ELSEVIER

Decision Support Systems 32 (2001) 53–69

Decision Support Systems

www.elsevier.com/locate/dsw

Building an agent-mediated electronic commerce system with decision analysis features

Nikos Karacapilidis [a,*], Pavlos Moraïtis [b]

[a] *Industrial Management Laboratory, Department of Mechanical Engineering and Aeronautics, University of Patras, GR 26500, Rion Patras, Greece*

[b] *LAMSADE, University of Paris-Dauphine 75775, Paris Cedex 16, France*

Abstract

This paper describes a web-based electronic commerce system in which customers and merchants delegate the related tasks to their personal software agents. Messages passed between these agents can fully encapsulate the associated parties' points of view towards a market transaction. More specifically, an offer request consists of a list of the product attributes the customer wants to know about, a partial order of their importance, and the constraints imposed. On the other side, an offer proposal can be tailored according to the information conveyed in the corresponding offer request. Advanced features of the system include the permanent existence of our agents in the market, thus being able to learn from it, their ability to act proactively in order to initiate a transaction, and the integration of an interactive multiple criteria decision-making tool, with which a buying agent performs a comparative evaluation of the proposals in a semi-autonomous way.

Keywords: E-commerce; Software agents; E-market; Web-based information system; Multiple criteria decision making

1. Introduction

Research work done during the last few years has dealt with a diversity of tasks involved in buying and selling goods and services in an electronic market (e-market) [1,19], and resulted to the implementation of a plethora of systems automating tasks such as product brokering, merchant brokering, and negotiation [5,17]. Moreover, as many analysts predict, agent-mediated electronic commerce would further revolutionize Internet Commerce (see, for instance, Ref. [16]). This is mainly due to the fact that most basic 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.

This paper describes a new agent-mediated electronic commerce system for the contemporary e-market. Its overall framework is not based on pre-classified ads; instead, the system's agents cooperate and get the related information in a real-time mode. Contrary to the majority of the already implemented systems, the one presented here addresses efficiently all of the following important issues.

(i) The permanent existence of agents in the e-market; that is, agents that do not "live" only during a specific transaction but much longer, upon

* Corresponding author.

E-mail addresses: nikos@mech.upatras.gr (N. Karacapilidis), moraitis@lamsade.dauphine.fr (P. Moraïtis).

0167-9236/01/$ - see front matter
PII: S0167-9236(01)00100-2

the subscription paid by their owners at the time they were launched (an actor may "hire" an agent for a month, a year, etc.).

(ii) The proactiveness and semi-autonomy of agents; that is, agents that 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). Semi-autonomy of agents assures the right level of control for the actions they could take (a fully autonomous agent could cause problems).

(iii) The maintenance of each actor's profile through the personalization of the agents involved; for instance, a customer's agent is supplied with a number of general interests (e.g., classical music, cruises) and preferences (e.g., one may dislike the black color on any product) of its actor, which can be enriched with more detailed ones each time the customer initiates a transaction. This gets extra value when agents "live" permanently in the e-market (see first issue above).

(iv) The ability of a seller agent to refine some of the customer's purchase criteria, argue in favor or against them, or even bring up new information to persuade him/her to accept its offer.

(v) The ability 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 requires a highly interactive tool, based on multiple criteria decision theory, that 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.

Table 1 provides a comparative insight of our approach against four representative e-commerce systems, regarding the abovementioned features (the checkmark symbol denotes the existence of the related feature, "×" declares its absence, while the asterisk stands for a partial or alternative implementation of it). More specifically, Excite's *Jango* (http://www.jango.com) provides a comparison shopping Internet site, just 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 above issues. Being more sophisticated, *PersonaLogic* (http://www.personalogic.com) 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. Regarding the issues above, 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 [2] helps users creating agents to negotiate the buying and selling of goods on their behalf,

Table 1
Features provided in our and some representative e-commerce system approaches

	Jango	PersonaLogic	Kasbah	Tete-a-Tete	Our approach
Permanent existence of e-market agents	✖	✖	✖	✖	✔
Proactiveness & semi-autonomy of agents	✖	✖	✷	✷	✔
Maintenance of each actor's profile	✖	✖	✷	✔	✔
Argumentation & criteria refinement features	✖	✖	✖	✷	✔
Handling of incomplete, inconsistent & conflicting data	✖	✖	✖	✷	✔
Progressive synthesis & comparative evaluation	✖	✷	✖	✷	✔

also allowing the specification of parameters to guide and constrain an agent's overall behavior. However, these agents do not live permanently but 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 upcoming 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* [12], 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 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. In conclusion, we 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 purchaser (see Section 4).

The remainder of the paper is structured as follows. The proposed e-market framework is analyzed in Section 2. The architecture of the software agents involved is illustrated in Section 3. The multiple criteria decision-making process is presented in Section 4. Finally, a real application of the system is discussed in Section 5, while concluding remarks and future work directions are outlined in Section 6. It should be made clear here that issues such as ordering, security, payment and delivery, while equally important in commercial transactions, do not fall in the primary scope of this paper.

2. The e-market framework

The proposed e-market system is based on a network of communicating agents that act as *artificial employees* of the related actors (i.e., customers and merchants), in that they perform a series of tasks for them. 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 (Fig. 1).

