy and template and the following metrics: the amount of units offered, the starting and minimum prices, the quantity and value of the items reserved in winning bids, quantity and value of the items actually purchased, the total number of bids, the number of winning bids and the number of finished transactions.

4.4 Supply Fact Table

The *Supply Fact Table* accumulates data about wholesale orders and deliveries of stock to the shop's warehouse: the amount of ordered and delivered product organized along the following dimensions: the id of the wholesaler that carried out the order, the ordered product and references to the date and time dimensions describing the order and delivery events.

4.5 Inventory Snapshot Fact Table

The *Inventory Snapshot Fact Table* is a helper table aggregating information about stock levels of products at daily granularity. This data is derived from the *Negotiation* and *Supply Fact* tables to make accessing stock level information easier and more efficient.

5 Information processing

Let us now discuss utilization of data stored by the e-shop. To start, in Table 1, we summarize key areas in which we envision that information collected by the e-shop can be used. This list is, obviously, not a comprehensive one. Rather, it is presented to indicate potential usefulness of the proposed data management approach.

Expected result	Method used
Length of the sales forecasting period	Estimating the longest period for which the sales amount time series is stationary using the P-value test
Forecasted sales amount	Double/triple exponential smoothing of the sales amount time series
Estimated margin of the forecast	Mean absolute deviation between the initialized forecasting model and historical data
Amount of units offered	Moving average of maximum bid amount of units across past negotiations
Evaluating the product price	Derivative following method—adjusting the price according to the result of previous change

Table 1: Data mining aims and methods

Using data stored in the data mart we have conceptualized and implemented, within the *SDA*, a few simple decision-making processes related to determining negotiation parameters and forecasting the sales volume. It should be noted that our goal was *not* to implement the complete *SDA* functionality—we have not yet, for example, approached the challenge of managing trust. It was also not our aim to perform a comparative analysis of possible solutions to the implemented functionalities in search of an optimal method or algorithm. What we set out to do was to show that the data gathered and stored by the *SDA* can be transformed into useful knowledge to be used by other agents in the system—in our case the *GA* and the *WA*. Let us start with a description of proposed approaches to knowledge extraction.

5.1 Setting negotiation parameters

The outcome of the process of determining the shop's negotiation parameters is the type of negotiation protocol to be used for the sale of specific product, the strategy for the *SeA*s to follow and a vector of parameters for both the strategy and protocol modules. The set of parameters may differ from one type of negotiation procedure to another and at this point we have decided to limit our scope to a simple multi-item English Auction without a reserved price and unlimited bid step. In this case, the only parameters the *SDA* needs to evaluate are the starting unit price of the product and the desired amount of product to be offered in a single negotiation.

5.1.1 Evaluating the price

The obvious goal of modifying the price of products is to maximize the shop's profit. A very interesting comparison of a few methods of automatic price evaluation can be found in [17]. We have decided to incorporate one of methods described there, namely the *Derivative Following Algorithm*. According to [17], this method should perform reasonably well, especially in an environment where all agents use the same strategy. It consists in setting the future price of the product based on the change of profit that occurred after the previous price

modification. Initially, the price is selected randomly and after some period of time, it is increased or decreased by some value. The decision whether to increase or decrease the price is random at first, then it is determined by the result of comparing the change in profit between past two periods. If the profit has increased due to the previous price change, then the price is modified in the same direction. If the profit has decreased, the direction of price change is reversed. With every period, the price modifier is exponentially decreased according to the following equation:

$$\delta_n = \frac{\delta_n(n_0 + 1.0)}{n_0 + currentPeriod} \tag{1}$$

where $n_0 = currentPeriod/10$.

A slight modification of the method was introduced to take into account also the number of clients registered in the shop looking for the specific product. The modification was meant to handle cases where the drop in profit is caused by the overall drop in demand for the product. Since the process of setting the price was based on information about the value of products sold, past prices of the product and the number of clients registered as looking for it; this information comes from the *Sales* and *Demand* fact tables (see [10]).

5.1.2 Setting the number of offered units

The reason why setting this parameters in multi-item auctions is important is because the items offered at a negotiation are reserved for the time of the auction—they cannot be offered at a different one. This is a reason why the shop should not offer too many items at a time. On the other hand, setting too small a value can result in losing clients who want to buy a greater number of items—being forced into taking part in consecutive negotiations may not be acceptable. We have, therefore, decided to set the amount offered to the most probable amount of a single bid.

