Intelligent Agents for an Artificial Market System

Nikos Karacapilidis
Industrial Management Lab, MEAD
University of Patras
26500 Rion Patras, Greece
+30 61 997257
nikos@mech.upatras.gr

Pavlos Moraïtis
Decision Support Systems Lab
Technical University of Crete
73100 Chania, Crete, Greece
+30 821 37347
moraitis@ergasya.tuc.gr

ABSTRACT
This paper describes an agent-based artificial market system whose underlying interaction protocols provide advanced features. Using the system, actors (i.e., customers and merchants) can delegate a variety of tasks to personal intelligent agents that act as their artificial employees. Contrary to other approaches, where a new agent is launched when their associated actors intend to perform a buying or selling transaction and "lives" only while this transaction is processed, our approach builds on a personalization of agents that permanently "live" in the market representing their actors' interests. Beyond just requesting and proposing an offer, agents in our system maintain a profile of their owners, which is updated upon the actor-agent interaction type. Furthermore, they can proactively ask their owners' permission to initiate a transaction (e.g., when a new product, which match one's profile, appears in the market). The system is also enabled with a highly interactive multiple criteria decision making tool that can handle ill-structured information during a purchase transaction, and perform a progressive synthesis and comparative evaluation of the existing proposals.

Keywords
Artificial market systems, e-commerce, multi-agent communication and collaboration, human-agent collaboration.

1. INTRODUCTION
It is broadly admitted that characteristics of software agents such as autonomy, proactiveness and "intelligence", together with their ability to cooperate, make them suitable for the delegation of traditional commercial transactions [4, 13]. Research work in agent-mediated electronic commerce has dealt with a diversity of tasks involved in buying and selling goods and services in an electronic market (e-market), while there is already a plethora of systems automating tasks such as product brokering, merchant brokering and negotiation [3].

This paper describes an agent-mediated artificial market system whose underlying interaction protocols provide advanced features. Its overall framework is not based on pre-classified ads; instead, the systems' agents collaborate in real-time mode. Using the system, actors (i.e., customers and merchants) delegate various tasks to their personal intelligent agents, which act as *artificial employees*. Contrary to the majority of the already implemented systems, the one presented here addresses efficiently many important issues.

AGENTS'01, May 28-June 1, 2001, Montréal, Quebec, Canada.

More specifically, our approach builds on the features of proactiveness and semi-autonomy of all software agents involved. Agents can take the initiative to contact their actors in order to start a transaction that seems "interesting" to them (e.g., when a new product, which matches one's profile, appears in the market), or trigger an actor's action (e.g., they can inform their merchant that a specific offer is of no interest in the market for the last month). We argue that semi-autonomy of agents assures the right level of control for the actions they could take; a fully autonomous agent could cause problems in such environments.

Second, our framework is based on a long (or even permanent) existence of agents in the e-market. In other words, agents do not "live" only during a specific transaction but much longer, upon the subscription paid by their owners at the time they were launched (i.e., an actor may "hire" an agent for a month, a year, etc.). This is highly associated with the personalization of the agents involved, through the maintenance of each actor's *profile*. For instance, a customer's agent can be assigned with a number of general interests (e.g., classical music, cruises) and preferences (e.g., one may dislike the color black on any product) of its actor, which can be enriched with more detailed ones each time the customer initiates a transaction, takes a decision to buy a certain product from a certain supplier, etc.

Third, our system enables the e-market's seller agents to refine (some of) a customer's purchase criteria during a transaction, argue in favor or against them, or even bring up new information to persuade him/her to accept their offers.

Finally, the approach proposed here is able to handle incomplete, inconsistent and conflicting information during a purchase transaction, and perform a progressive synthesis and comparative evaluation (across a set of attributes) of the existing proposals. This is performed through the use of a highly interactive tool, based on multiple criteria decision theory, which enables customers easily examine alternative scenarios (by selecting which of the proposals' attributes to be taken into account) and recommends the best solution according to the information at hand.

The remainder of the paper is structured as follows: Section 2 illustrates the architecture of the system's agents involved. Section

3 describes the set of interactions taking place in the artificial market, by analyzing activities and messages passed. Section 4 focuses on the multiple criteria decision making process for a purchase transaction. Finally, Section 5 comments on related work and outlines application building details and future work directions. It is made clear at this point that issues such as ordering, security, payment and delivery, while equally important in commercial transactions, do not fall in the scope of this paper.

2. AGENTS ARCHITECTURE

The development of the software agents proposed in our system is based on a generic and reusable architecture, conceived after examining the pros and cons of existing approaches (see, for instance, [10, 12, 16]). Even if the two agent types involved do not have the same functionality, they are built on the same basic architecture principles (see Figures 1 and 2); their constituent modules are tailored, according to their specific type (e.g., the decision making module of a seller agent is usually simpler than that of a purchaser agent). The architecture of each agent type is described in detail below.

2.1 The purchaser agent

A purchaser agent is composed of three modules (namely, the *communication, coordination* and *decision making* modules), which run concurrently and intercommunicate by exchanging internal (i.e., *intra-agent*) messages. A purchaser agent remains idle while no messages arrive at its communication module. As soon as a message arrives, the communication module (after transforming it to an intra-agent message) sends it to the coordination module using a message queuing mechanism. All modules adopt this behavior and remain idle while no messages to be processed are available. The same intra-agent queuing mechanism facilitates all three modules.