2.1. Cooperation protocol

An agent's profile serves its *personalization*, that is, the process with which an actor supplies his/her

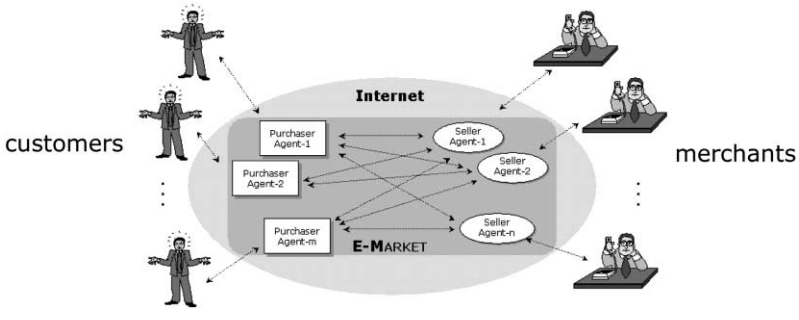

Fig. 1. Participants in an e-market.

agent with the necessary information to sketch himself/herself. 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.

The above "general" knowledge is updated and enriched with more specific one each time a transaction is taking place. For instance, 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. In the first case, a customer looking for a certain good or service contacts his/her purchaser agent and provides it with all the necessary information; in turn, the purchaser agent requests (from all or some seller agents) offers that may fulfill its actor's interests (standardization issues, emerged here, are not addressed in this paper). Similarly, a merchant may ask his/her seller agent to broadcast or selectively send an offer for a certain product. This leads to the second case above, which 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. In case that the customer is interested in the certain product, the purchaser agent is capable to take action in order to retrieve related offers and evaluate them all. In a similar way, 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.

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 the above information, a seller agent is able to build an offer that is as close as possible to the purchase request. 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. 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.

2.2. Communication protocol

To establish the cooperation protocol described above, agents communicate by exchanging messages of various types. Each message type conveys certain semantics associated to a particular task of the purchase procedure. Each time an agent (purchaser or seller) receives a message, it immediately knows what reasoning procedure it must activate in order to deploy 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, namely customers, purchaser agents, seller agents, and merchants. These interactions are schematically represented in Fig. 2, whereas the messages involved are analyzed in the sequel of this section. Throughout the rest of this paper, motivated by a recent marketing firm's report pointing out that "25% of all new-vehicle buyers use the Internet to arm themselves with vehicle product and pricing information during the vehicle-shopping process" [18], we consider a scenario where a customer is interested in purchasing a new car.

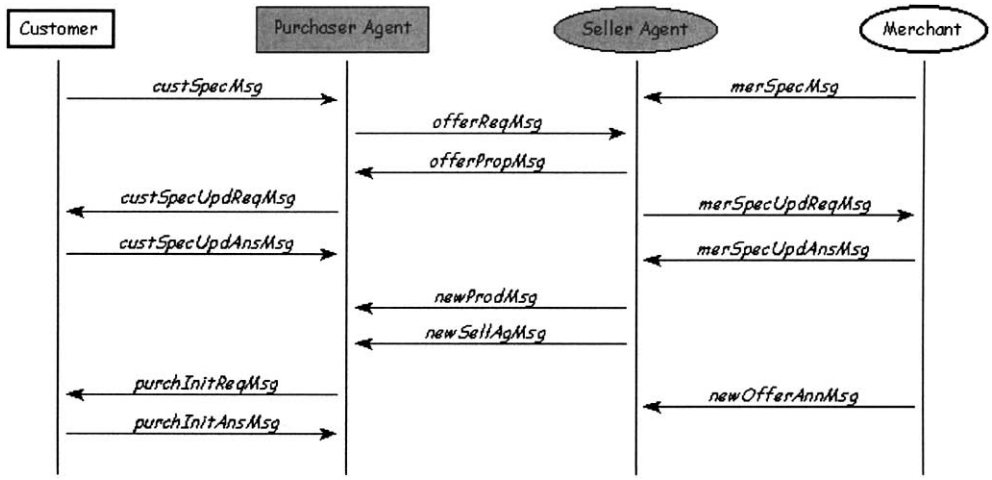

Fig. 2. A schematic summary of actors' interactions.

2.2.1. The offer request message (`offerReqMsg`)

This message is sent by a `purchaser` to a `seller` agent and conveys information formulating the related customer's specifications for the product to be purchased. The structure of these messages is illustrated in Fig. 3. As shown, the `receiver` field may be "any" in case of broadcasting or a list of `seller` agents' IDs in case of selective communication. The purchaseID field refers to a general *purchase category* (vehicles in our case), while purchaseObj refers to the specific good the customer intends to buy (i.e., a new car). The criterion and feature fields provide information about the specific product's attributes. In our framework, *criteria* represent broader evaluation metrics (that is, metrics that may stand for more than one purchase categories; e.g. cost, performance, aesthetics, etc.), while *features* concern specific attributes of a criterion, which usually depend on the specific product to be bought (for example, other features are attached to the criterion of safety in the case of purchasing a car than those attached when purchasing a motorboat). Fields `featureValue` and `impact` are to be completed by a `seller` agent (the meaning of the former is straightforward; the purpose of the latter is explained in the following message type). The `description` field for the scenario used in this paper may contain entries such as (`cost`, `purchase_price`, `?featureValue`, `?impact`), (`cost`, `annual_service_cost`, `?featureValue`, `?impact`), (`performance`, `maximum_speed`, `?featureValue`, `?impact`), (`performance`, `acceleration`, `?featureValue`, `?impact`), etc.