To determine how many items a single customer may bid, the moving average method has been used, taking into account the maximum amount offered in a single bid in the course of past negotiations. The data has been taken from the *Sales Fact Table*.

5.2 Sales forecasting

Forecasting product sales has been realized by analyzing the time series of the amount of sales, with the period length equal to the prediction horizon. The conditions determining the choice of a method of analysis was the possible existence of both trend and seasonal fluctuations of the data. Therefore, we have decided to use the exponential smoothing model ([2]).

The possible existence of seasonality in the data determined the use of triple exponential smoothing. The problem with this method, however, is the amount of data needed to estimate seasonal changes is large—it needs two complete cycles of the data to initialize these values. This posed a problem in cases

when the system has not yet accumulated enough sales data. To overcome this challenge we have introduced several countermeasures. Firstly, the length of a cycle of data was variable—seasonality can be analyzed on a weekly, monthly, quarterly, half-yearly or yearly level depending on the amount of available data. Secondly, if we have insufficient data to even perform a weekly seasonal analysis, the forecasting method is automatically switched from triple to double exponential smoothing, which does not take seasonality into account and hence does not pose any data amount requirements.

The forecast deviation, which is another parameter of the prediction message sent to the *WA* by the *SDA* is simply calculated as the the mean absolute deviation calculated basing on the initialized model and the historical data used in the process.

The forecasting process uses the data about the amount of items sold (i.e. the reservations of which have been confirmed by the clients and the sales process finalized) taken from the *Sales Fact Table*.

6 Sample information utilization

6.1 Test setup

We have performed several tests to establish usability of data processing methods described in previous sections. To be able to accurately describe the input conditions of the test we have developed a simple testbed, consisting of two agents "stubbing out" the parts of the system that interact with the *SDA*:

1. the *Shop Agent Stub*—engaged in the same communication protocols as the *SA*, and

2. the *Warehouse Agent Stub*—a mock agent of the *WA*.

Thanks to this approach we were able to feed the *SDA* agent with exactly the information we wanted output for, without the need of setting up complex relations between other agents in the system—especially multiple clients. We could also more easily simulate compressed flow of time.

To simulate the amount of items bought by clients we have incorporated a very simplistic economic model. We have assumed that each client has a maximum price at which he/she will buy the product. If the shop's price is higher, than the client will simply resign. We have also assumed that in each negotiation only one buyer can take part. With these assumptions all negotiations consist of a single bid if the price is fine for the client and no bid if it is too high. The client's maximum price is calculated according to the normal distribution with the mean value taken from the scenario definition file and possibly changing every day and the variance being a certain percent of the mean value (the percentage constant across the experiment).

6.2 Test scenario

In the test scenario used to check the *SDA*'s ability to adapt to changing conditions we have made the following assumptions:

- The length of the simulation is 1200 days.
- The length of the forecast horizon is 3 days, so the total number of forecasting periods is 400.
- The shop under consideration is the only one selling the product (there is no competition).
- Clients always buy 10 items of the product if such quantity is sold by the shop.
- The mean price every client is prepared to pay for the product is generated based on a normal distribution, with the mean of 5 and the variance 0.5. The relative variance of the maximum price has been set to 0.2.
- The number of clients visiting the shop for the product has been generated for each day according to the normal distribution with the mean rising linearly from 30 at the beginning of the simulation to 60 at the end, with a relative variance of the mean value of 0.05.

In the figure 5 we depict the number of clients registering at the shop, while figure 6 shows the results of a sample test of predicting sales.

In this scenario we have tested the sales forecasting module. Its task was made difficult not only by the rising trend of the input data and this data's deviance but also by additional deviance caused by the functioning of the price modification module. Despite those problems we have managed to prevent stock shortages in 78% of periods with a mean amount of overstocked product equal to 20% during overstocked periods. During the understocked periods, the mean amount of the items not sold due to the product shortage was 10% of the sold quantity with the median of 8%.

7 Concluding remarks

In the paper we have presented a comprehensive picture of information management within the e-store part of an agent-based e-commerce system. First, we introduced scenarios in which data elements are generated. Next we have discussed flow of messages involved in leading to storing this data in the e-shop's knowledge base, as well as utilization of knowledge extracted from it. Finally, samples of utilization of data stored in the central repository were presented and a specific application experimentally evaluated through simulation. Currently the application is further tested. In the next step we plan to start developing a larger portfolio of data mining methods to be used within the context presented here.

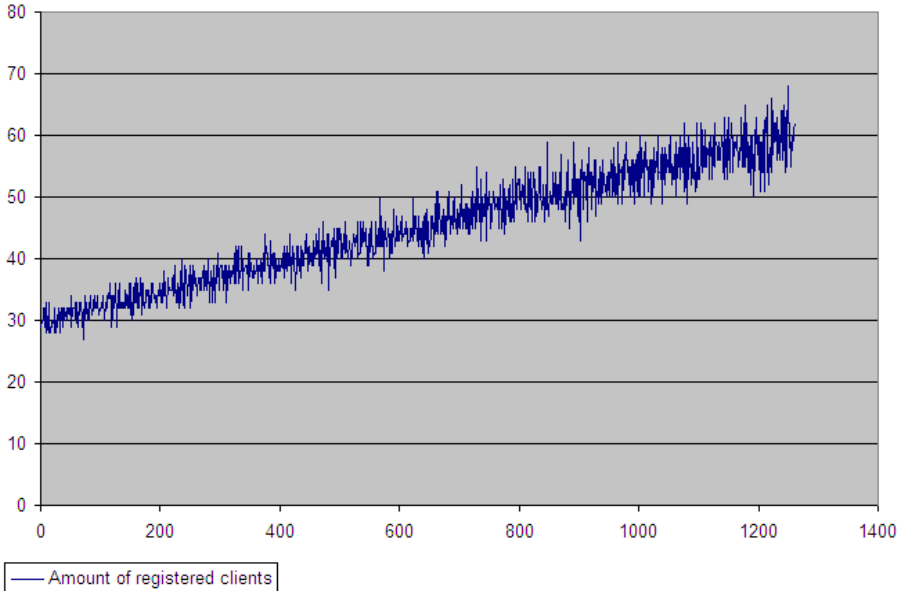

Figure 5: Test input data

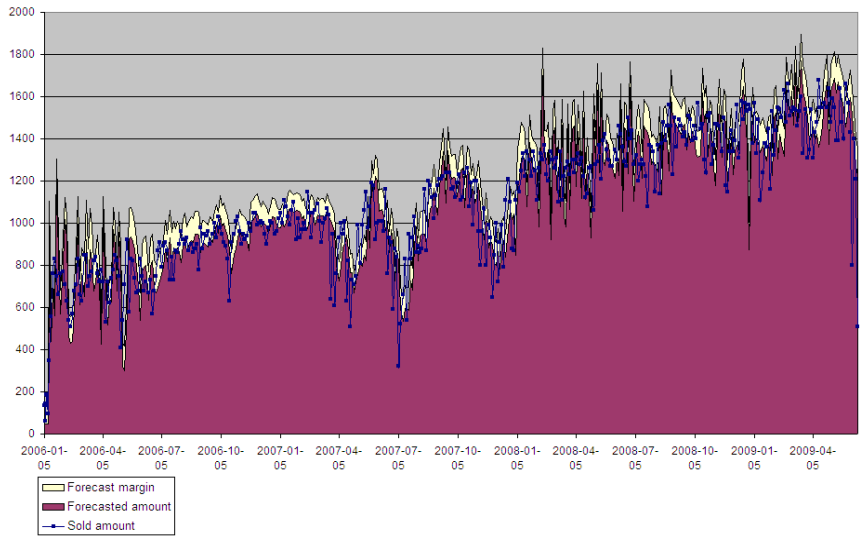

Figure 6: Results of the test

References

[1] Agorics. http://www.agorics.com/Library/Auctions/.

[2] Nistsematech e-handbook of statistical methods. http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc43.htm.

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

[4] P. C. J. N. Bartolini C. A software framework for automated negotiation. In *Proceedings of SELMAS'2004*, pages 213–235. LNCS 3390, Springer Verlag, 2005.

[5] C. Bădică, A. Bădită, M. Ganzha, and M. Paprzycki. *Developing a Model Agent-based E-commerce System*, pages 555–578. E-Service Intelligence—Methodologies, Technologies and Applications. Springer, 2007.

[6] C. Bădică, M. Ganzha, M. Gawinecki, P. Kobzdej, and M. Paprzycki. Towards trust management in an agent-based e-commerce system—-initial considerations. In A. Zgrzywa et al., editors, *Proceedings of the MISSI 2006 Conference*, pages 225–236, Wrocław, Poland, 2006. Wrocław University of Technology Press.

[7] C. Bădică, M. Ganzha, M. Gawinecki, P. Kobzdej, M. Paprzycki, M. Scafes, and G.-G. Popa. Managing information and time flow in an agent-based e-commerce system. In D. P. et. al., editor, *Proceedings of the Fifth International Symposiom on Parallel and Distributed Computing*, pages 352–359. IEEE CS Press, Los Alamitos, CA, 2006.

[8] V. Chavez and P. Maes. 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.

[9] M. Dominiak, W. Kuranowski, M. Gawinecki, M. Ganzha, and M. Paprzycki. Efficient matchmaking in an agent-based grid resource brokering system. In M. G. et. al., editor, *Proceedings of the International Multiconference on Computer Science and Information Technology*, pages 327–335, 2006.

[10] M. Drozdowicz, M. Ganzha, M. Gawinecki, P. Kobzdej, and M. Paprzycki. Designing and implementing data mart for an agent-based e-commerce system. *IADIS International Journal on WWW/INTERNET*, 2008. (to appear).

[11] M. Ganzha, M. Gawinecki, P. Kobzdej, and M. Paprzycki. Model agent-based ecommerce system. In S. et. al., editor, *Development of Multi-Agent Systems in Socio-Economic Environments*. Placet, Warsaw, Poland, 2008. in Polish.

[12] M. Ganzha, M. Gawinecki, P. Kobzdej, M. Paprzycki, and C. Bădică. Functionalizing trust in a model agent based e-commerce system. In M. B. et. al., editor, *Proceedings of the 2006 Information Society Multiconference*, pages 22–26. Josef Stefan Institute Press, 2006.

[13] M. Ganzha, M. Gawinecki, P. Kobzdej, M. Paprzycki, and T. Serzysko. Implementing commodity flow in an agent-based model e-commerce system. In *Parallel Processing and Applied Mathematics*, LNCS, pages 400–408.

[14] M. Ganzha and M. Paprzycki. Adapting price negotiations to an e-commerce system scenario. In K. Saeed et al., editors, *Proceedings of the CISIM Conference*, pages 380–386, Los Alamitos, CA, 2007. IEEE CS Press.

[15] M. Gawinecki, M. Ganzha, P. Kobzdej, M. Paprzycki, C. Bădică, M. Scafes, and G.-G. Popa. Managing information and time flow in an agent-based e-commerce system. In D. Petcu et al., editors, *Proceedings of the 5th International Symposiom on Parallel and Distributed Computing*, pages 352–359, Los Alamitos, CA, 2006. IEEE Press.

[16] M. Gawinecki, P. Kobzdej, M. Ganzha, and M. Paprzycki. Introducing interaction-based auctions into a model agent-based e-commerce system—preliminary considerations. In R. do Nascimento et al., editors, *Proceedings of the EATIS Conference*, ACM Digital Library, New York, NY, 2007. ACM Press.

[17] J. Kephart, J. Hanson, and A. Greenwald. Dynamic pricing by software agents. *Computer Networks*, 32:731–752, 2000.

[18] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling,* 2^{nd} *Edition.* John Wiley & Sons, 2002.

[19] W. Kuranowski, M. Ganzha, M. Paprzycki, and M. Dominiak. In S. et. al., editor, *Development of Multi-Agent Systems in Socio-Economic Environments*, chapter Software Agents as Resource Brokers in the Grid, pages 369–391. Placet, Warsaw, Poland, 2008. in Polish.

[20] K. Laudon and C. Traver. *E-commerce. business. technology. society (*2^{nd} *ed.).* Pearson Addison-Wesley, 2004.

[21] P. Maes, R. Guttman, and A. Moukas. Agents that buy and sell: Transforming commerce as we know it. 42(3):81–91, 1999.

[22] D. Rolli and A. Eberhart. A descriptive auction language. *Electronic Markets – The International Journal*, 2005.

[23] T. Serzysko, M. Gawinecki, P. Kobzdej, M. Ganzha, and M. Paprzycki. Introducing commodity flow to an agent-based model e-commerce system. In *Proceedings of the 2007 IAT Conference*, 2007.

[24] D. Trastour, C. Bartolini, and C. Preist. Semantic web support for the business-to-business e-commerce lifecycle. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 89–98, New York, NY, USA, 2002. ACM Press. `http://acm.org/10.1145/511446.511458`.

[25] K. Wasilewska, M. Gawinecki, M. Paprzycki, M. Ganzha, and P. Kobzdej. Optimizing blackboard implementation of agent-conducted auctions. *IADIS International Journal on WWW/INTERNET*, 2008. (to appear).

[26] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2002.