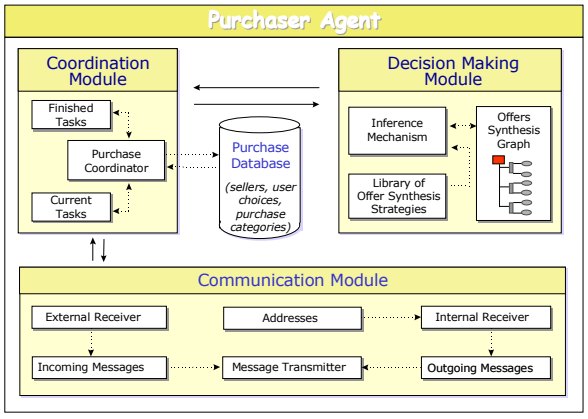

Figure 1: Architecture of a purchaser agent.

The *communication module* of a purchaser agent is responsible for the agent's interaction with its environment, that is the seller agents and the human user it assists. It sends and receives messages, while internally interacts with the *coordination module*. Its functionality is as follows: an *internal receiver* transforms each internally queued message (produced by the coordination module) to an inter-agent message and, in the sequel, stores it to the *outgoing messages* queue. In the case of selective communication, it also adds the receiving agents' addresses. An *external receiver* handles the opposite case, by transforming each external message received to an internal one and adding it to the *incoming messages* queue. Finally, a *message transmitter* monitors the incoming and outgoing queues, sending the queued messages to its coordination module, to another agent or to its human user, accordingly.

The *coordination module* handles the parts of the cooperation protocol that concern any type of interaction between (i) the purchaser and the seller agents, and (ii) the purchaser agent and the customer it assists. The related message types (presented in the next section) pass through the *purchase coordinator* component. In addition, the coordination module keeps track of the agent's *finished tasks* (that is, the *purchase history*) and the tasks being currently processed (e.g., a purchase evaluation for a specific product can be momentarily suspended due to searching for supplementary information). As shown in Figure 1, the module interacts with both the communication and the decision making modules. For instance, each time the decision making module needs to interact with the customer, it first sends a message to the coordination module which, in turn, attaches additional information (if required) and forwards it to the communication module. Similarly, the coordination module may filter the content of a received message before forwarding the related data to the decision making module.

In many cases, the purchaser agent has to access its *purchase database*, which contains all necessary information about the sellers (e.g., the products each seller agent provides), user choices (e.g., criteria, features, preferences and constrains for products the customer is interested in) and finally, purchase categories (the mSQL relational database is used). Retrieving the appropriate data, a purchaser agent is aware of which sellers it can buy a product from, the products each seller agent provides, the customer choices about a specific product, and the history of the buying transactions made so far. Moreover, an update of such a purchaser agent's database can occur asynchronously, in that the customer may add, remove or refine items of the corresponding lists at any time, independently of the current transaction.

Finally, the *decision making module* is composed of three components, namely an *inference mechanism*, a *library of offer synthesis strategies*, and the *offers synthesis graph*. It actually deploys the agent's reasoning mechanism that: (i) implements the behavior of the agent by using appropriate rules; for instance, the agent acts proactively upon the reception of some messages, sent by seller agents (see Section 3), and (ii) performs a synthesis and a comparative evaluation of the offers proposed by the seller agents; this process ultimately aims at finding the best offer (to be then recommended to the customer), according to the customer's choices and the information at hand. The inference mechanism is supplied with the necessary knowledge to perform the above tasks.

Note that a *strategy* encapsulates the appropriate information in order for the agent to perform the above comparative evaluation. It prescribes the algorithms to be followed in: (i) conflicting or inconsistent cases; for instance, stating whether the purchaser agent should alert its master in case of a conflict, or simply ignore it and conclude the issue with the consistent parts of the existing information (semi-autonomy of the agent), (ii) the sequencing of the evaluation process, that is specifying when to interact with the customer, whether iterations are allowed, etc., and (iii) the underlying multiple criteria decision making process.

Due to the variety of the information and mechanisms involved, the processes of offer synthesis and evaluation are described in detail in Section 4.

2.2 The seller agent

The architecture of a seller agent is similar to that of a purchaser (Figure 2). The *communication module* has exactly the same functionality with the homonymous module of the purchaser agent. The *coordination module* is responsible for the cooperation between (i) the seller and the purchaser agents, and (ii) the seller and its merchant. A *selling coordinator* manages the exchange of the related messages (see next section). The *selling database* keeps records of the products specification, and potential or regular customers (based on the history of previous transactions) to be informed about the release of a new or promoted product. Merchants may update the related databases of their software agents at any time.

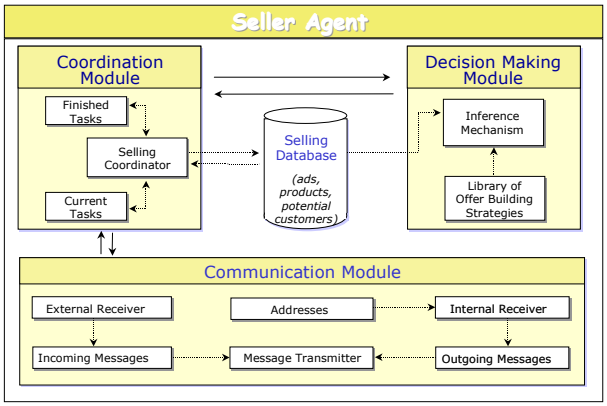

Figure 2: Architecture of a seller agent.

Finally, the *decision making module* consists of an *inference mechanism* and a *library of offer building strategies*. As in a purchaser agent, the inference mechanism of a seller also implements its proactive behavior (see next section). Furthermore, it uses the appropriate strategies to build offers for a requested or promoted product (see Section 4). Each such strategy reflects the selling policy to be followed by the seller agent, and may depend on the specific customer, product to be sold, merchant status, and so on (for instance, a different policy may be adopted when selling a new than a second-hand car).

3. TOWARDS AN ARTIFICIAL MARKET

The proposed e-market system is based on a network of communicating agents that act as *artificial employees* of the actors involved (i.e., customers and merchants), in that agents perform a series of tasks for them. Due to the diversity and complexity of the associated transactions, a proper definition of the interactions between humans and software agents, as well as a provision of procedures for data processing automation, are of high importance.

Agents interact by exchanging messages of various types. Each message type conveys certain semantics associated to a particular task of an e-market transaction. Each time an agent receives a message, it immediately knows what reasoning procedure it must activate in order to set up the most appropriate answer or action (if any) to the message received, or what kind of update it has to perform in its domain specific knowledge. In other words, each message type concerns a specific kind of interaction between the different kinds of participants in our framework. Table 1 provides a list of these messages, together with a short description of them.

Message Type	Sender	Receiver	Description
`offerReqMsg`	purchaser_Ag	seller_Ag	It conveys information about the customer's specifications for a product to be purchased
`offerPropMsg`	seller_Ag	purchaser_Ag	It concerns an offer proposal; it is actually a reply to an offer request (see previous message type)
`custSpecMsg`	customer	purchaser_Ag	Used to describe customer's criteria, preferences and constraints for a certain product to be purchased
`custSpecUpdReqMsg`	purchaser_Ag	customer	Used to request and provide, respectively, supplementary information regarding the customer's opinion about features, preferences and arguments introduced by a *seller* agent
`custSpecUpdAnsMsg`	customer	purchaser_Ag	
`purchInitMsg`	customer	purchaser_Ag	Initiation of a purchase transaction by a customer
`purchInitReqMsg`	purchaser_Ag	customer	They concern the interaction that is proactively initiated by a *purchaser* agent's request about whether the customer is interested in purchasing a (new or promoted) product that matches his/her interests
`purchInitAnsMsg`	customer	purchaser_Ag	
`merSpecMsg`	merchant	seller_Ag	Used to describe a merchant's products; it conveys their specification (similar to *custSpecMsg* above)
`merSpecUpdReqMsg`	seller_Ag	merchant	These concern the interaction that is proactively initiated by a *seller* to get extra information (e.g., when new features appear in a related offer request) or up-date existing one (e.g., when its offers get discarded)
`merSpecUpdAnsMsg`	merchant	seller_Ag	
`newProdMsg`	seller_Ag	purchaser_Ag	Sent whenever the database of a *seller* agent is updated with a new or promoted product
`newSellAgMsg`	seller_Ag	purchaser_Ag	Sent whenever a new *seller* agent is uploaded in the market (info about its coordinates and products)
`newPurchAgMsg`	purchaser_Ag	seller_Ag	Sent whenever a new *purchaser* agent is uploaded in the market (info about its coordinates and profile)
`newOfferAnnMsg`	merchant	seller_Ag	Sent whenever a merchant wants to launch a new offer (for a specific product, under a certain strategy)
`newOfferMsg`	seller_Ag	purchaser_Ag	Sent as soon as a *seller* agent receives the above message
`custDecisMsg`	customer	purchaser_Ag	Used to describe a customer's decision about the acceptance or rejection of an offer (this message type is also forwarded from a *purchaser* to a *seller* agent, and from the latter to its associated merchant)

Table 1: Messages passed in the e-market.

The most important human-agent collaboration issues of our approach are illustrated in the sequel through Figures 3-5. More specifically, processes performed at each actor/agent's side are depicted in the activity diagram of Figure 3, while two communication patterns (i.e., allowed sequences of messages

passed) are shown in the interaction diagrams of Figure 4 and 5 [14].

Actors are logged in the system, "hire" their personal agents (by paying a subscription depending on the time they want them to "live"), create a *profile* for them, and launch them in the e-market. An agent's profile serves its *personalization*, that is, the process with which an actor supplies his/her agent with the necessary information to sketch himself/herself (Fig. 3, *place specs* activity). Initially, this information may concern general interests, preferences and constraints, when speaking about a customer, or the market area and set of services offered, when speaking about a merchant. In such a way, a purchaser agent may be aware that its actor is generally interested in classical music and philosophical books while a seller agent that its actor commercializes music CDs and permanently makes offers to its regular customers.

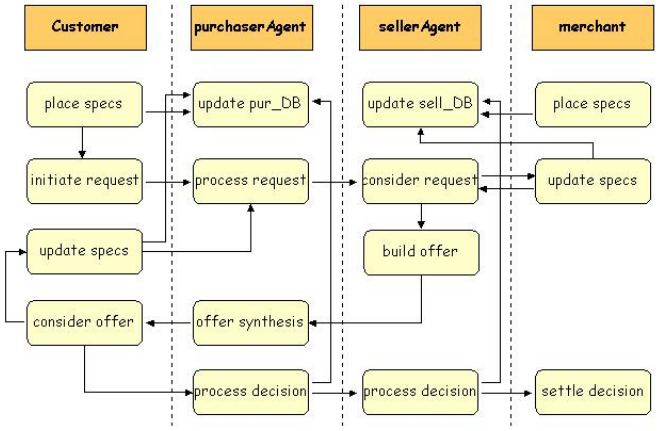

Figure 3: Agent-human activity diagram.

The above "general" knowledge is updated and enriched with more specific one each time a transaction is taking place (Fig. 3, *update specs* activity). For instance, at the time a customer is about to buy a trip, its purchaser agent will *learn* that he/she considers some adventure issues (e.g., the option to dive and rock-climb), prefers exotic destinations, and is not willing to pay more than a certain amount for it. This knowledge will be reused when the customer will buy his/her next trip. Similarly, a seller agent is able to update its profile each time it performs a transaction (e.g., by categorizing a customer as a regular one the second time a purchase agreement has been made with him/her, in order to send him special offers in the future, or "keeping a note" that this customer is interested in diving trips, thus refraining from sending him ski resort offers).

E-market transactions in our system are initiated either by an actor or an agent (scenarios 1 and 2 in Figures 4 and 5, respectively). In the first case (see Figures 3 and 4), a customer looking for a certain good or service contacts his/her purchaser agent and initiates a purchase transaction (Fig. 4, *purchInitMsg* message); in turn, the purchaser agent requests (from all or some seller agents) offers that may fulfill its actor's interests (Fig. 4, *offerReqMsg* messages).

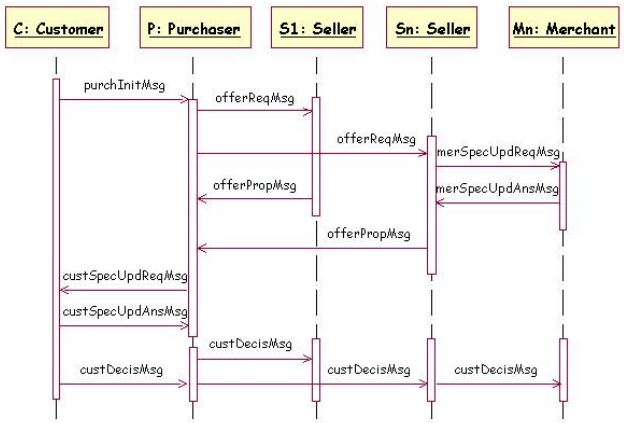

Figure 4: Agent-human interaction diagram: scenario 1.

Whenever a match between a purchaser and a seller agent is established, the latter gets information about the customer's buying criteria, preferences that may hold among them, as well as constraints explicitly imposed. By getting such a request, and presuming that the appropriate information exists in its selling database, a seller agent can directly build and propose an offer that is as close as possible to the purchase request (*offerPropMsg* message, sent by the seller agent S1). Otherwise (i.e., not enough information in the database), it has first to contact its merchant for an update of the related specifications (*merSpecUpdReqMsg* and *merSpecUpdAnsMsg* messages, exchanged between the seller agent Sn and its associated merchant, before the *offerPropMsg* message, sent by Sn).

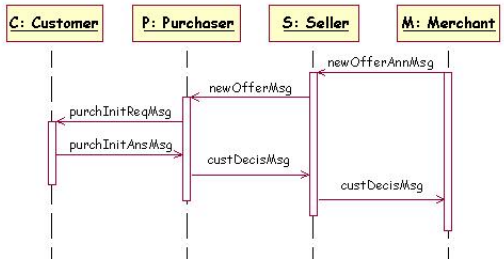

Figure 5: Agent-human interaction diagram: scenario 2.

Having collected a bunch of such offers, a purchaser agent has to consider and evaluate them all, the aim being to eventually recommend the best one to its user. At this stage, messages *custSpecUpdReqMsg* and *custSpecUpdAnsMsg* (see Figure 4) are used to request and provide, respectively, supplementary information regarding the customer's opinion about features, preferences and arguments introduced by a *seller* agent. Finally, the customer makes a decision about the acceptance or rejection of a proposal, which is forwarded to all interested parties.

We argue at this point that conflicts among the different seller agents' points of view are usually inevitable; before responding to a purchaser agent's request, each seller agent would have tailored its offer according to the range of goods at hand. Moreover, each seller agent may adopt its own *strategy* and, subsequently, propose an offer that fulfills (some of) the purchaser agent's goals at a certain

level. Offers may also differ about the relative values of criteria. In addition, the purchaser and the seller agents may have arguments supporting or against alternative solutions. Finally, before making a decision, the purchaser agent may have to confront the existence of insufficient information; that is, information that would be useful for making a decision is missing (these issues are discussed in detail in the next section).

Figure 5 corresponds to the second scenario mentioned above, that is an e-market transaction initiated by an agent. A merchant may ask his/her seller agent to broadcast or selectively send an offer for a certain product (see *newOfferAnnMsg* and *newOfferMsg* messages). This is related to the capabilities of our system's agents to be proactive and semi-autonomous. More specifically, a purchaser agent whose profile matches to a merchant's offer takes the initiative to contact its actor and ask his/her opinion to go ahead (either to purchase directly the certain product, as shown in Figure 5, or retrieve related offers and evaluate them together with the above offer, in the way explained above). Similarly acting, a seller agent, when noticing that the market's purchaser agents continuously discard its offers due to their prices, can suggest its actor to lower them.

begin
1. get `offerReqMsg`
2. search the selling db for `objectType = purchaseObj`
3. *if* not exists then *exit*,
   else for each `objectType.objToBeSold` found *do*
   if all `offerReqMsg.constraints` are satisfied
   3.1. create a new `offerPropMsg`
   3.2. `offerPropMsg.sender` ← `agentID`;
   `offerPropMsg.receiver` ← `offerReqMsg.sender`;
   `offerPropMsg.msgID` ← `offerPropMsg.msgID` + 1;
   `offerPropMsg.purchaseID` ← `offerReqMsg.purchaseID`;
   `offerPropMsg.purchaseObj` ← `objectType.objToBeSold`
   3.3. search in `objectType.objToBeSold` for criteria and associated features appear in `offerReqMsg.description`;
   for each match (`criterion` & associated `feature`) found *do*
   add in `offerPropMsg.description` a new item `(cr, f, fv, imp)` where, `cr` and `f` correspond to the match found, while `fv` and `imp` the values of `featureValue` and `impact`, respectively, of `objToBeSold`
   end for
   3.4. *for* each additional pair of `criterion` and `feature` (that not exists in `offerReqMsg.description`)
   if the respective `impact` is `pro` (positive)
   add in `offerPropMsg.description` a new item `(cr, f, fv, imp)` where, `cr` and `f` correspond to the additional match, while `fv` and `imp` the corresponding values of `featureValue` and `impact`, respectively, of `objToBeSold`
   end if
   end for
   3.5.search in `objectType.objSold` for `offerReqMsg.preferences`;
   for each found *do*
   if there exist `arguments` referring to it
   add in `offerPropMsg.preferences` a new item `(cr1, impRel, cr2, listOfArg)` where, `cr1, impRel` and `cr2` correspond to the match found, while `listOfArg` are links referring to this preference of `objToBeSold`;
   add in `offerPropMsg.arguments` a new item `(argID,pref,just,imp)` where, `argID, pref, just` and `imp` the corresponding values of `objToBeSold`
   end if
   end for
   3.6. *for* each additional preference of `objToBeSold` (that not exists in `offerReqMsg.preferences`) *do*
   if there exist `arguments` referring to this preference
   add in `offerPropMsg.preferences` a new item `(cr1, impRel, cr2, listOfArg)` where, `cr1, impRel, cr2` and `listOfArg` the corresponding values of `objToBeSold`;
   add in `offerPropMsg.arguments` a new item `(argID,pref,just,imp)` where, `argID, pref, just` and `imp` the corresponding values of `objToBeSold`
   end if
   end for
   end if
end for
end

Figure 6: An offer building strategy

4. DECISION MAKING ISSUES

Having defined the architecture of the purchaser and seller agents, as well as their cooperation and communication protocols, this section focuses on the multiple criteria decision making process *per se*. This process takes place in the decision making module of the purchaser agent. The tool implemented for the automation of this process is an extension of the work presented in [6].

Throughout the rest of this paper, we consider the following example scenario: A customer has uploaded in the e-market his/her assistant purchaser agent and is now interested in purchasing a new car. Before asking the agent to initiate a purchase transaction, the customer has well shaped in his/her mind that the criteria of *performance*, *cost*, and *safety* (let him/her ignore for the moment that of *firm's image*) are critical for the buying decision. He/she has also ranked the relative importance of performance and safety as the former being more important than the latter. Moreover, he/she intends to pay less than 30,000 Euros. He/she could also desire the maximum speed of the car to be more than 200 km/h. Note that, according to the purchaser agent's strategy, only pieces of that knowledge can be made transparent to the seller agents, the rationale being that the less clear the specification of the customer's intentions are, the more offers will be finally submitted by the seller agents.

Concerning the building of an offer by a seller agent, its inference mechanism gets as input an *offerReqMsg* message, consults the *selling database* and the library of strategies to retrieve the appropriate data and algorithms, respectively, and produces an *offerPropMsg* message as output. The offer building strategy used in our scenario is sketched in Figure 6.

4.1 Offer synthesis

As soon as a purchaser agent gets a new offer proposal, it integrates it with the ones already arrived and constructs an *offers synthesis graph*, which is presented to the customer through the web interface shown in Figure 7 (there is an optional *time limit*, set by the purchaser, after which no more offers for the specific purchase transaction are accepted). In this graph instance, there are three proposals so far, namely offer-12: car-6.1, offer-29: car-A25 and offer-16: car-XY-34, submitted by sellerAgent-33, sellerAgent-12 and sellerAgent-3, respectively. As is the case here, an offer may consist of:

- *criteria* (e.g., criterion-22.5: safety; criterion-22.8: cost) together with the associated *features*, their *values* and *impact* (e.g., (feature-22.5.3: airbag, 2, neutral), (feature-22.8.1:

purchase_price, 25000, pro)), where the last entity reflects the opinion of a seller agent about the feature value it provides (it can be pro (+), con (-), or neutral; see the related buttons in Figures 7 and 8);

- *preferences* brought up either by the purchaser agent when requesting an offer (e.g., preference-22.13: (performance, more_important, safety)) or the seller agent itself when replying to such a request (e.g., preference-33.3: (safety, more_important, performance));
- *arguments* in favor (e.g., argument-33.13: (report12: "safety was the big issue in 1998 car sales")) or against (e.g., argument-33.11: (if (maximum_speed > 200) then (accident_risk, high))) a preference.

An offer synthesis graph also includes the *constraints* asserted by the customer. Constraints are of the form (feature, relation, featureValue), where the desired value may fall into a numerical range, a set of discrete values, or a list of predicates (e.g., constraint-22.1: purchase_price, less_than, 30000).

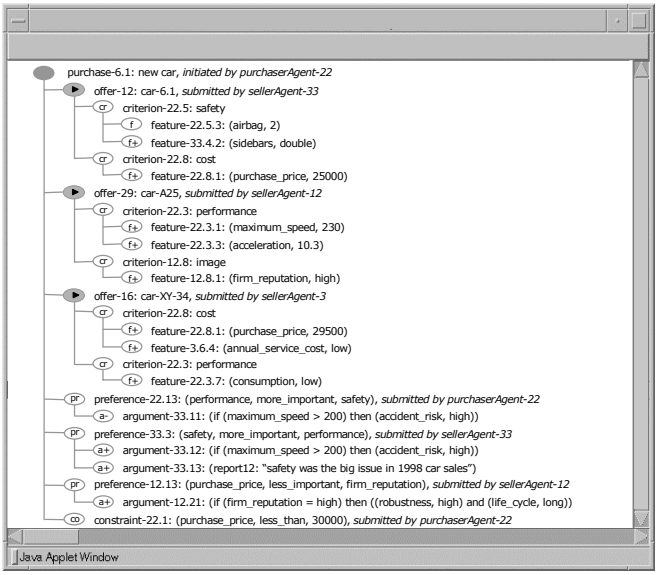

Figure 7: The offers synthesis graph.

4.2 Interacting with the user

Preferences and constraints are kept together at the bottom part of the graph, since these refer to the overall purchase transaction. Each graph entry has an *activation label* indicating its current status (it can be *active* or *inactive*). By default, all entries are initially active. Viewing the graph through a standard web browser, the customer is able to *inactivate* any of its nodes (by using the mouse and clicking on them), the rationale being that their corresponding data types do not suit to his/her interests.

Each offer's feature value is also checked against the existing constraints. According to the offer building strategy followed in the example presented here, the offer proposals sent do not violate these constraints; however, since the strategy followed by a seller agent is not known to the purchaser, this check is always performed at the latter's side. Nodes that violate a constraint become automatically inactive, that is upon the presentation of the offer synthesis graph to the customer. He/she may then - at any time - activate again (some or all of) these nodes (with the mouse) and test again the outcome of the decision making procedure. In general, the manual activation/inactivation of the graph nodes enables the customer to further elaborate the problem, by examining various alternative scenarios.

Figure 8 shows the status of the offers synthesis graph after the customer's intervention. Inactivation of a node renders all of its children nodes inactive; for instance, inactivation of preference-22.13: (performance, more_important, safety) also inactivates argument-33.11: (if (maximum_speed > 200) then (accident_risk, high)). Upon inactivation, the color of the associated button for each node changes (note the darker background color) denoting that these nodes will not be taken further into account in the decision making process.

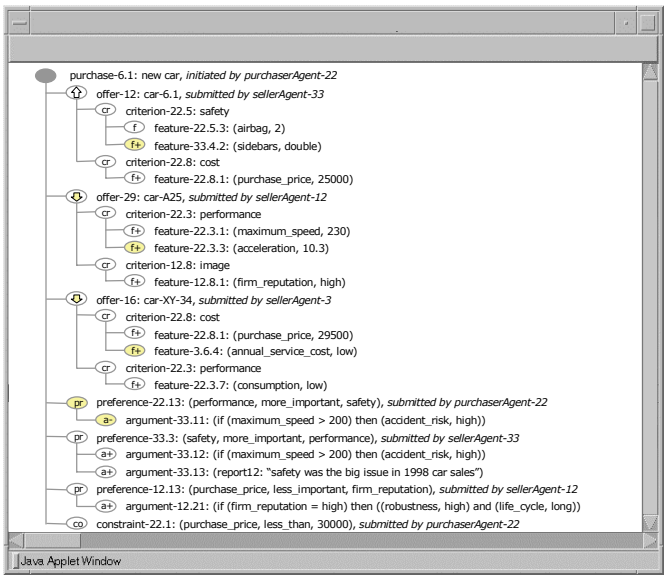

Figure 8: Intervention of customer and purchase suggestion.

4.3 Detection of conflicts and inconsistencies

Apart from an activation label, each preference has a *consistency label*, which can be *consistent* or *inconsistent*. Each time a preference is inserted in the offer synthesis graph, a mechanism checks if the constituent features or criteria of it exist in another (already inserted) preference. If yes, the new preference is considered either *redundant*, if it also has the same importance relation, or *conflicting*, otherwise. A redundant preference is ignored (not inserted in the graph), while a conflicting one is put next to the previously inserted preference, the rationale being to gather together conflicting preferences and stimulate the user to contemplate on them (that is, to select which one to inactivate), until only one becomes active. Such an instance is illustrated in Figure 8, with the entries preference-22.13: (performance, more_important, safety) and preference-33.3: (safety, more_important, performance).

If both features (or criteria) of a new preference do not exist in a previously inserted preference, its *consistency* is checked against

previous active and consistent preferences. Consider, for example, a situation, where there exist two preferences (feature-x, more_important, feature-y) and (feature-y, more_important, feature-z). A new preference (feature-z, more_important, feature-x) is inconsistent with respect to the first two ones, although it is not directly conflicting with either one. Inconsistency checking is performed through a polynomial ($O(N^3)$, N the number of the associated features) *path consistency* algorithm. Although the algorithm interacts with the database where the offers synthesis graph is stored (the public-domain *mSQL* has been integrated in the module), the algorithm is efficient; even for cases involving offers with numerous criteria and associated features, execution time is negligible.

4.4 The weighting schema

Active and consistent preferences participate in the weighting scheme (only preference-33.3 and preference-22.13 in the example of Figure 8). A detailed description of the algorithm used to assign weights to an offer's features appears in [7]. The basic idea is that the weight of a feature (or a criterion) is increased every time it is more important than another one (and decreased when is less important), the final aim being to extract a total order of offers. Since only partial information may be given, the choice of the initial maximum and minimum weights may affect the purchaser agent's recommendation. However, the above weighting scheme is not the only solution; alternative schemes, based on different algorithms, have been also implemented. The *score* of each offer is calculated from the weights of the active features the offer consists of (we assume that an offer which includes no product features gets score = 0), according to the formula:

$$\text{score}(\text{offer}_i) = \sum \text{weight}(\text{feature}_j) - \sum \text{weight}(\text{feature}_k),$$

where feature_j (feature_k) an active feature which refers to offer_i having a positive (negative) impact. The scores of offer-12, offer-29 and offer-16 in Figure 8 are 11, 9 and 10, respectively. Concerning the first one, both feature-22.5.3 and feature-22.8.1 have score 5.5, while feature-33.4.2 is inactive; similarly, for the other two offers, it is score(feature-22.3.1) = score(feature-12.8.1) = score(feature-22.3.7) = 4.5, while feature-22.3.3 and feature-3.6.4 are inactive. Therefore, offer-12 is the one recommended by the purchaser agent (as shown in Figure 8, the best proposal is accompanied by an "up-arrow" button, while the rest by a "down-arrow" one). Once again, this may change in the future upon a different configuration (activation/inactivation) of the offers' features by the customer, or the receipt of a new offer (assuming that the time limit given has not been exceeded).

5. DISCUSSION

Several interesting works have been already proposed in the area of agent-mediated electronic commerce. For instance, Excite's *Jango* [5] provides a comparison shopping Internet site, allowing users to specify the name and category of an item before searching on-line stores for the lowest prices available. It is based on a rather low-level approach, which does not address any of the issues highlighted in our approach. Being more sophisticated, *PersonaLogic* [15] provides a set of predefined, category-based "guides", and allows customers impose constraints, to be then exploited by a constraint satisfaction engine in order to prune alternatives that do not satisfy them. Compared to our system, only a comparative evaluation of the matched offers is supported; however, the constraints imposed are predetermined, upon the "guide", and cannot refined or amended.

Kasbah [1] helps users creating agents to negotiate the buying and selling of goods on their behalf, also allowing the specification of parameters to guide and constrain an agent's overall behavior. However, these agents live only during the completion of a certain transaction, thus not fully exploiting each actor's profile, as well as the proactiveness and semi-autonomy of their agents (e.g., towards newcoming offers or requests). Negotiation in Kasbah is straightforward and based on some simple heuristics; this makes it intuitive for users to understand what their agents are doing in the marketplace, but does not allow for: (i) "open" argumentation and criteria refinement, (ii) handling of incomplete, inconsistent and conflicting data, and (iii) progressive synthesis and comparative evaluation of the entries matched. .

Finally, *Tête-à-Tête* [9], unlike most other online negotiation systems that competitively negotiate over price, equally considers product and merchant features to help the shopper simultaneously determine what to buy and whom to buy from. It also provides a set of pre-determined, user-profile based specifications for the requirements of each product category, and multi-attribute utility theory to rank merchant offerings. Tête-à-Tête is certainly close to our approach in that it allows consumer-owned shopping agents and merchant-owned sales agents cooperate across multiple terms. However, the issue of the long (or even permanent) existence of a consumer's agent is not fully exploited (see comments above); in addition, proactiveness and semi-autonomy of both shopping and sales agents is limited compared to our approach (see Sections 2 and 3). Furthermore, argumentation in our system is more flexible, in that it does not have to be based on pre-determined specifications. The mechanisms deployed can efficiently handle any kind of incomplete, inconsistent and conflicting data. We also argue that our interface for the progressive synthesis and comparative evaluation of multiple offers is more intuitive and closer to the real way of thinking of a customer (see Section 4).

The system presented in this paper is fully implemented in Java and runs on Windows NT. It is based on previous well-tried work concerning intra-agent control [10], inter-agent communication [10], and automation of multiple criteria decision making [6, 7]. Agents communicate using TCP/IP, while actors interact with them through web interfaces. All transactions carried out use information encoded in XML/EDI format [2, 17]. A start-up company is currently testing the system's first fully integrated version.

A future work direction concerns the integration of a more elaborated negotiation stage (for an overview, see [8]), by exploiting a multicriteria-based negotiation model that has first been presented in [11]. Negotiation of a set of product features, for instance, between a purchaser and a seller agent will enable the latter better tailoring its offer proposal and, eventually, work more efficiently for its merchant. We also intend to work towards an adaptation of the system that will address various types of auctions; the system's architecture and interaction protocols are "open", thus their appropriate fine-tuning is not expected to be a hard task.

6. ACKNOWLEDGMENTS

Our thanks to Nikos Spanoudakis for helping us create the interaction diagrams included in this paper.

7. REFERENCES

[1] Chavez, A., and Maes, P. Kasbah: An Agent Marketplace for Buying and Selling Goods, In *Proceedings of PAAM-96*. London, UK, 1996.

[2] Glushko, R., Tenenbaum, J., and Meltzer, B. An XML Framework for Agent-Based E-Commerce, *Communications of the ACM* 42(3), pp. 106-114, 1999.

[3] Guttman, R., Moukas, A., and Maes, P. Agents as Mediators in Electronic Commerce, *Electronic Markets* 8(1), pp. 22-27, 1998.

[4] Guttman, R., Moukas, A., and Maes, P. Agent-Mediated Electronic Commerce: A Survey, *Knowledge Engineering Review* 13 (3), 1998.

[5] Jango: http://www.jango.com

[6] Karacapilidis, N., and Papadias, D. Hermes: Supporting Argumentative Discourse in Multi-Agent Decision Making, In *Proceedings of AAAI-98,* AAAI/MIT Press, pp. 827-832, 1998.

[7] Karacapilidis, N., and Papadias, D. A Computational Approach for Argumentative Discourse in Multi-Agent Decision Making Environments. *AI Communications Journal* 11(1), pp. 21-33, 1998.

[8] Lo, G., and Kersten, Gr. Negotiation in Electronic Commerce: Integrating Negotiation Support and Software Agent Technologies, In *Proceedings of the 29th Atlantic Schools of Business Conference*, Halifax, NS, 1999, (CD ROM).

[9] Maes, P., Guttman, R., and Moukas, Al. Agents that Buy and Sell, *Communications of the ACM* 42(3), pp. 81-91, 1999.

[10] Matsatsinis, N., Moraïtis, P., Psomatakis, V., and Spanoudakis, N. Intelligent Software Agents for Products Penetration Strategy Selection, In *Proceedings of MAAMAW-99*, Valencia, Spain, June-July 1999.

[11] Moraïtis, P., and Tsoukias, A. A Multi-Criteria Approach for Distributed Planning and Conflict Resolution for Multi-Agent Systems, In *Proceedings of ICMAS-96*, MIT Press, pp. 213-219, 1996.

[12] Nwana, H., Ndumu, D., Lee, L., and Collis, J. ZEUS: A Tool-Kit for Building Distributed Multi-Agent Systems, *Applied Artificial Intelligence Journal* 13(1), pp. 129-186, 1999.

[13] Nwana, H., Rosenschein, J., Sandholm, T., Sierra, C., Maes, P., and Guttmann, R. Agent-Mediated Electronic Commerce: Issues, Challenges and some Viewpoints, In *Proceedings of Autonomous Agents 98,* ACM Press, pp. 189-196, 1998.

[14] Odell, J., Van Dyke Parunak, H., and Bauer, B. Extending UML for Agents, in Proceedings of the Agent-Oriented Information Systems AAAI-2000 Workshop, Austin, TX, pp. 3-17, 2000.

[15] PersonaLogic: http://www.personalogic.com

[16] Sycara, K., and Zeng, D. Coordination of Multiple Intelligent Software Agents, *International Journal of Cooperative Information Systems* Vol. 5 (2-3), 1996.

[17] Webber, D. Introducing XML/EDI Frameworks, *Electronic Markets* 8(1), pp. 38-41, 1998.