OfferReqMsg		
`sender`	=	<the purchaser agent's ID>
`receiver`	=	<list of the seller agents' IDs or "any">
`msgID`	=	<the message's ID>
`purchaseID`	=	<the specific purchase's ID>
`purchaseObj`	=	<the object to be purchased>
`description`	=	<list of tuples of the form (`criterion, feature, ?featureValue, ?impact`)>
`preferences`	=	<list of tuples of the form (`criterion, impRel, criterion`), or (`feature, impRel, feature`)>
`constraints`	=	<list of tuples of the form (`feature, relation, featureValue`)>

Fig. 3. Structure of an *offer request* message.

Preferences provide a means to weigh pairs of criteria or features. They are represented by tuples of the form (criterion, impRel, criterion) or (feature, impRel, feature), where impRel ∈ {more(less)_important_than, equally_important_to}. A preference of the form (criterion-1, impRel-x, criterion-2) implies preferences of the form (feature-i, impRel-x, feature-j), where feature-i and feature-j have been classified under criterion-1 and criterion-2, respectively. Finally, *constraints* provide a way to express a customer's wish about the value a certain feature should possess. For the new-car case, (max_speed, more_than, 200), (airbag, at_least, 2) and (price, less_than, 30000) can be such data types.

2.2.2. The offer proposal message (offerPropMsg)

This message concerns an offer proposal sent by a seller agent in reply to a purchaser agent's offer request. Offer proposals are structured by seller agents according to a certain strategy, which serves (as much as possible) the interests of their owners. In general, such a strategy underlines the positive features of an object to be sold (even if these features have not been explicitly included in the corresponding offer request), while simultaneously attempts to underestimate, or even "hide", the negative ones.

An *offer proposal* message (see Fig. 4) has a similar structure to that of an *offer request*. Compared to it, the description field now contains the purchaser agent's answer for the feature requested (i.e., featureValue) and its opinion about the *impact* of this answer regarding the product it suggests. The impact field takes a value from the set {pro, con, neutral}. For example, (performance, acceleration, 8.5, pro) denotes a purchaser agent's opinion that an acceleration rate of 8.5 is a plus for the product it offers, while (cost, purchase_price, 25000, neutral) denotes its inability to decide whether a price of 25,000 is a plus or a minus for its offer. It should be noted here that the list of tuples in the description and preferences fields of an offer proposal may not necessarily be identical to those of the corresponding offer request, in that the seller agent may (upon the strategy it follows for a specific setting) tailor its offer (by not filling in all entries included in an offer request, while adding new ones). For instance, a merchant of very expensive cars may have his/her seller agent not providing any information about prices, while trying to promote other features such as safety, image, etc.

Furthermore, an offer proposal may contain arguments about the purchaser agent's preferences. Such arguments may either be stored in the related database or come up after the interaction of a seller agent with its human agent. The purpose of the *justification* provided is to further validate or challenge the customer's choices before the comparative evaluation of the offer proposals received.

2.2.3. The Customer–PurchaserAgent messages

There are five message types used for the communication between a customer and his/her purchaser agent, namely custSpecMsg, custSpecUpdReqMsg, custSpecUpdAnsMsg, purchInitReqMsg and purchInitAnsMsg. Using the first one, a customer is able to inform his/her assistant agent about the product(s) he/she would like to buy together with a related specification (i.e., criteria, features, preferences, constraints, etc.). The next two concern an update of this specification (at

OfferPropMsg		
sender	=	<the seller agent's ID>
receiver	=	<the purchaser agent's ID>
msgID	=	<the message's ID>
purchaseID	=	<the specific purchase's ID>
purchaseObj	=	<the object proposed by the seller agent >
description	=	<list of tuples of the form (criterion, feature, featureValue, impact)>
preferences	=	<list of tuples of the form (criterion, impRel, criterion, listOfArg), or (feature, impRel, feature, listOfArg)>
arguments	=	<list of tuples of the form (argId, preference, justification, impact)

Fig. 4. Structure of an *offer proposal* message.

any time of a purchase transaction). They may request and provide, respectively, supplementary information regarding the customer's opinion for a feature value, a preference, or some arguments introduced by a seller agent. Finally, the last two message types concern the interaction that is proactively initiated by a purchaser agent's request about whether the customer is interested in purchasing a new product (which has just been launched by a seller agent, see Section 2.2.5). As it will be described in more detail in Sections 2.2.4, 2.2.5, 2.2.6 and 2.2.7, a customer may either accept or reject certain items of an offer proposal; such messages are sent to a purchaser agent during the comparative evaluation of the offerings received and help in determining the best alternative.

2.2.4. The Merchant–SellerAgent messages

There are also three message types serving the communication between a merchant and his/her seller agent, namely merSpecMsg, merSpecUpdReqMsg and merSpecUpdAnsMsg. Using the first one, a seller agent becomes aware of the products it can sell, by getting a specification of them that reflects the corresponding merchant's policy (it may promote the advantages of these products, and so on). In addition, a seller can use a merSpecUpdReqMsg message to ask its owner about additional information (e.g., when new criteria or preferences, not currently available in its selling database, appear in an offer request) or his/her opinion on which strategy to be followed in a particular setting. The merchant's reaction is represented by a merSpecUpdAnsMsg.

