A Negotiation Model in Agent-mediated Electronic Commerce

Mokdong Chung
Department of Computer Engineering
Pukyong National University
599-1, Daeyeon-3Dong, Nam-Gu,
Pusan, Korea, 608-737
mdchung@pknu.ac.kr

Vasant Honavar
Department of Computer Science
Iowa State University
210 Atanasoff Hall
Ames, Iowa 50011-1040, USA
honavar@cs.iastate.edu

Abstract

Today's first generation shopping agent is limited to comparing merchant offerings usually on price instead of their full range of attributes. Even in the full range comparison, there is not a good model which considers the overall features in the negotiation process. Therefore, the negotiation model needs to be extended to include negotiations over the more attributes. In this paper, we propose a negotiation model in the agent-mediated electronic commerce to negotiate over prices, product features, warranties, and service policies based on utility theory and simple heuristics. We will describe a prototype agent-mediated electronic commerce framework called Pmart. This framework provides the software reuse and the extensibility based on the object-oriented technology. It is implemented on Windows-based platforms using Java and CORBA for the network transparency and platform independence.

1. Introduction

The combination of electronic commerce and autonomous intelligent agents is potential to deliver enormous economic benefits. In many web sites, people can go to find, buy, and sell goods such as OnSale [17], UCE [22], and Amazon [1]. But none of these sites has the notion of an autonomous agent which will actually do the work of negotiating to find the best possible deal on your behalf [4]. Furthermore, today's first generation shopping agent is limited to comparing merchant offerings usually on price instead of their full range of value. Even in the full range comparison, there is not a good model which considers the overall features in the negotiation process appropriately. Therefore, the negotiation model needs to be extended to include negotiations over the more attributes.

Agent mediated electronic commerce enables cheap negotiation between buyers and sellers on the details of an individual transaction. Auctions provide a well-defined and simple framework for negotiation between self-interested buyers and sellers [19]. And they discover the optimal price for a good through the bidding action of self-interested agents. Although auctions traditionally allow negotiation over price alone, auction mechanisms can be extended to include negotiations over the full range of attributes. Consumers manage their own negotiation strategies over an extended period of time this is where agent technologies come in [13]. Bidding agents make tradeoffs between features and price within an auction framework.

Therefore we will suggest, in this paper, an agent-mediated electronic commerce framework called *Pmart* (***Pukyong-mart***) based on the multiagent to negotiate over prices, product features, warranties, and service policies. And we will suggest a negotiation model which is based on special knowledge and general knowledge at the same time. The former is based on the MAUT (Multi-Attribute Utility Theory) [14] and the latter is based on the previous purchase history and the simple heuristics [8]. This framework provides the software reuse and the extensibility based on the object-oriented technology and software components technology. Therefore, we could make different types of electronic commerce systems by simply changing the contents of Pmart External that is the variant part of Pmart.

The paper is organized as follows: section 2 is concerning related works, section 3 a negotiation model of Pmart, section 4 the design and implementation of Pmart, and section 5 the conclusion and the future work.

2. Related work

MIT Media Lab developed an agent based market called Kasbah [3,11] where users may assign the task of buying or selling a specified good to an agent which then performs negotiation and settlement of deals according to the users' choice. And this is served a useful platform for experiments with groups of users. But this system did not intend to be a general market infrastructure and does not offer distribution nor interaction models. After buying

0-7695-0933-9/00 $10.00 © 2000 IEEE

agents and selling agents are matched, the only valid action in the distributive negotiation protocol is for buying agents to offer a bid to sellers. Selling agents respond with either an answering "yes" or "no". Given this protocol, Kasbah provides buyers with one of three negotiation "strategies": anxious, cool-headed, and frugal. This negotiation takes the form of multi-agent, bilateral bargaining but uses simple raise or decay functions.

Tete-a-Tete [12,13] is an agent-mediated comparison shopping system that allows consumers to consider dimensions other than price in their buying decisions for complex products. This system provides a unique negotiation approach to retail sales. Unlike most other on-line negotiation systems which competitively negotiate over price, Tete-a-Tete agents cooperatively negotiate across multiple terms of transaction, such as warranties, delivery times, service contrasts, and return policies. This system uses integrative negotiation interaction model, together with a decision support module, creates an improved online shopping environment for both consumers and merchants. The decision support module uses MAUT and product customization which is based on distributed constraint satisfaction. But this has some problems basically related to inferring the probabilities in the decision-making problem under uncertainty.

AuctionBot [26] is general purpose internet auction server at University of Michigan where users create new auctions to buy or sell products by choosing from a selection of auction types and specifying its parameters. And buyers and sellers can then bid according to the multilateral distributive negotiation protocols of the created auction. In a typical scenario, a seller would bid a reservation price after creating an auction and let AuctionBot manage and enforce buyer bidding according to the auction protocols and parameters.

Magnet [21] was developed by University of Minnesota. The goal of the Magnet project was to develop a semantic model for the integration of planning, contracting, scheduling, and execution in a multi-agent market domain.

The SICS MarketSpace [7] prototype was developed in Intelligent Systems Lab, Swedish Institute of Computer Science using Java. And it consists of a personal assistant allowing the user to describe his/her interests. Two agents augmented web shops. It provides a directory service which allows user and service agents to register interests and find agents with matching interests.

A variety of architectures have been proposed for electronic commerce and multi-agent automated contracting [7,11,13,21,26]. And several protocols have been developed and proposed to support automated contracting and negotiation among multiple agents.

But most these systems competitively negotiate over only price, and especially systems such as Kasbah, AuctionBot, Magnet, SICS MarketSpace do not have sophisticated negotiation strategies like Pmart. P-mart agents cooperatively negotiate across multiple terms of transaction, such as prices, warranties, delivery times, service contrasts, and return policies.

3. A negotiation model of Pmart

We have two alternatives in negotiating between buyer and seller agents. The first approach is using only *MAUT* consistently. The second is using *MAUT* and if fail even after a few trials, say *numS*, then resorting to the previous purchase history data base and simple heuristics. This history database contains information about successfully purchased transaction so far. This may be general knowledge on purchasing a good. The reason why we want to select the second approach is as follows:

1) The negotiation starting from special knowledge (MAUT) to general knowledge (purchase history and simple heuristics) is sound in the general inference mechanism [20].
2) If the negotiation fails even after a few trials of MAUT-based negotiation, we can infer this MAUT-based negotiation might be less than the general knowledge-based negotiation. Because general knowledge might represent general purchasing behaviors of all clients so far, and also because the utility function which was assessed by the buyers are not consistently reasonable, resorting to this general knowledge might be reasonable.

Therefore, we can provide hybrid, two-fold decision making process that involves the usage of special knowledge in a first decision step based on MAUT, and general knowledge in a second decision step based on the purchase history database:*CaseBase* and simple heuristics. In such a model, we propose a kind of *case-based reasoning* to observe behavior in previous similar situations.

We will discuss the concepts of MAUT and simple heuristics in the following sections.

3.1. A procedure for assessing Utility Function

Regardless of the technique being used to assess a utility function, the specific points or objectives that must be considered and accomplished by any assessment procedure are essentially the same as ensuing five sections [14].

3.1.1. Preparing for Assessment. Let X be the evaluator function, which associates to a consequence Q the real number $x = X(Q)$. Are higher x values more or less desirable? Ask whether we prefer a consequence x_1 to consequence x_2. We might ask him whether or not he prefers consequence T to consequence S in Fig. 1. Fig.1 shows a two-attribute, Y and Z, consequence space. By consequence Q we mean the consequence where $y = y_1$ and $z = z_1$. The consequence R is the consequence where $y = y_2$ and $z = z_2$.

3.1.2. Identifying relevant independence assumptions. We discuss procedures to verify whether Y and Z are additive independent and if either attribute is utility independent of the other.

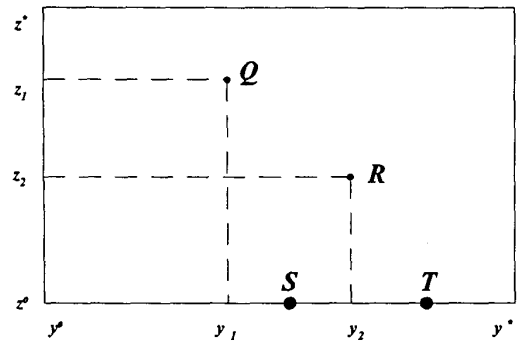

Figure 1. A two-attribute consequence space

3.1.3. Identifying relevant qualitative characteristics. We want to determine whether or not the utility function is monotonic. If x_k is greater than x_j, is x_k always preferred to x_j? Next, we want to determine whether u is risk averse, risk neutral, or risk prone. We ask the decision maker if he prefers $<x+h, x-h>$ or x for arbitrary chosen amounts of x and h. After the qualitative characteristics have been identified, we need to assess quantitative utility values for a few points.

3.1.4. Choosing a Utility Function. As a simple illustration of several points, suppose that the decision maker's utility function was monotonically increasing in x and decreasingly risk averse. We know a family of utility functions that satisfies these characteristics is

$u(x) = h + k(-e^{-ax} - be^{-cx})$, where a, b, c, and k are positive constants.

Now we verify relevant independence assumptions: that is check whether Y was utility independent of Z. It was concluded that Y was utility independent of Z. In a similar manner, Z was found to be utility independent of Y.

Generally, if the utility function $u(x_1, x_2, x_3)$ is additive and utility independent, then

$u(x_1, x_2, x_3) = k_1u_1(x_1) + k_2u_2(x_2) + k_3u_3(x_3)$, where $u_i(x^o_i) = 0$, $u_i(x^*_i) = 1$, for all i.

3.2. Utilization of simple heuristics. The reason why we use simple heuristics is that it is difficult to predict the preferences of users which are related to the attributes of a good. Also it is difficult for even users to determine their preferences quantitatively which is related to the attributes of a good.

There are two types of rationality: unbounded rationality and bounded rationality [9]. In decision models based on unbounded rationality, there are decision-making strategies that have little or no regard for the constraints of time, knowledge, and computational capacities that real human face. This is traditionally modeled by probability theory and the best-known realizations are the maximization of expected utility and Bayesian model.

In decision models based on bounded rationality, there are satisficing, and fast and frugal heuristics. Satisficing searches through a sequence of available alternatives. Fast and frugal heuristics use little information and computation to make a variety of kinds of decisions. There is a sound reason why a person might base a decision on only one reason rather than on a combination of reasons: combining information from different cues requires converting them into a common currency, a conversion that may be expensive. Assume that there is a common currency for all beliefs and desires, namely, quantitative probabilities and utilities. Although this is a mathematically convenient assumption, the way we look at the world does not always conform to it. And this heuristics employ a minimum of time, knowledge, and computation to make adaptive choices. Little time and knowledge would be to employ a form of *one-reason decision making* in which he need only find a single piece of information to determine his decision. This might base a decision on *only one reason* rather than on a combination of reasons [16].

Different environments can have different specific fast and frugal heuristics. But specificity can also be a danger if a different heuristic were required for every slightly different environment, we would need an unworkable multitude of heuristics. Fast and frugal heuristics avoid this trap by their simplicity and enable them to generalize well to new situations [9]. Take The Best tries cues in order, searching for a cue that discriminates between the two objects. If the discriminating is found, it serves as the basis for an inference, and all other cues are ignored. Since it does not integrate information or require extensive computations, it is fast. Since it has a stopping rule to effect limited search for cues, it is frugal. Take The Best outperforms multiple regression, especially when the training set is small [10].

3.3. Algorithm

Algorithm ***Negotiate()***

/**

Define *GoodDB* as the database which contains goods and users' preferences. If good G_i is selected in step 1, good G_j is selected in step 2, and G_k was purchased by the buyer, let's describe this transaction as T_{ijk}. Define $n(T_{ijk})$ as the number of the transactions successfully purchased so far. Define *CaseBase* as the history database which contains $n(T_{ijk})$. Define *numS* as the times of trials to use special knowledge. Define *numG* as the times of trials to use general knowledge. The total number of goods in a product, N = *numS* + *numG*, if every selected good is unique. If *numS* is equal to the total number of goods in a product, N, this algorithm uses only special knowledge. Else if *numS* is 0, this algorithm uses only general

knowledge. Otherwise, this algorithm uses both special knowledge and general knowledge.
**********/

Step 1 // *Using special knowledge based on MAUT.*

1.1 : *Preferences()* /* Determine the utility function by interaction with the user according to MAUT:
$u(x_1, x_2, x_3) = k_1u_1(x_1) + k_2u_2(x_2) + \ldots + k_nu_n(x_n)$,
where $u_i(x^o_i) = 0$, $u_i(x^*_i) = 1$, and k_i is constant for all i.
$u_i(x_i)$
- risk prone : $b(2^{cx} - 1)$
- risk neutral : bx
- risk averse : $b\log_2(x+1)$
where b, $c > 0$ constants */

1.2 : Define the set of all goods as $V(G) = \{G_1, G_2, \ldots, G_n\}$. Also define $V(S)$ as the subset of $V(G)$ which are selected according to MAUT in step 1. Define the number of elements in $V(S)$ as *numS*, where $V(S)$ is sorted from the most promising good to the least promising good according to MAUT.

1.3 : For every element of $V(S)$, negotiate and remove the element from *GoodDB*. If succeeded, then *Success()*. If N = *numS*, then *Fail()*.

/* If negotiations failed even after *numS* trials, change the negotiation policy from special knowledge-based negotiation to the general knowledge-based negotiation. */

Step 2 // *Using general knowledge based on the purchase history database:CaseBase and simple heuristics.*

2.1 : Define the set of the remaining goods which are not selected in step 1 as $V(R)$, where
$V(G) = V(S) + V(R)$ if every selected good is unique.
For every element G_k in $V(R)$ where $n(T_{ijk}) > 0$,
2.1.1 Retrieve G_k whose value of $n(T_{ijk})$ is the highest in *CaseBase*
2.1.2 Negotiate and remove G_k from *GoodDB*.
2.1.3 If succeeded then *Success()*

2.2 : *CuesOrder()* // The order of importance in cues is // determined by interacting with the buyer.

2.3 : // There is not such a G_k in step 2.1
For every element G_k in $V(R)$ // all remaining goods
2.3.1 *Heuristics*(G_k). If there is not such a G_k, then *Fail()*.
2.3.2 Negotiate and remove G_k from *GoodDB*.
2.3.3 If succeeded, then *Success()*.

// End of Negotiate()

Algorithm Heuristics(G_k) // *Take the best, ignore the rest.*

Step 1 : *TakeTheBest*(G_k)
/* If the most important preference is, say the cue ω, then good G_k in *GoodDB* whose value of the cue ω is the most desirable, is selected according to the result of Take The Best. */

Step 2 : If there is not such a G_k in *GoodDB* then return.

// End of Heuristics()

Algorithm Success()

Step 1 : Add $n(T_{ijk})$ in *CaseBase* by 1, and terminate the negotiation with success

Step 2 : exit(0)

// End of Success()

Algorithm Fail()

Step 1 : Terminate the negotiation in fail.

Step 2 : exit(1)

// End of Fail()

This algorithm is flexible. Because if we want to use only special knowledge that is MAUT, we can set *numS* = N. If we want to use both special knowledge and general knowledge, we can set $0 <$ *numS* $< N$. If we want to use only general knowledge, we can set *numS* = 0.

3.4. A negotiation scenario

Let's assume the initial *GoodDB* for a product P_1 is shown in Table 1. Here we assume three attributes and x_4 which is the seller's reserved margin.

Table 1. *GoodDB* for a product P_1

	G_1	G_2	G_3	G_4	G_5	G_6	G_7
x_1(price)	$50	$70	$40	$50	$30	$55	$45
x_2(delivery)	7	1	10	5	14	7	7
x_3(warranty)	3	6	2	5	1	3	2
x_4(margin)	.2	.3	.1	.2	0	.2	.1

Algorithm Negotiate()

Step 1: Using special knowledge based on MAUT.

1.1 : *Preferences()* // Utility function is determined by MAUT and suppose "risk averse" is selected by the user.

1.2 : suppose *numS* = 3.
Suppose good G_4, G_3, and G_1 are selected in sequence according to the utility function which was established in the previous step. Therefore V(S) = $\{G_4, G_3, G_1\}$

1.3 : Negotiation between buyer agent and seller agent is starting from G_4 to G_1 in V(S). Suppose all negotiations failed.

Step 2:Using general knowledge:CaseBase and heuristics

2.1 : Let's assume the content of CaseBase is shown in Table 2. Because $n(T_{ijk})$ of G_2, G_3, and G_4 are not zero as shown CaseBase, G_2, G_3, and G_4 could be the members of $V(R)$. But G_3 and G_4 failed in step 1. Therefore $V(R) = \{G_2\}$, and the agents can negotiate with G_2.
Also suppose this negotiation failed.

2.2 : *CuesOrder()* //Suppose order of cues is (x_2, x_1, x_3).

2.3 : 2.3.1 : *Heuristics()*
2.3.2 : G_7, G_6, G_5 are selected in sequence.
Because the order of cues is (x_2, x_1, x_3), the goods are selected according to this order. The value x_2 of G_7 and G_6 is the same, but G_7 is preferable in the second cue, x_1. Therefore G_7 is selected first.
2.3.3 : If the negotiation with G_7 failed and the

buyer purchase G_6 then the negotiation would be succeeded. The contents of CaseBase will be changed as shown in Table 3. Table 3 shows the adding T_{426} into Table 2. T_{426} means good G_4 is selected in step 1, good G_2 is selected in step 2, and G_6 was purchased by the buyer.

Table 2. *CaseBase* before purchasing G_6

T_{ijk}	$n(T_{ijk})$
T_{122}	1
T_{123}	2
T_{534}	2

Table 3. *CaseBase* after purchasing G_6

T_{ijk}	$n(T_{ijk})$
T_{122}	1
T_{123}	2
T_{426}	1
T_{534}	2

4. Design and implementation of Pmart

4.1. Pmart in distributed environments

Pmart is an application framework which provides electronic commerce services in the distributed environment as shown in Fig.2. This framework consists of Application layer, Framework layer, and Middleware layer. Application layer consists of variant components such as Pmart External and domain knowledge. Framework layer consists of invariant components such as Client, Server, and DB Server. When the domain is changed, Application layer could be changed whereas Framework layer needs not to be changed.

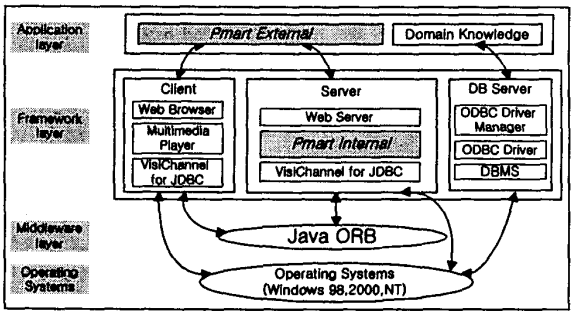

Figure 2. Distributed environment of *Pmart*

4.1.1. Pmart External and Domain Knowledge. Pmart External consists of agents which are sensitive to the changes of the domain, and Domain Knowledge is the knowledge pertaining to the specific domain. We could construct different types of electronic commerce systems by simply changing the contents of Pmart External.

4.1.2. