Applied Artificial Intelligence, 17:901–925, 2003

ISSN: 0883-9514 print / 1087-6545 online
DOI: 10.1080/08839510390242259

Taylor & Francis
Taylor & Francis Group

AN AGENT-BASED SYSTEM FOR PRODUCTS PENETRATION STRATEGY SELECTION

N. MATSATSINIS, P. MORAÏTIS, V. PSOMATAKIS, and N. SPANOUDAKIS
Technical University of Crete,
Decision Support Systems Laboratory,
University Campus–Chania, Greece

This paper describes an agent-based system implementing an original consumer-based methodology for product penetration strategy selection in real-world situations. Agents are simultaneously considered according to two different levels: a functional and a structural level. In the functional level, we have three types of agents: task agents, information agents, and interface agents assuming task fulfillment through cooperation, information gathering tasks, and mediation between users and artificial agents, respectively. In the structural level, we have elementary agents based on a generic reusable architecture and complex agents considered as an agent organization created dynamically in an hierarchical way.

The need to combine data and experts' knowledge in order to solve complex and ill-structured decision problems is a major concern in modern marketing and management science. Strategic decision-making procedure is a complex distributed task, involving several actors belonging to different levels of responsibility and having complementary functionalities within an organization. In their article, Meyers et al. (1999) demonstrated the value of implementation knowledge to marketing and innovation researchers and practitioners.

One of the most interesting and difficult decision-making cases is the design and development of a new product. Many experts in this field (see, for example, Nylen 1990; Urban and Hauser 1993; Kotler 1994) have pointed out the importance of successful development of a new product for the viability of enterprises. Therefore, the new product development process needs efficient tools to support their decisions. The aim of decision support systems in marketing is to increase the effectiveness of managers with the

Address correspondence to Nikolaos Matsatsinis, Technical University of Crete, Decision Support Systems Laboratory, University Campus, 73100–Chania, Crete, Greece. E-mail: nikos@ergasya.tuc.gr

support of suitable scientific tools during the different phases of the decision-making process (Simon 1960; Sprague and Carlson 1982).

This paper, which is an extended version of the work presented in Matsatsinis et al. (1999), proposes a distributed, agent-based framework for the implementation of a strategic marketing DSS that supports new product development and market penetration strategy selection. The product development process is collaborative, involving multi-disciplinary functions and heterogeneous tools. From a strategic marketing point of view, the proposed system is the first to introduce a multicriteria methodology application in the product development process (Matsatsinis and Siskos 1999).

The necessity for a system facilitating interaction among many different distributed actors (or organized group of actors), along with an extensive review of all the characteristics considered in the literature (see, for example, Jennings et al. 1996; Sycara and Zeng 1996) as necessary for agency technology use, motivated us towards an agent-oriented approach. These characteristics, which are present in the specific problem of new product development process are: a) the inherent distribution of problem-solving abilities (the agents perform different data analysis, brand choice, and multicriteria analysis methods), data, and information; b) the necessity of flexibility, modularity (agents can appear and disappear in the system without disturbing its functionality), and reusability (customization of agents for new decision makers); and c) problem solving complexity involving coordination between actors expressing different points of view.

The aim of this paper is therefore twofold. The first is to show the added value that emerges from agent technology use in the domain of marketing. The second is to propose a system with particular features (e.g., a multicriteria methodology application in the product development process) for marketing specialists and end-users concerned with the new products development process (e.g., decision makers from production departments, management departments, and other non-marketing expert people) by providing all the necessary details for the system's development.

CONSUMER-BASED METHODOLOGY FOR PRODUCTS PENETRATION STRATEGY SELECTION

To support the product development process, Matsatsinis and Siskos (1999) proposed an original consumer-based methodology (Figure 1). It is based on the use of different models for data analysis, multicriteria analysis, and brand personal choice.

During the market survey, every consumer expresses his evaluations on a set of reference products involved in the research on the basis of a group of criteria. Finally, he is requested to rank the products according to the order of preference. The collection of this kind of data requires a specific questionnaire.

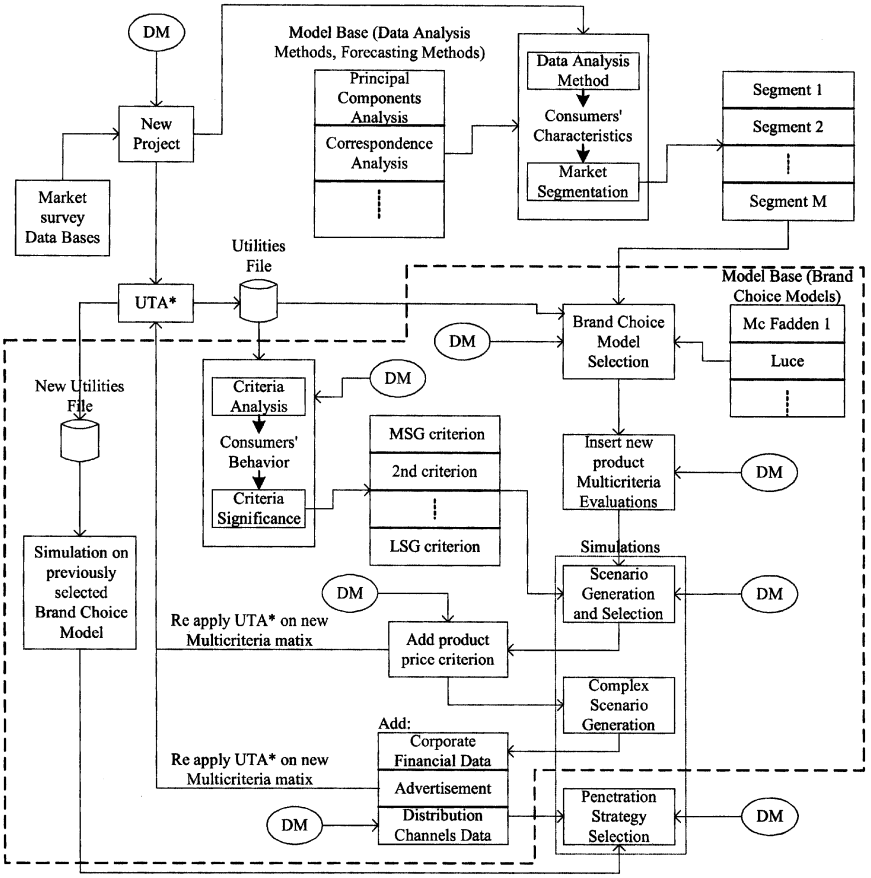

FIGURE 1. Methodological flowchart (source: Matsatsinis and Siskos 1999).

The initial phase of this methodology aims to acquire an overall frame of the particular survey. This is followed by the use of data analysis models in order to determine consumer and market features. This task is called *Market Segmentation.* Market trends are identified through this approach. Concurrently, the multicriteria method UTASTAR (Siskos and Yannacopoulos 1985) is applied to the multicriteria consumer preferences in order to determine the criteria explaining each of the consumer's choices. This method assesses a utility function $u(g)$, which is as consistent as possible with the consumer ranking, where $g = (g_1, g_2, \ldots, g_n)$ is the vector of the criteria on which the products are evaluated. The consumer's utility function is assumed to be additive: $u(g) = p_1u_1(g_1) + p_2u_2(g_2) + \cdots + p_nu_n(g_n)$, where $u_i(g_i)$ is the estimated marginal utility of the criterion g_i, normalized between 0 and 1, and p_i is a weighting factor of the i-th criterion, the sum of weights being equal to one: $\sum_{i=1}^{n} p_i = 1$.

The UTASTAR method estimates the utility function for each consumer separately which is as consistent as possible with the rank order of the products used; the relative importance of the criteria is then derived from this utility model. This preference disaggregation analysis is called *Criteria Analysis*. The use of models of consumer personal choice allows for the market simulation and the calculation of the market shares of the competitive products taking part in the research. This aims at the selection of the most suitable model approach, as close as possible to the real market shares (*Brand Choice Task*). The next step concerns the design of the new product by simulating its introduction into the market using the multicriteria estimations. It is followed by the application of alternative scenarios. With the help of the selected brand choice model, the market simulation and the calculation of the new market shares to be expected (after the introduction of the new product) are performed. This process involves *Scenario Generation and Complex Scenario Generation*. Based on the results of the scenarios application, the choice of the most appropriate penetration strategy for the new product is made. This is the main task and is called *Penetration Strategy Selection*.

THE SYSTEM'S ARCHITECTURE

In Figure 2, we present an agent-based system used by decision makers, who can be corporation board members, each simulating his own scenarios and finally selecting a penetration strategy for a new or an existing product in a board meeting.

Agents' Types, Functionalities, Structure, and Knowledge

An agent's knowledge is acquired during a knowledge acquisition stage, using different domain experts' knowledge, and through interactions with the other agents of the system as well as the human users. In our approach, agents are considered according to two different levels: a *functional* level and a *structural* level. In the *functional level*, we consider three types of agents like in Sycara and Zeng (1996): *interface agents*, *information agents*, and *task agents*.

The functionalities of *interface agents* are those we can find in the literature (Laurel 1997; Sycara and Zeng 1996): initiation of a task, responsibility of system interactions with the user, results presentation to user queries in a way appropriate to the user's profile (e.g., according to the level of responsibility in an organization), and determination of what categories of task agents should be involved, so that a user query is correctly taken into account.

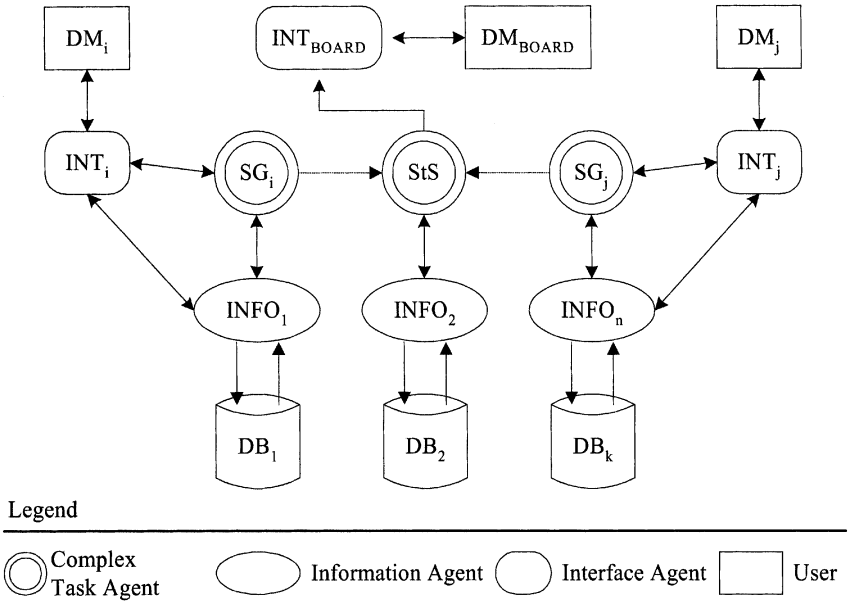

FIGURE 2. Agent-based architecture.

The functionalities of *information agents* are also those we can find in the literature (Knoblock and Ambite 1997; Sycara and Zeng 1996). Their goal is to provide information and expertise on various topics by drawing on relevant information from the system's general database, remote heterogeneous databases in the Internet, other information agents, or interface agents.

Finally, *task agents* specialize in performing specific tasks. They can interact with all types of agents in order to carry out their jobs. These are the most sophisticated agents of our system and they can have an elementary or complex structure. For the application presented in this paper, we conceived different types of task agents (elementary and complex), each corresponding to different *generic tasks* (e.g., perform data analysis, generate a scenario) involved in the methodology presented earlier. We can have several occurrences performing the same specific task (for example, several agents performing data analysis).

We, therefore, have the following types of elementary task agents:

- *Data Analysis (DA) Agent*: Such an agent performs data analysis on an input data set (see Figure 1, e.g., correspondence analysis, principal components analysis, etc.). Such an agent has the knowledge that enables him to choose appropriate data analysis methods, which are effective on any particular input data set. Finally, he can combine and evaluate each applied method's outputs.

- *Brand Choice* (*BC*) *Agent*: Such an agent uses inputted multicriteria tables (see Figure 1) in order to choose the appropriate brand choice model(s) and effectively model the behavior of the consumers that participated in a particular market research (Figure 1, e.g., LUCE, Mc Fadden 1, etc.).
- *UTASTAR Agent* (*UTS*): Such an agent performs the UTASTAR multicriteria method on an inputted market research. He can locate and distinguish multicriteria questions while identifying alternative products used in any market research. Its output is the utility table (referred to as "utility file" in Figure 1).
- *Market Expert Agent* (*ME*): Such an agent selects a market strategy, depending on scenarios and on knowledge that includes corporate information, distribution channels information, etc.

By using these elementary agents, we build complex agents, taking into account the methodology's complex tasks achievement. We consider that by using the complex agent concept to gather together agents involved in some complex task (if the task's nature allows it) achievement, the system's scale and coordination complexity can be decreased, making the application's modeling easier. Actually, coordination, even within a large-scale application, is carried out, either between agents within relatively small-scale groups or between a reduced number of complex agents that are entities of an upper layer. The involved complex agents are:

- *Scenario Generation* (*SG*) *Agent*: Such an agent is composed by at least a DA, a BC, and a UTS agent. He coordinates the scenario and complex scenario generation task (see Figures 1 and 8).
- *Strategy Selection* (*StS*) *Agent*: Such an agent is composed by a BC, a ME, and a UTS agent. He coordinates the penetration strategy selection task.

Agent Organization

Agents can be geographically distributed allowing the interaction with users of different levels of responsibility and involvement in the main problem solving. Agents (elementary and/or complex) interact with each other by means of inter-agent messages. A message is structured in such a way that allows the transfer of the necessary information and semantics between agents for cooperative work accomplishment. Our agent organization allows the following interactions types (for an extended illustrative scenario see "An Example of the System's Operation"):

- New agents are introduced while others "died." An agent will be informed when a new member enters his community or when another leaves. A new agent must present himself to the other agent's community by broadcasting a message with his identity (e.g., his address, his abilities, his

preferences, etc.). This is important from a software-engineering point of view, because it allows modularity, reusability, and flexibility of the system.
- During the problem-solving process, appropriate agent activation dynamically forms an organizational structure that fits with the current goal (a specific task accomplishment, an information retrieval, etc.). In our system, interface agents activate task agents. They can perform or assign a specific task or tasks to a set of cooperating task agents by using appropriate criteria (e.g., their abilities, their availability, and their performance for a similar task in the past, etc.).
- Activities of information agents are initiated, either top down by a user or a task agent through queries, or bottom up through monitoring information sources for a particular information. Once the monitored condition has been observed, the information agent notifies interested agents, by means of messages, of the updated information.
- The interface agents can receive messages from users. After one such agent infers a user's needs from a user request, he starts a new task, giving it an *id*. He then uses information gathered or previously held in order to decide which task agent should be the first to work for this task. While the task agents are carrying out a task, the interface agent that started it monitors its progress.

AGENT ARCHITECTURES

In this section, we present the elementary and complex agent architectures.

Elementary Agent Architecture

The agents, which are used for the presented application modeling, are based on a generic reusable architecture that we conceived, following the general BDI-type philosophy (Georgeff and Ingrand 1989; Rao and Georgeff 1992) and inspired by the different agent architectures presented in the literature (see, for example, Brazier et al. 1997; Sycara et al. 2003; Witting 1992). Different functional agent types have the same basic architecture principles, regardless of the category to which they belong. However, the different modules are more or less sophisticated according to their specific type (e.g., the planning model of an information agent is simpler than that of a task agent). Our agent architecture (Figure 3) is composed of three modules (*Communication, Planning and Reasoning modules*) that intercommunicate through internal message exchanging (called intra-agent messages).

These modules run concurrently. An agent remains idle while no messages arrive to his communication module. As soon as a message arrives, the communication module determines its importance and, after transforming it to an intra-agent message, sends it to the planning module by means of a

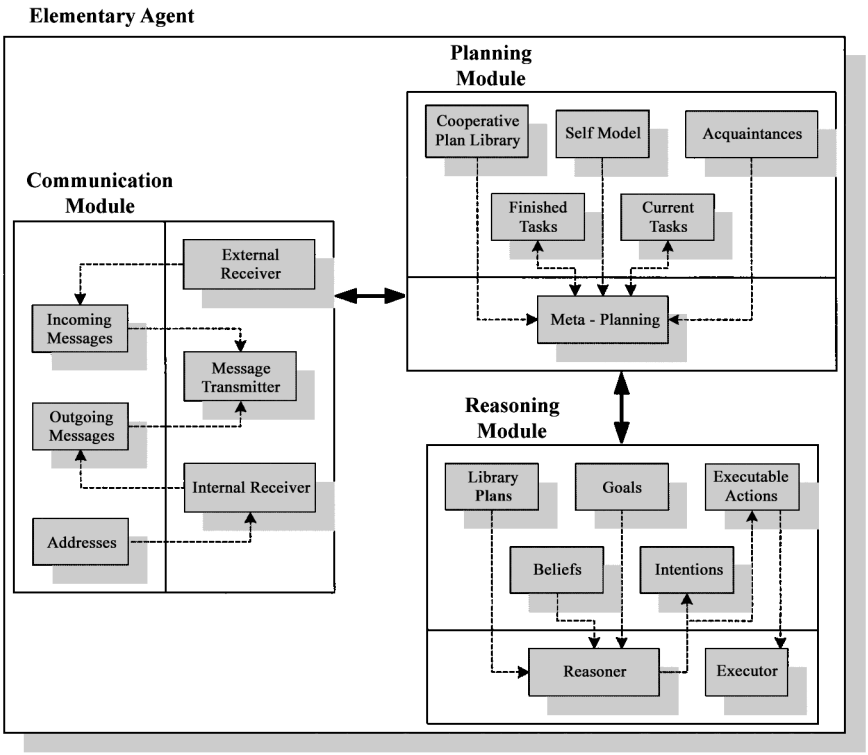

FIGURE 3. Elementary agent architecture view.

message queuing mechanism. All modules adopt this behavior and remain idle while no messages are available for procession. The same intra-agent queuing mechanism facilitates all modules. Thus, intra-agent control is achieved via the intra-message mechanism. This mechanism is implemented at the module class level and it facilitates all types of modules.

As can be seen in Figure 4 (featuring a UML class diagram) the *Agent* class aggregates three *Module* objects and *two FIFO_Priority_Queue* objects (the class names imply their functionality). The first queue facilitates intra-agent control between the communication and planning modules while the other facilitates the planning and reasoning modules. The queues are FIFO priority queues which mean that messages are read in the same sequence that they were sent and those with greater priority are read before those with lesser priority. This intra-message queuing service is implemented by the agent class public methods *updateCommPlanQueue*, *readCommPlanQueue*, *updatePlanReasQueue*, and *readPlanReasQueue* (the first two methods provide intra-message insertion to and retrieval from the queue between the communication and planning modules while the other two accommodate the other queue in the same manner).

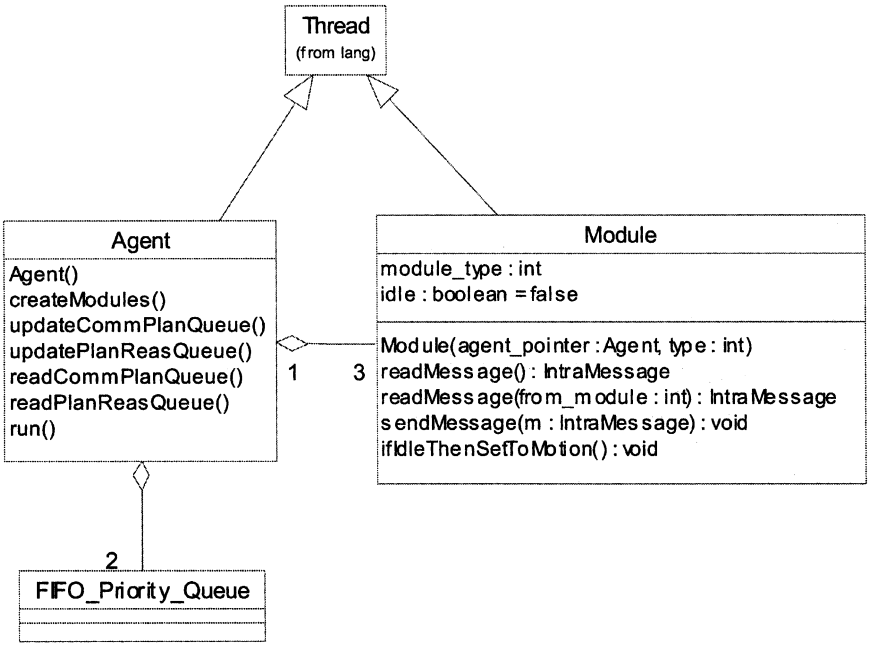

FIGURE 4. Elementary agent class diagram.

The *Module* class subclasses all types of modules. It automates intra-message passing procedure and all types of modules can use the *sendMessage* and *readMessage* methods in order to send or receive intra-agent messages. Those methods are responsible for determining which module is sending to which and for queuing and retrieving messages using the appropriate methods of the *Agent* class. The overloaded method *readMessage* can be invoked with a parameter, which specifies the module from which a message will be received, so that in the case of the planning module, which can receive messages by both the communication and planning modules, the read queue can be specified.

Communication Module

The communication module is responsible for the agent's interaction with his environment. It sends and receives messages, while internally it interacts with the planning module. Its functionality is quite straightforward: An *internal receiver* process transforms each internally queued message to an inter-agent message, adding the receiving agent's address and then stores it to the *outgoing messages* queue. An *external receiver* process realizes the opposite by transforming each received external message to an internal format and writing it to the *incoming messages* queue. A *message transmitter* process monitors the incoming and outgoing queues sending all queued messages either to the planning module or to another agent accordingly.

It has also the necessary "intelligence" to assume the interaction of an elementary agent with his parent agent (case of complex agent).
The inter-agent and intra-agent messages are encapsulated data structures which store the following attributes and support public update and read methods for each one of them:

- *message_id* (an integer that is assigned to the message by the sender so that he can recognize responses which will have the same id).
- *message_type* (the type of the message): available message types for both inter and intra-agent messages (corresponding to the agent interaction types):
  - *Birth* (announcement of a new agent in the community).
  - *Death* (announcement of the destruction of an agent).
  - *Task* (message relevant to a specific task).
  - *Info* (information message).
  - *Task_Init* (messages relevant to the initiation of a task, agents proposing, and agents accepting tasks).
  - *Team* (complex agent team creation message, only intra-agent message type).
- *importance_coefficient* (an integer denoting the importance of the message).
- *message_body* (the information that is to be communicated).
- *task_id* (the id of the task in whose context this message is sent).
- *sender* (the agent that sends the message).
- *receiver* (the agent that is to receive the message).

The *Communication_module* class is featured in Figure 5. It aggregates two queues (for storing incoming and outgoing inter-agent messages), a *Server* (it is a TCP socket server to whom other communication modules can send messages), and an *AddressBook* class (for storing other agents' TCP/IP addresses). Whenever it wants to send a message, it instantiates a *Client* class, which sends the message to another communication module's server.

Planning Module

In our system, agents receive task requests, which they might adopt as *goals*, either from users or from other agents. In order to be able to accomplish effectively all necessary activities that lead to the final goal, agents need to create and execute detailed plans. Since agents are supposed to be "born," act, and "die" in real-world, distributed, open environments, the formulation of a global plan for the whole system would essentially constrain fundamental *agency* characteristics. Hence, our proposed architecture suggests that each agent is internally structured with a distinct *planning module* (Figure 3).

The *planning module* is responsible for the appropriate goal translation into an activity *plan*, which is a sorted list of all the tasks agents need to

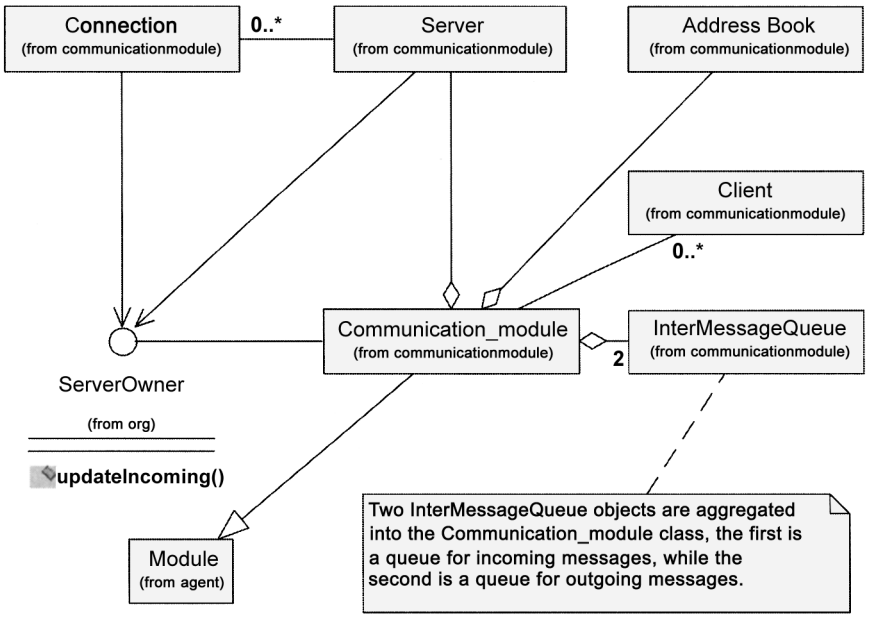

FIGURE 5. Communication module class diagram.

accomplish until the goal is met. The selection of a plan usually actuates a planning module to ask for other agent's cooperation, while tasks might be composed by subtasks, whose timely completion presupposes parallelism. In coherence, an agent's plan accomplishment might partially rely on other agents, those performing parts—subtasks—of the plan.

In this approach, the planning module consists of one distinct process (implemented as a separate thread) and five data structures (Figure 3):

- The *Planner* (process) implements the actual planning algorithms for the agent. Any decision making is necessary to manipulate and revise its data structures (listed below). It is also responsible for the intra-module communication through the intra-message exchange mechanism.
- The *Cooperation Plan Library* is the data structure holding the available plans for those goals the agent is able to achieve. A plan is a direct mapping of a goal (task) to its assorted list of sub-goals (subtasks).
- The *Finished Tasks* is a structure maintaining a task history log for each agent. It holds the list of all the tasks accomplished by the agent.
- The *Current Tasks* is a structure monitoring the tasks under accomplishment, their state (running, suspended), as well as information for the plan to which each one of them belongs.
- The *Self Model* is a structure holding information about the agent itself. In other words, it's an abilities structure for internal use by the planner.

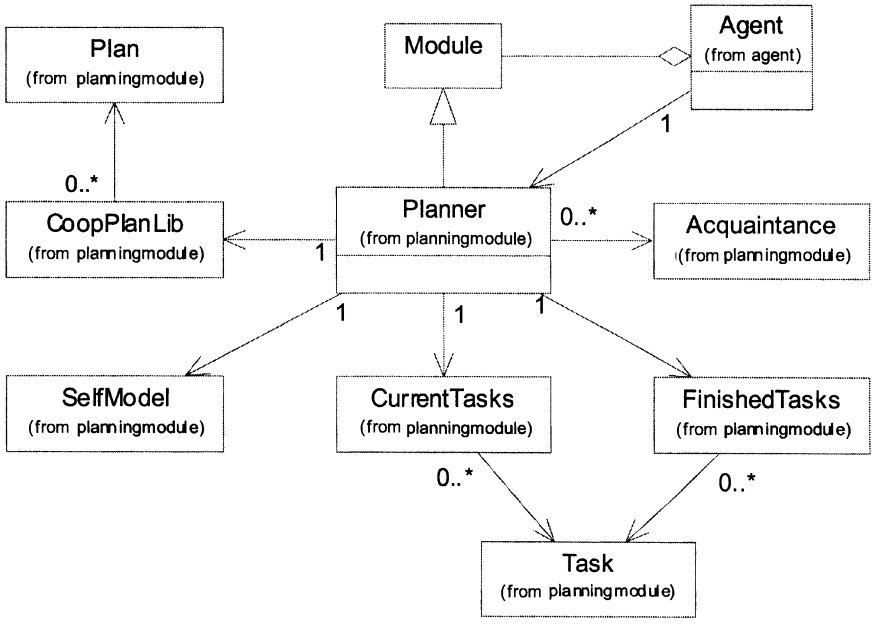

FIGURE 6. Planning module class diagram.

- The *Acquaintances* is the structure where the planner keeps information about other possibly useful agents within the system. Whenever the planner needs to ask for cooperation it first checks for the appropriate acquaintance through this structure.

The planning module intercommunicates with both the communication and reasoning modules, something expressing its intrinsic administrative role. Following the general internal communication policy, the planning module gets active when an *IntraMessage* appears into one of the queues. Messages are processed by the planner process, which is responsible for module reaction on each request, information arrival, domain change, etc.

Figure 6 presents an implementation modeling in UML for the planning module. Apart from the six components already described above, there are another two supporting structures, one modeling a plan, and one modeling task.

Reasoning Module

Apart from the planning ability, it is essential for agents to make rational decisions. Our proposed agent architecture suggests a distinct module to handle all the listed problems (Figure 3).

The *reasoning module* makes decisions on *goal* or *intention* adoption, actions execution, and monitors the allocated tasks execution progress. For

the implementation of the above concepts, reasoning module is using one process (the reasoner) and five data structures:

- The *Reasoner* implements the reasoning algorithms. It uses the module's structures for deciding which goals should be adopted, generating intentions and determining which actions (derived from intentions) should be executed or suspended.
- The *Executor* executes the actions in the executable actions list.
- The *Plan Library* is the structure containing pre-compiled plans of actions necessary for complex tasks' decomposition and achievement.
- The *Beliefs* of an agent are dynamically changing structures, representing the agent's knowledge about the environment in which the plan is executed.
- The *Goals* of an agent is a structure holding information on tasks that will be accomplished. The goals structure implements a priority queue, where higher priority tasks (or subtasks) are handled first by the reasoner. Our goals structure substitutes *desires* from the generic BDI architecture (Rao and Georgeff 1992).
- The *Intentions* of an agent is a structure containing those plans of actions that have been chosen for (eventual) execution in order to achieve the defined goals.
- The *Executable Actions* is a structured list containing the primitive plans (sequences of primitive actions) to be executed by the executor. These plans are a subset of those contained to the intentions' structure and are chosen according to several criteria, relative to a specific context. The implementation of the list is an FIFO queue, where the first action inserted is executed first.

The reasoning module intercommunicates internally only with the planning module. Following the general internal communication policy, the reasoning module gets initially active when an *IntraMessage* appears into the queue it shares with the planning module.

Figure 7 presents an implementation modeling in UML for the reasoning module. There are also two implementation supporting structures: one modeling a plan reduction formula (*RedLayout* class) and one for task modeling.

Complex Agent Architecture

Complex agents can belong to the three functional types defined earlier. The architecture of a complex agent is similar to the one of an elementary agent. Therefore, he is composed of the same three modules (*Communication, Planning and Reasoning module*) which intercommunicate through internal message exchanging. The intra-agent control (interaction between the three components) is the one of the elementary level.

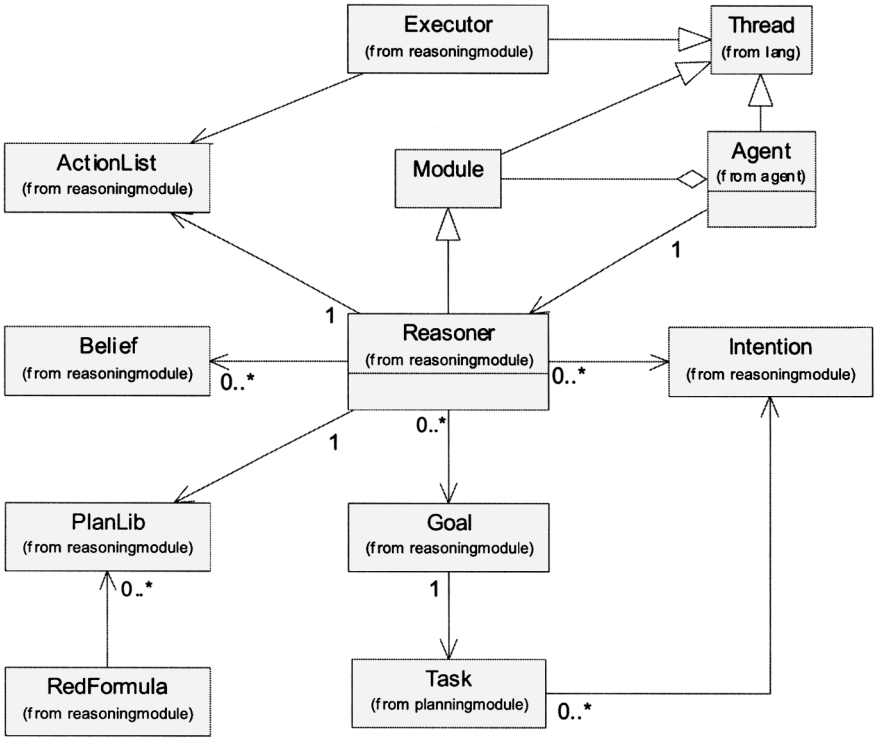

FIGURE 7. Reasoning module class diagram.

The difference is situated in the structure of the reasoning module. The group of agents (elementary and/or complex) which compose it assumes its role. The task achievement of an agent (parent) developed in *n-layer* is, therefore, the result of the set of agents' (his descendants) cooperation belonging to the previous $(n-1)$ *layer*. The reasoning module could be therefore considered as an *agent organization*.

The interaction between a complex agent's reasoning and planning modules (like in the elementary level) of an $(i+1)$ layer agent is established through the i layer agents that are components of the reasoning module of the $(i+1)$ layer agent (for example, the *Brand Choice* Agent sends the result of his work to the planning module of the *Scenario Generation* Agent, Figure 8). In this context, an inter-agent message sent by an i layer agent to an $(i+1)$ layer agent is transformed to an intra-agent message of the $(i+1)$ layer agent. The organization of reasoning module agents is dynamically generated as presented earlier. Its role is to achieve any tasks(s) allocated by the planning module. The agent's organization generation process is initiated by selecting appropriate agent(s) (according to the task's nature) chosen by the planning module. An illustrative scenario of the mechanism that supports this operation will be provided later in this chapter. Agents can be of a different

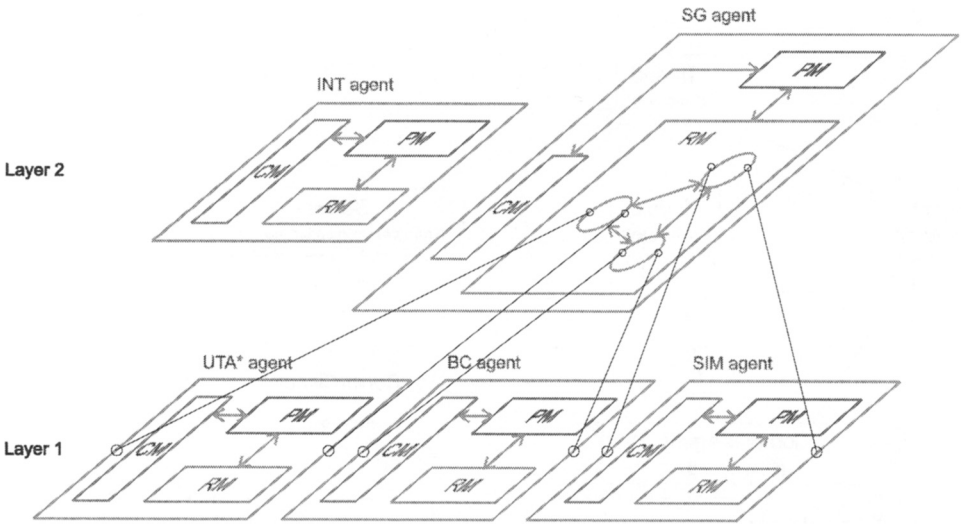

FIGURE 8. Layers of an SG-complex agent structure.

nature (e.g., static, mobile) not necessarily implemented in the locality of the parent agent, but they are, however, aware that they have the same parent. In Figure 8, a UTA* agent, a BC agent, and an SIM agent compose the reasoning module of a complex SG agent. The UTA* agent provides the multicriteria analysis service (using his knowledge on analyzing a market research and selecting multicriteria question), the BC agent selects a brand choice model for the multicriteria analysis (using his knowledge base on model selection), while the SIM agent simulates scenarios for the user (using either its knowledge base or user input provided via an INT agent to the SG agent).

Communication and planning modules have exactly the same structures and functionalities as in the elementary level. The difference is in the *ComplexAgent* class (see Figure 9). Instead of a third *Module* object (what would become its reasoning module), the agent utilizes a *Server*, an *InterMessageQueue*, and instantiates *Client* classes.

Whenever it recruits an agent of a lower level, it informs it that its TCP/IP address has changed and it sends him the address of the server that it utilizes (instead of the one that listens for its communication module and that does not cease to exist). This TCP/IP address exists only for the agents of the lower level. The other agents continue to contact him normally via its communication module. The inter-agent messages that arrive to this server are transformed to intra-agent messages in a transparent way and are subsequently placed in the queue for the planning module. Correspondingly, intra-agent messages that are queued by the planning module for the reasoning module are transformed into inter-agent messages and are sent via

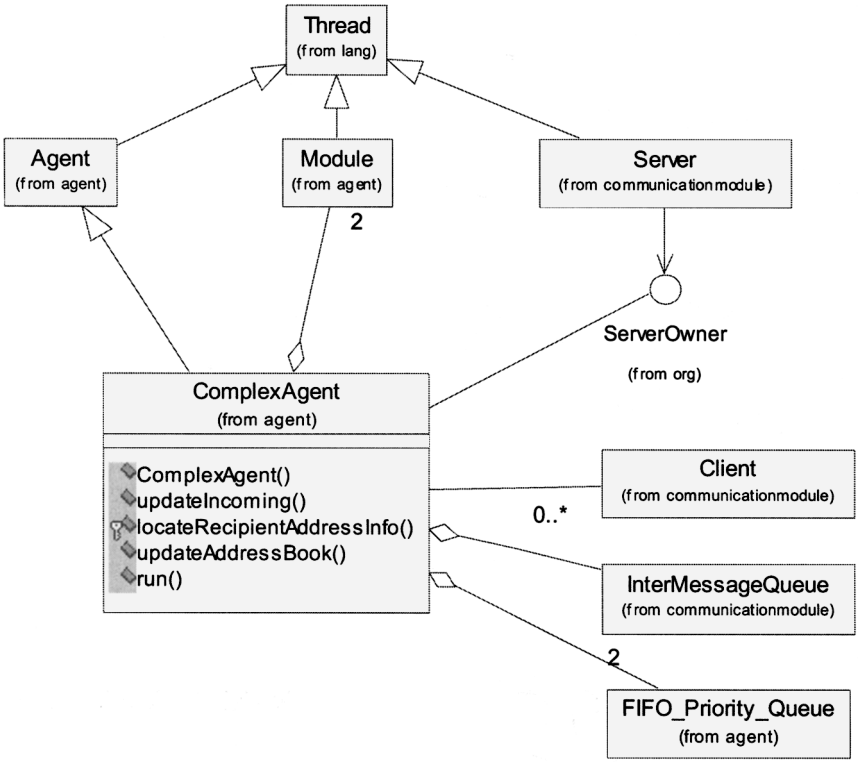

FIGURE 9. ComplexAgent class diagram.

a *Client* to the corresponding agent of the lower level. In order to facilitate this procedure, the complex agent uses an AddressBook object, which is updated by the communication module via a special intra-message that it receives from the planning module and contains the TCP/IP addresses of the agents in the lower level.

In Figure 10, a complete scenario of a complex agent's team creation is provided. The sequence starts after the planning module has decided what other agents will be needed for task execution, has sent to them the appropriate proposals via *Task_Init* inter-agent messages, and they have all answered positively. At that point it sends to the communication module the intra-message of type *Team* making the communication module update the complex agent's internal structure with the addresses of the agents of the team and send messages to them denoting the new address of the complex agent.

After technically presenting the complex task agent, we can say that two processes facilitate a complex task achievement. A top-down process assumes that the task's decomposition in several subtasks is achieved across the different layers, while a bottom-up process performs the synthesis of different solutions proposed at different layers. We can have complex information

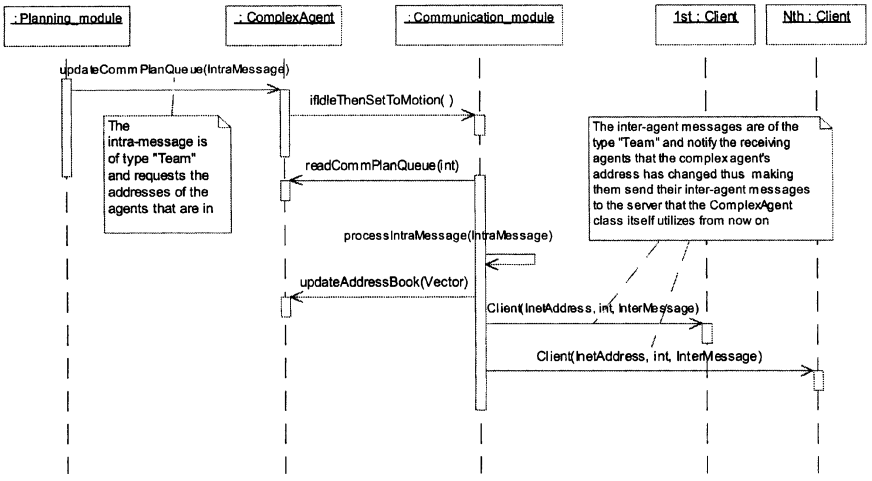

FIGURE 10. New team creation scenario.

agents when an information retrieval must be accomplished through the achievement of several specific information-gathering goals. Different specialized information agents representing layers of a complex information agent can take these goals into account. We can also have a complex interface agent able to take into account (through his elementary interface agents) the different points of view of board members during a distributed decision-making process.

AN EXAMPLE OF THE SYSTEM'S OPERATION

The system is running on Windows 2000. The Windows execution environment is necessary because many methods (UTA*, data analysis methods, etc.) used by the different agent type reasoning modules have been implemented using the Visual Basic programming language and are in the form of dynamic link libraries (DLLs). The agent architecture framework, though, has been implemented in pure Java and thus can be executed under any Unix-like operating systems, such as Solaris, Linux, etc., as well.

We proceed with an example of the system's operation, which shows how the proposed methodology is implemented by our system. The questionnaire that was used as an input is the same as in Matsatsinis and Siskos (1999), which was a real-world case study.

The agent's interactions are presented in sequence diagrams. For presentation purposes, the different agent types are symbolized as classes and the inter-agent messages as class methods. Thus the UML notation for sequence diagrams (Eriksson and Penker 1998) can be used adequately to present the inter-agent interactions that occur during the systems execution. The reader

should note that inter-agent interactions are invisible to the decision maker that interacts only with the appointed INT agent via a Java applet.

In this example, a decision maker attempts to decide on a market penetration strategy for a new product, a fruit juice in this example, which is currently under development. Related field market surveys have been accomplished and a general database has been built, containing all the collected data (questionnaires, consumer answers, etc.).

As soon as the decision maker (user) contacts the system, an interface (INT) agent initiates a project and she/he is asked to define the project's goal and constraints on data manipulation ("Request to simulate a new product penetration strategy" task, see Figure 11).

After having the initial information, the interface agent is able to delegate the task to the competent agents of the system. In our example, interface agent delegates the job to a Scenario Generation (SG) agent, who is immediately sending collaboration requests to a Data Analysis (DA) agent, an Utastar (UTS) agent, and a Brand Choice (BC) agent via appropriate *Task_Init* inter-agent messages. The contacted agents reply with an affirmative or negative *Task_Init* message. In the case of an over-employed agent that declines the SG agent request, the SG agent's planning module selects another agent to whom he sends the *Task_Init* message. As soon as the team

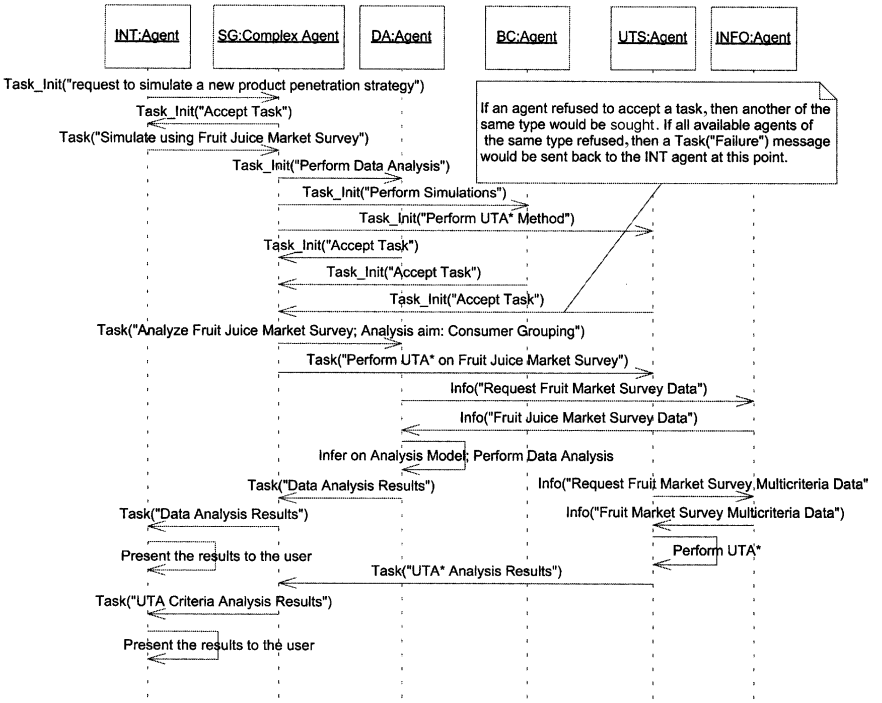

FIGURE 11. Example, agents' interactions (1).

is created, the SG agent sends to each agent the corresponding subtask via *Task* messages.

Depending on the decision maker's constraints, the DA agent requests the necessary data for analysis from an information (INFO) agent. The result he gets is the filtered data spreadsheet upon which a data analysis method is selected and applied. Methods selection is made according to a user's analysis target. The first data analysis target in the fruit juice example is consumer grouping, so *principal components analysis* is chosen. The method's outcome, a eigenvalues table, a correlation table, and a variables interpolation table are sent to the INT agent (via the SG agent who intervenes on all agent interactions thus coordinating the team's work) in order to be presented to the user as the *consumers characteristics*.

Concurrently (with the DA agent), the UTS agent requests from an INFO agent the constrained data multicriteria table, on which he applies the Utastar method in order to gain the utilities files and the *criteria significance* tables, which are forwarded to the INT agent in order to be presented to the user.

At this stage, the INT agent presents the consumer's characteristics and the criteria significance tables (that is, both the outcomes of data analysis and Utastar methods) to the decision maker who is now able to select the *market shares* he is interested in. When the market shares spreadsheets are prepared, the INT agent forwards them to the SG agent (''Prepare market shares for the chosen target market'' task, see Figure 12) who, in turn, forwards them to the BC agent who is responsible for the appropriate brand choice model selection for each one of the market segments. The aim of all brand choice models is to find out the preferences of the consumers as far as the products of the market are concerned. The determination and selection of the most appropriate model depends on two factors. The first one is the width of utilities allocated by the consumers of the segment, and the second is the type of distribution of these utilities. It is the BC agent's duty to reason and select the most applicable model for each market segment.

In continuance, the INT agent presents to the decision maker the multicriteria matrix with the mean values for each related product. The user is able to make changes on that table and declare a range for those criteria values she/he wants to experiment with. Finally, those user's preferences (modified multicriteria tables) are forwarded to the BC agent (''Request simulation'' task, see Figure 12).

The BC agent applies the previously selected brand choice models to the market segments, implementing the market simulation that way. The BC agent simulates market for all possible combinations among criteria values in the user-defined range. Market simulation scenarios are presented graphically to the decision maker. Depending on the market shares, she/he decides which of the scenarios will be used for the complex simulations. Those (user

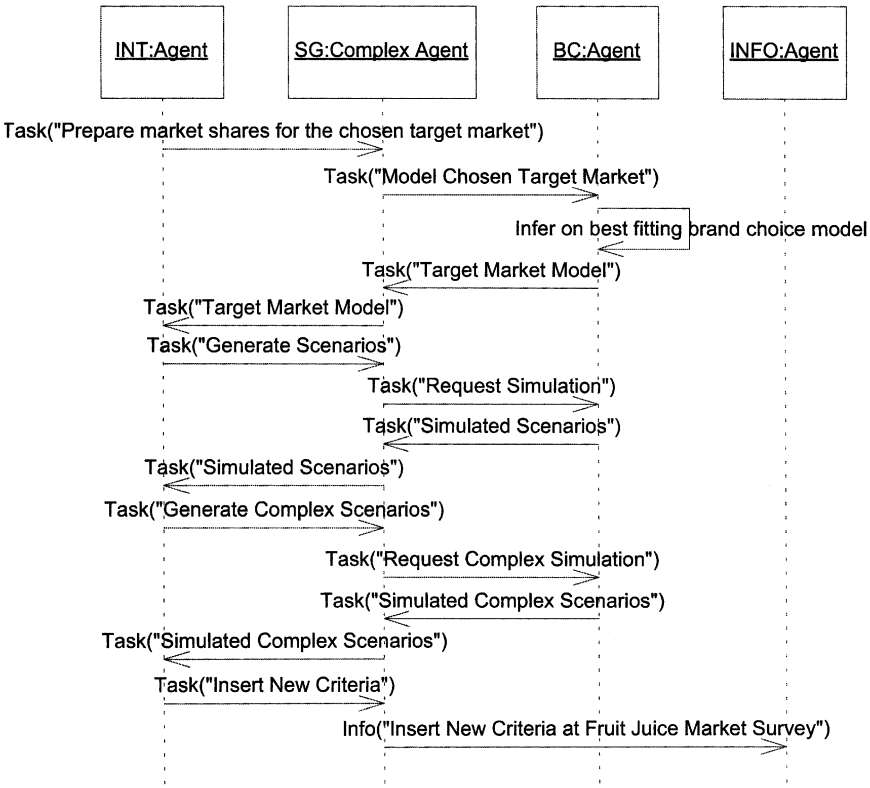

FIGURE 12. Example, agents' interactions (2).

selections) are transferred back to the BC agent ("Request complex simulation" task, see Figure 12), who introduces price as an independent variable and implements the complex market simulations procedure. The results are presented to the decision maker (through the GS and INT agents again) and she/he decides which are the most practical for the new product development. For the selected scenarios, the user has the ability to try new criteria introduction. If she/he introduces new criteria, they will be sent to the INFO agent with the request for new multicriteria tables construction (one for each consumer, see last message in Figure 12).

If the decision maker is not satisfied (for any reason) with the strategies selected, she/he might reintroduce criteria and request another market analysis (repeat the procedure described here from any desired point forward).

After all decision makers (probably coming from different enterprise departments like marketing, development, sales, etc.) have edited their scenarios and saved them on the general database, a board member can select them in order to simulate them using the inference abilities of the ME agent. So, in the sequence presented in Figure 13, a board member (alone or during

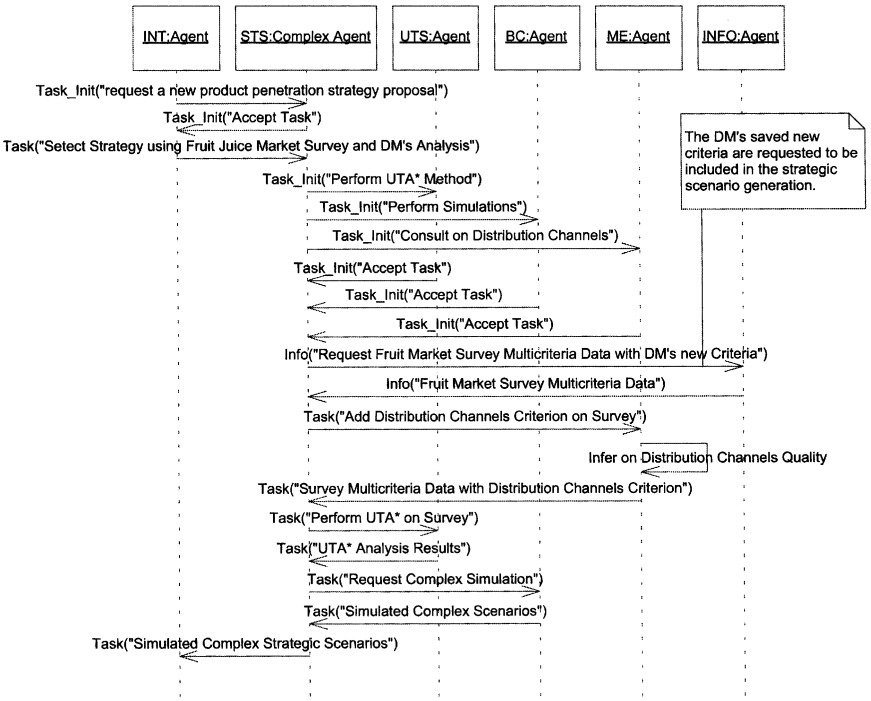

FIGURE 13. Example, agents' interactions (3).

a board meeting) requests to simulate the scenario of one or more decision makers, and introduce the enterprises' distribution channels quality criterion. An available strategy selection (STS) agent asks the EM agent to introduce the new criterion to the multicriteria matrices in the general database. Then he sends the consumer multicriteria matrices to a UTS agent, requesting another Utastar method application and the composition of new utilities files. As soon as those new utilities files are available, they are sent to the BC agent who will perform the simulations and return the latter to STS in order to (through the INT agent) be presented to the board member (sorted by the market shares that they provide to the company).

RELATED WORK AND CONCLUSIONS

This paper presented an agent-based system to support new products penetration strategy selection process for the first time in a real world distributed context with use of multicriteria methodology.

Among existing systems in the literature, the one that comes closest to the capabilities of the proposed system is the MARKEX system (Matsatsinis and Siskos 1999). Besides, the proposed system is actually an improved version of MARKEX. The proposed system has been redesigned and implemented

based on the technology of intelligent agents. The most important differences between them can be pinpointed in the artificial intelligence techniques used (expert system versus intelligent agents), in the design (structural versus object–oriented), the method of operation (the methodology is externally implemented and follows a standard procedure, whereas in the new system the operation concerns the implementation of all the processes by the intelligent agents based in the objectives set by the decision maker), and, finally, in the participation level of human decision makers in the process (the system operates only with the presence and under supervision of the decision maker, whereas in the new system the participation of the decision maker has been limited to the initialization of the problem with the establishment of the objectives, the solution of partial problems, and the selection of the proposed solution).

In comparison to the CSAS system (Liberatore and Stylianou 1995), they both handle the new product development process and are the only ones that use multicriteria methodologies (AHP and UTASTAR, respectively) and data analysis methods. However, they use different artificial intelligence techniques (expert systems versus intelligent agents) and different methodologies (customer satisfaction versus consumer behavior analysis). In addition, the proposed system covers the consumer's purchase decision process, sales forecasting, and incorporates the ability to apply scenarios and simulations regarding the reaction and behavior of the market.

The TeleDSS system (Huang et al. 2000) considers the development of an agent-based framework for collaborative product development that pays more attention to the use of workflow management as a mechanism to facilitate the teamwork in a collaborative product development environment rather than supporting the decision of new product development. The system proposed by Haque et al. (2000) provides decision support for project managers and engineers during the early phases of new product development in a concurrent engineering (CE) environment. The authors suggest a process that handles the application of concurrent tasks in the development of new products but leave out the design and development of new products and the process of strategic marketing decision making. Liang and Huang (2002) propose an agent-based system of collaborative information and a solution procedure for designing with modules to develop modular products. This system also focuses in the support of processes for new product development (modular developing). Chung et al. (2003) investigate the use of ontologies, agents, and knowledge-based planning techniques to provide support for adaptive workflow or flexible workflow management, especially in the area of new product development within the chemical industries. A general conclusion is that the above-mentioned systems approach the new product development problem from the perspective of supporting the development processes, whereas the proposed system attempts to approach the problem

from the perspective of new product design and development based on the analysis of consumers' behavior.

The hybrid system (Li 2000) has been developed to: provide a logical process for strategic analysis; support group assessment of strategic marketing factors; help the coupling of strategic analysis with managerial intuition and judgment; help managers deal with uncertainty and fuzziness; and produce intelligent advice on setting marketing strategy. The proposed system supports the following three stages of marketing strategy development process: strengths, weakness, opportunities, and threats analysis (SWOT); portfolio summary of the current product/market status; and setting marketing objectives and strategy. This system deals only with the issue of marketing strategy and in this its subject is significantly superior to the proposed system. However, the latter also handles the selection of the most appropriate penetration strategy, indicated by the anticipated market shares produced by the simulation of the market's behavior under different circumstances. In the same way, the following systems deal only with the solution of partial problems or simply support the decision-making process. ARISTOTE (Pinson and Moraitis 1996) aims at helping corporate managers address the feasibility and coherence of long-term plans of actions. The system supports a framework for the cooperation of decision makers; cooperation that eventually will facilitate decision-making. An architecture for intelligent agent-assisted decision support system development has been proposed by Wang (1997). The system supports the collaborative decision-making process and the marketing planning decisions. CMA (concurrent marketing analysis) is a new marketing DSS and information management approach (Schwartz 2000), which allows managers to conduct concurrent, interrelated analysis on their decision problems. It consists of a new approach to marketing decision problems, which has the potential to move marketing decision support systems to a new level of integration and effectiveness. Ha et al. (2002) propose a dynamic customer relationship management (CRM) model utilizing data mining and a monitoring agent system to extract longitudinal knowledge from the customer data and to analyze customer behavior patterns over time for the retailer. In any case, one can find an extended survey of intelligent decision support systems in marketing in Matsatsinis and Siskos (2003).

From the multi-agent systems point of view, the proposed system is built by using a generic reusable agent architecture, where agents are considered simultaneously in two levels: a functional and a structural level. In the functional level, we have three types of agents: Tasks, INT, and INFO agents; while in the structural level, we have elementary and complex agents. The structural level distinction enables our system to support many different points of view in different levels of abstraction in a decision-making procedure, which is crucial for strategic marketing, while it facilitates interactivity between individual decision makers and an organized group of decision makers.

The difference with other works (see, for example, Brazier et al. 1995; Jennings et al. 2000a; 2000b) is, like in Sycara's and Zeng's (1996) work, the differentiation between three types of agents in the functional level, allowing efficient operation for real complex tasks achievement involving coordination, information gathering, and user interaction. Compared to Sycara and Zeng (1996), the difference is that we introduce the structural level consideration for the three types of agents (even if we have only implemented complex task agents in this application). We consider that by using the complex agent concept, thus gathering together agents involved in some complex task (if the task's nature allows it) achievement, the system's scale and coordination complexity can be decreased, therefore, simplifying application modeling. Actually, coordination, even within a large-scale application, is carried out either between agents (elementary and/or complex) within relatively small-scale groups, or between a reduced number of complex agents, components of a distributed system, or components of an upper layer. In the latter case, coordination is carried out by intra-agent control primitives assuming interaction between different layers of agents of a complex agent.

In conclusion, we have presented an agent-based system for new product penetration strategy selection. This issue is faced in its real-world distributed nature and this is due to the advantages provided by agent technology use. This system can be used by different enterprises producing different types of products in order to simulate and evaluate the expected impact in the market of the products they plan to launch. The use of the system can be done before these products have been really produced in order to make the best choice on a commercial level and thus avoid a probable failure, which can often be fatal for an enterprise.

REFERENCES

Brazier, F. M. T., B. M. Dunin-Keplicz, N. R. Jennings, and J. Treur. 1995. Formal specification of multi-agent systems. In *Proceedings of First International Conference on Multi-Agent Systems* (*ICMAS'95*), pages 25–32, San Francisco, CA.

Brazier, F. M. T., B. M. Dunin-Keplicz, N. R. Jennings, and J. Treur. 1997. DESIRE: Modeling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*. Special issue on Formal Methods in Cooperative Information Systems: Multi-agent Systems 6(1): 67–94.

Chung, P. W. H., L. Cheung, J. Stader, P. Jarvis, J. Moore, and A. Macintosh. 2003. Knowledge-based process management: An approach to handling adaptive workflow. *Knowledge-Based Systems* 16:149–160.

Eriksson, H., and M. Penker. 1998. *UML Toolkit*. New York: John Wiley & Sons.

Georgeff, M. P., and F. F. Ingrand. 1989. Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (*UCAI-89*), pages 972–978, Detroit, MI.

Ha, S. H., S. M. Bae, and S. C. Park. 2002. Customer's time-variant purchase behavior and corresponding marketing strategies: An online retailer's case. *Computers & Industrial Engineering* 43:801–820.