2.2.5. A new product announcement (newProdMsg)

Each time the database of a seller is updated (either when a new product is introduced or an existing product is updated due to promotion reasons), the seller uses this type of message to broadcast its characteristics to the possibly interested purchaser agents of the e-market.

2.2.6. A new seller agent announcement (newSellAgMsg)

Each time a new seller is created and uploaded, it broadcasts this type of message to announce its presence to the purchaser agents of the e-market. This message contains information about its coordinates and the products it can sell.

2.2.7. A new offer announcement (newOfferAnnMsg)

Each time a merchant launches a new product or an offer promoting one of his/her already existing products, he/she sends this message to his/her seller agent.

3. The e-market software agents

The development of the software agents proposed in our e-market framework is based on a generic and reusable architecture, which has been conceived after examining the pros and cons of various existing approaches (see, for instance, Refs. [7,13,15,20]). Even if the two agent types involved do not have the same functionality, they are built on the same basic architecture principles (see Fig. 5). However, their constituent modules are tailored, according to their specific type (e.g., the decision-making module of a seller agent is usually much simpler than that of a purchaser agent). The architecture of each agent type is described in detail below.

3.1. The Purchaser agent

A purchaser agent is composed of three, 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.

3.1.1. The communication module

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 mod-*

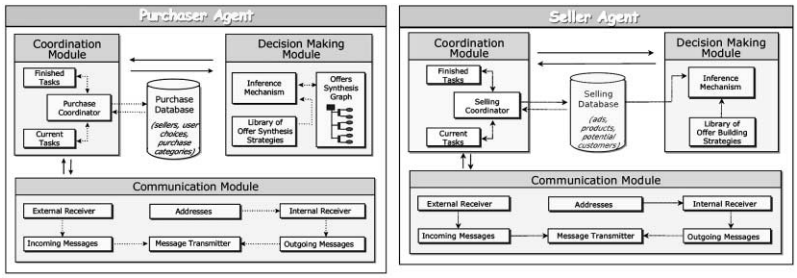

Fig. 5. Architecture of (a) a `purchaser` agent and (b) a `seller` agent.

ule. 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 or point-to-point communication, it also adds the receiving agent's address. An *external receiver* handles the opposite, 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 communication module described here is an adaptation of that appearing in Ref. [13].

3.1.2. The coordination module

This 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 (see previous 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 under evolution (e.g., a purchase evaluation for a specific product can be momentary suspended due to searching for supplementary information).

As shown in Fig. 5a, 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 (see, for instance, the `newProdMsg`, and `newSellAgMsg` message types), 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. It is a relational database, where all the above elements are structured in tables. Retrieving the appropriate data, a `purchaser` agent is aware of which sellers it can buy a product from, the products each `seller` agent provides, and the customer choices about a specific product.

3.1.3. The decision-making module

The decision-making module is composed of three components (Fig. 5a), namely an *inference mechanism*, a *library of offer synthesis strategies*, and the *offers synthesis graph* (see Section 4). It actually deploys the agent's reasoning mechanism that (i) implements the proactive behavior of the agent by using appropriate rules (which are activated upon the reception of a specific message, sent by `seller` agents), 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, 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.

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 decision-

```
begin
1. get offerReqMsg
2. search the selling database 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 and 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
```

Fig. 6. An offer-building strategy.

making process of the purchaser agent is described in detail in Section 4.

3.2. The Seller agent

The architecture of a seller agent is similar to that of a purchaser (see Fig. 5b). The communication module has exactly the same functionality with the homonymous module of the purchaser agent.

3.2.1. The coordination module

This 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, for instance, the message types offerReqMsg, offerPropMsg, merSpecMsg, newSellAgMsg and newProdMsg presented in Section 2). The coordination module interacts with the communication and the decision-making modules, as in a purchaser agent. The *selling database* basically keeps records of the products' specification together with an indication whether the value of a specific feature is a strong, indifferent or weak point. This information is represented by tuples of the form (product, criterion, feature, featureValue, impact), where impact may take a value from the set {pro, con, neutral}. The database may also contain information about potential customers (purchaser agents) to be informed about the release of a new product.

3.2.2. The decision-making module

This module consists of an *inference mechanism* and a *library of offer-building strategies*. As in a purchaser agent, the inference mechanism of a seller implements its proactive behavior, that is, actions taken upon the reception of a specific message sent by its merchant. Furthermore, it uses the appropriate strategies to build offers for a requested or promoted product. Each such strategy comprises a set of rules reflecting 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).

Concerning the building of an offer, the inference mechanism gets as input an offerReqMsg, consults the *selling database* and the library of strategies to retrieve the appropriate data and algorithms, respectively, and provides a set of offerPropMsg as output. The offer-building strategy used for our example scenario is sketched in Fig. 6.

4. The e-market multiple criteria decision-making process

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 Ref. [8]. For the scenario used in this paper, let the customer have well shaped in his/her mind that the criteria of *performance*, *cost*, and *safety* (let him/her ignoring for the moment that of *firm's image*) are critical for the buying decision, while 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 for the car.

4.1. Synthesis of offer proposals

For each specific offer request, the *purchase coordinator* of the purchaser agent receives offer proposals, which are progressively sent by a subset of the e-market's seller agents. Fig. 7(a) illustrates a proposal (offer-12: car-6.1), made by sellerAgent-33. This offer consists of criteria (e.g., criterion-22.5: safety, criterion-22.8: cost) together with the associated features, feature values and impact (e.g., (feature-22.5.3: airbag, 2, neutral), (feature-22.8.1: purchase_price, 25000, pro)), 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,

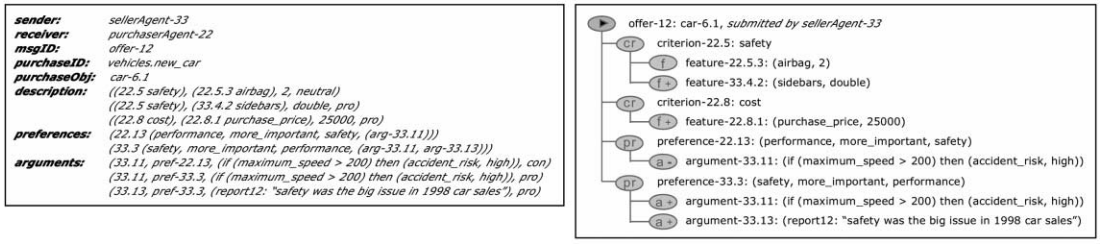

Fig. 7. A seller agent's offer proposal.

more_important, performance)) and, finally, 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.

Appropriate mechanisms in the decision-making module of the purchaser agent transform an offerPropMsg, such that appearing in Fig. 7a, into a graph of the form shown in Fig. 7b. As soon as a purchaser agent gets a new offer proposal, it integrates it with the ones already arrived, constructing an offers synthesis graph. We assume that there is a time limit, set by the purchaser, after which no more offers for the specific purchase transaction are accepted.

4.2. Interacting with the user

The offers synthesis graph is presented to the customer through a web interface (see Fig. 8). Note that preferences and constraints have been kept together at its bottom part (that is, after the offer proposals), since these refer to the overall purchase transaction. Also, note that the constraints appearing in this graph come from the related offerReqMsg

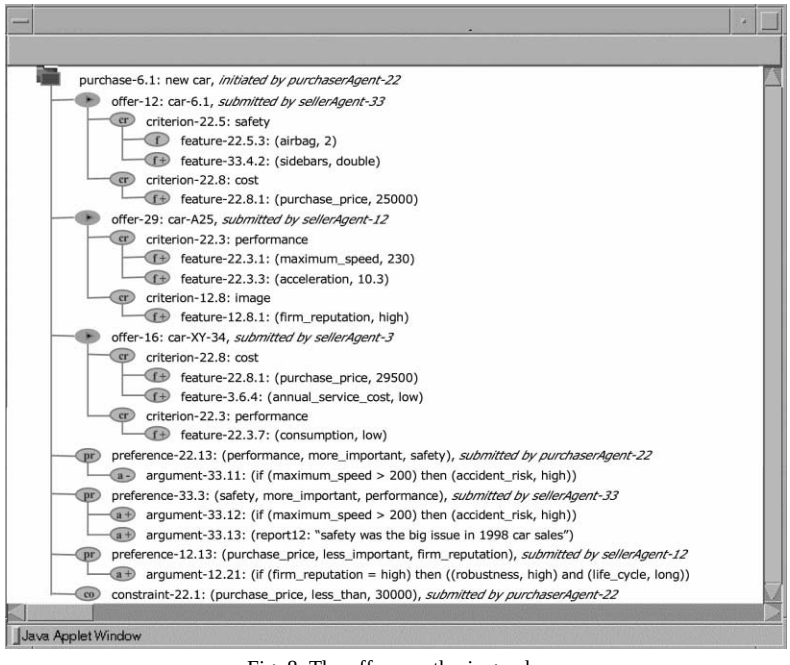

Fig. 8. The offers synthesis graph.

message (or possibly from updates of the initial specification, initiated by the customer); an `offerPropMsg` does not contain constraints. Each entry in this graph 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 the graph's nodes (by using the mouse and clicking on them), the rationale being that their corresponding data type do not suit to his/her interests.

Fig. 9 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 gray color) denoting that these nodes will not be taken further into account in the decision-making process.

Each offer's feature value is also checked against the existing constraints (note that in the offer-building strategy described in Fig. 6, 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). Recall that a constraint has 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. 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 (again, by using the mouse) and test again the outcome of the decision-making procedure. In general, the manual activation and inactivation of the graph nodes enables the customer to further elaborate the problem, by examining various alternative combinations.

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 (this follows the receipt of a new `offerPropMsg`), a mechanism checks if both

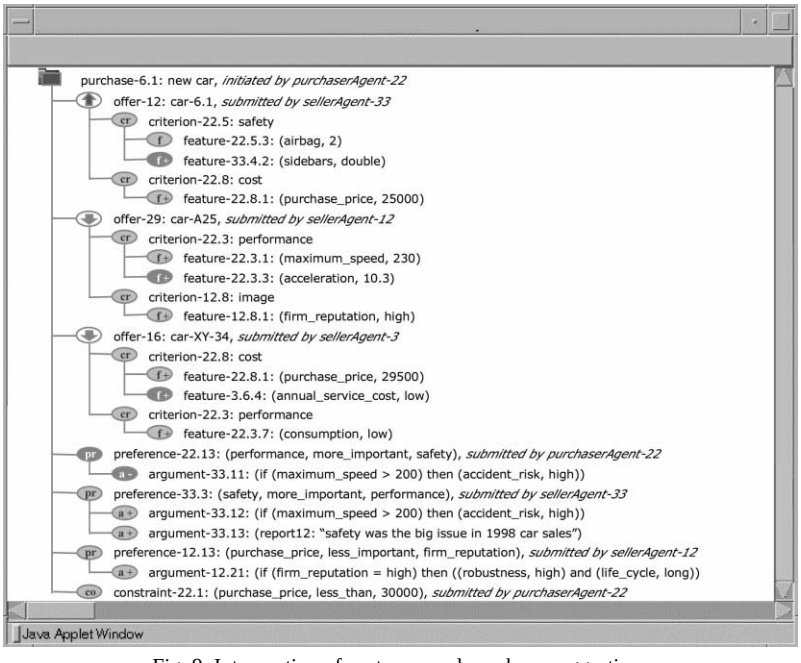

Fig. 9. Intervention of customer and purchase suggestion.

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 Fig. 9, 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 [11]. 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 Fig. 9). In order to demonstrate how the algorithm for assigning weights to an offer's features works, we use the example shown in Fig. 10. There exist five features and four preferences that relate them as illustrated in Fig. 10(a). Each feature has a `weight = (max_weight + min_weight)/2`. `Max_weight` and `min_weight` are initialized to some predefined values (in the following, we assume that initially `min_weight = 0` and max_weight = 10). The arrowed lines correspond to the "more important" relation (e.g., (f_1, `more_important`, f_2)) and the dotted line to the "equally important" relation (e.g., (f_3, `equally_important`, f_4)). Path consistency explicates all the "more important" relations. More specifically, *topological sort* [9] is applied twice to compute the possible maximum and minimum weights for each feature (Fig. 10(b)). The weight is the average of the new `max_weight` and `min_weight`: `weight`(f_1) = 6, `weight`(f_2) = 4.5, `weight`(f_3) = 5, `weight`(f_4) = 5 and `weight`(f_5) = 4.

The basic idea behind the above scheme 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,

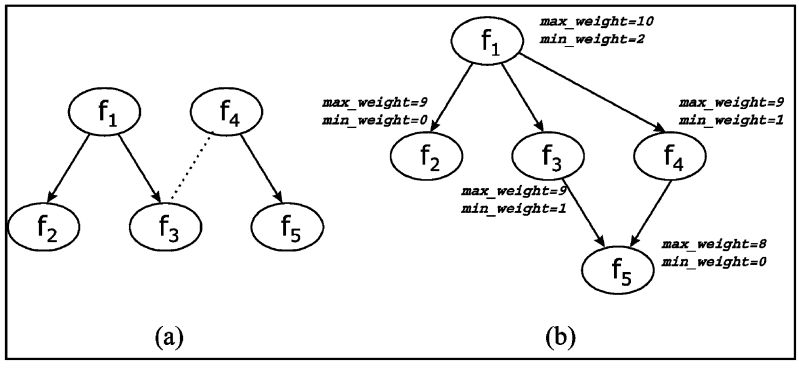

Fig. 10. The weighting mechanism.

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:

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

where $\texttt{feature}_j$ ($\texttt{feature}_k$) an active feature which refers to $\texttt{offer}_i$ having a positive (negative) `impact`. The scores of `offer-12`, `offer-29` and `offer-16` in Fig. 9 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 system (as shown in Fig. 9, 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 (in case that the time limit given has not been exceeded).

Note that the repudiation (or not) of transactions concerns only the interested users of the system. That is, independently of the system's recommendations, users may proceed or not to perform a transaction. In other words, the system is capable to always recommend the best case (according to the information at hand), but the users are the ones to decide its adoption or not.

5. Discussion and application

The today's success of agent-mediated electronic commerce in creating new markets and reducing various transaction costs will arguably be boosted by the displacement of the current buying and selling processes with new business models [3]. As concluded in a recent survey [6], such a change will take place as software agent technologies come to better deal with issues such as the ambiguity of content, personalization of preferences, complexity of goals, and dynamics of the related environments.

The agents involved in our approach (both `purchaser` and `seller` agents) address the above, the ultimate aim being to provide extra functionality and add more value to the users. This comes out with the following facts: First, they have all the necessary characteristics enabling them being completely personalized, playing the role of an *artificial employee*. More specifically, a user profile is permanently maintained by his/her corresponding agent and can be updated at any time upon the kind of the user–agent interaction. In such a way, our framework allows the agents "live" in an e-market, learn from it and proactively ask its owner's opinion about the initiation of a transaction, whenever a new or promoted product is launched. Second, our `purchaser` agent is enabled with an interactive multiple criteria decision-making tool, allowing the handling of incomplete, inconsistent and conflicting information, as well as the progressive synthesis and comparative evaluation of the offer proposals. Furthermore, we believe that the overall framework presented in this paper is "closer" to a real business model, in that its communication and cooperation protocols allow the implementation of an artificial environment that better represents a real market. We should note here that the usability of our framework is highly associated with the adoption of the proposed e-market from all actors involved. The permanent existence of agents (i.e., maintenance of actors' profiles) implies a certain cost, which has to be weighed against the potential profits arising from a new product offer, an offer for regular customers, etc.

Our system is fully implemented in Java (Windows NT operating system) and runs on the world-wide web. Agents communicate using TCP/IP, while actors interact with them through web interfaces. All transactions carried out use information encoded in XML/EDI format (for more on the advantages of such an approach, see Refs. [4,21]).

A start-up company is currently testing the system's first fully integrated version, the real application being the shares market in Cyprus. In Fig. 11,

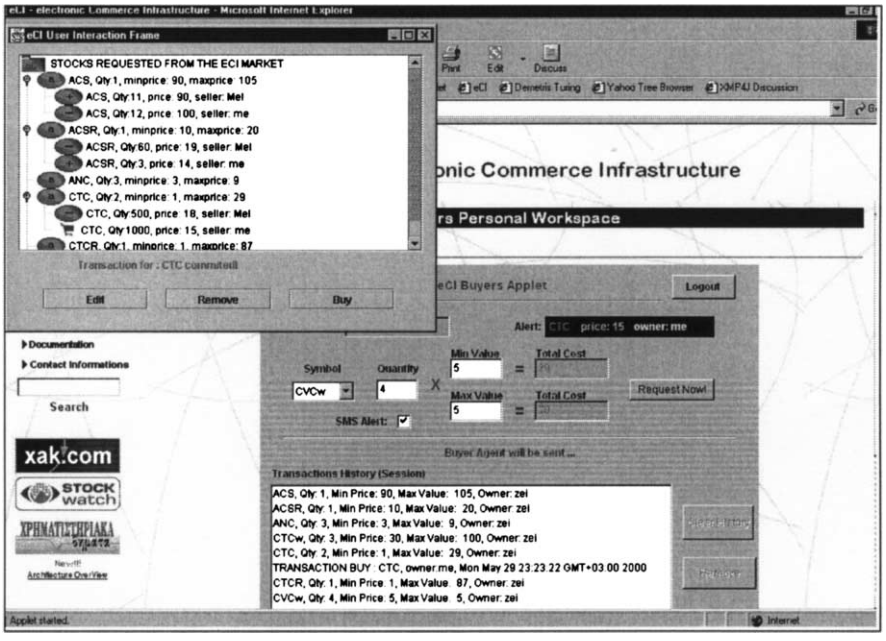

Fig. 11. The Cyprus Shares Market application.

the upper left window shows the offer synthesis graph while the lower window keeps track of a user's activities, that is: (i) requests to buy or sell, and (ii) real transactions taking place between two users which are registered in the e-market. Note that for this very application, a negotiation stage has been also foreseen; this is similar to the approach followed in Kasbah [2] (it is based on a minimum and a maximum share value, which are defined by the user during the initiation of a request).

Fig. 12 illustrates the registration and log-in applets of our system. As shown, users need first to provide all the necessary information to get registered in the e-market (left window) and exploit the system's advanced features, such as notification by email or a message in their mobile phone that a request has been satisfied. In the sequel (or whenever they want to "visit" the e-market again), they need to authenticate themselves through the log-in applet (see Fig. 12, right window).

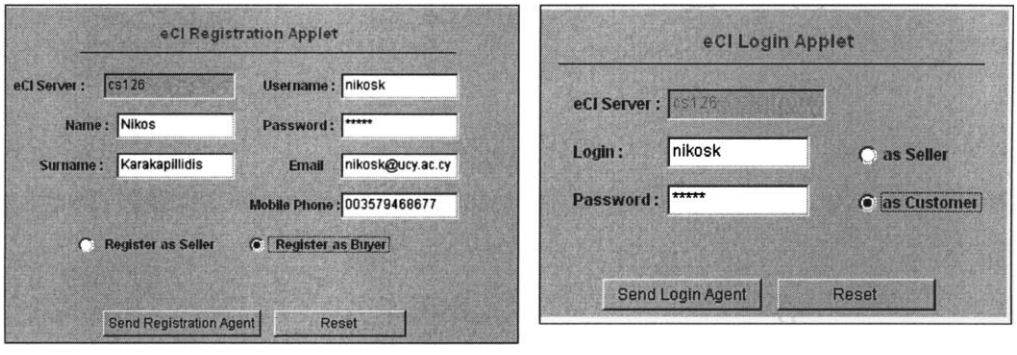

Fig. 12. The registration and log-in applets.

6. Conclusions

This paper has presented a web-based system for agent-mediated cooperation in an electronic market. The main contribution concerns the personalization of the agents involved, the progressive synthesis and multiple criteria comparative evaluation of the alternative proposals, and the specification of (communication and cooperation) protocols attempting to bring real and artificial markets closer. Future work directions concern the integration of a more elaborated negotiation stage (for an overview, see Ref. [10]), by exploiting a multicriteria-based negotiation model that has first been presented in Ref. [14]. 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.

References

[1] Y. Bakos, Towards friction-free markets: the emerging role of electronic marketplaces on the internet. Communications of the ACM 41 (8) (1998) 35–42.
[2] A. Chavez, P. Maes, Kasbah: an agent marketplace for buying and selling goods. Proceedings of PAAM-96, London, UK, 1996.
[3] G. Froehlich, H.J. Hoover, W. Liew, P. Sorenson, Application framework issues when evolving business applications for electronic commerce. Information Systems 24 (6) (1999) 457–473.
[4] R. Glushko, J. Tenenbaum, B. Meltzer, An XML framework for agent-based e-commerce. Communications of the ACM 42 (3) (1999) 106–114.
[5] R. Guttman, A. Moukas, P. Maes, Agents as mediators in electronic commerce. Electronic Markets 8 (1) (1998) 22–27.
[6] R. Guttman, A. Moukas, P. Maes, Agent-mediated electronic commerce: a survey. Knowledge Engineering Review 13 (3) (1998).
[7] J. Hands, M. Bessonov, M. Blinov, A. Patel, R. Smith, An inclusive and extensible architecture for electronic brokerage. Decision Support Systems 29 (4) (2000) 305–321.
[8] N. Karacapilidis, D. Papadias, Hermes: supporting argumentative discourse in multi-agent decision making. Proceedings of AAAI-98. AAAI/MIT Press, Cambridge, MA, 1998, pp. 827–832.
[9] D.E. Knuth, Fundamental Algorithms. 2nd edn., Art of Computer Programming, vol. 1, Addison-Wesley, New York, 1973.
[10] G. Lo, Gr. Kersten, Negotiation in electronic commerce: integrating negotiation support and software agent technologies. Proceedings of the 29th Atlantic Schools of Business Conference, Halifax, NS, 1999 (CD ROM).
[11] A. Mackworth, E. Freuder, The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. Artificial Intelligence 25 (1985) 65–74.
[12] P. Maes, R. Guttman, Al. Moukas, Agents that buy and sell. Communications of the ACM 42 (3) (1999) 81–91.
[13] N. Matsatsinis, P. Moraïtis, V. Psomatakis, N. Spanoudakis, Intelligent software agents for products penetration strategy selection. Proceedings of MAAMAW-99, Valencia, Spain, June–July, 1999.
[14] P. Moraïtis, A. Tsoukias, A multi-criteria approach for distributed planning and conflict resolution for multi-agent systems. Proceedings of ICMAS-96. MIT Press, Cambridge, MA, 1996, pp. 213–219.
[15] H. Nwana, D. Ndumu, L. Lee, J. Collis, ZEUS: a tool-kit for building distributed multi-agent systems. Applied Artificial Intelligence Journal 13 (1) (1999) 129–186.
[16] H. Nwana, J. Rosenschein, T. Sandholm, C. Sierra, P. Maes, R. Guttmann, Agent-mediated electronic commerce: issues, challenges and some viewpoints. Proceedings of Autonomous Agents 98. ACM Press, New York, 1998, pp. 189–196.
[17] M. Papazoglou, A. Tsalgatidou, Business to business electronic commerce issues and solutions. Decision Support Systems 29 (4) (2000) 301–304.
[18] Power, J.D. and Associates. "One Out of Four New Car Buyers Shop the Internet", Internal report, 1998. Available at: http://www.jdpower.com/releases/80914car.html.
[19] B. Schmid, M. Lindemann, Elements of a reference model for electronic markets. Proceedings of HICCS '98, Hawaii, January 6–9, vol. IV, 1998, pp. 193–201.
[20] K. Sycara, D. Zeng, Coordination of multiple intelligent software agents. International Journal of Cooperative Information Systems 5 (2–3) (1996).
[21] D. Webber, Introducing XML/EDI frameworks. Electronic Markets 8 (1) (1998) 38–41.

Before joining the Industrial Management Lab at the University of Patras (Greece), as an Associate Professor, Dr. Nikos Karacapilidis held research and teaching positions at the University of Cyprus (Dept. of Computer Science, Visiting Assistant Professor), the DILITH Lab of EPFL (Switzerland), the AI and Design Group of INRIA-Sophia Antipolis (France), the AI Group at GMD (Germany), and the Dept. of Electrical Engineering of Queen Mary and Westfield College (University of London, UK). He holds a PhD in Engineering (1993) and a BSc in Computer Engineering (1989), both from the University of Patras. His work is published in various OR and AI journals. His current research interests are on the areas of E-Business, Advanced Information Systems, Computer-Supported Cooperative Work, Argumentation and Negotiation Systems, and applications of the above on the web.

Dr. Pavlos Moraïtis had graduate studies at the University of Paris-Dauphine, France (MA in Computer Science and PhD in Computer Science/Artificial Intelligence). He has taught at the University of Paris-Dauphine and the Ecole Superieure d'Informatique–Electronique–Automatique, France. He has also worked as a post-doctoral researcher at the LAMSADE Laboratory, University of Paris-Dauphine, and CNET–FRANCE TELECOM. He was project manager in a research project on agents, supported by a CNET contract. He held Visiting Professor positions at the Technical University of Crete, Greece (Dept. of Production Engineering and Management) and at University of Cyprus (Dept. of Computer Science). Currently, he is an Associate Researcher at the LAMSADE Laboratory, University of Paris-Dauphine, France, and Adjunct Associate Professor at the Technical University of Crete, Greece.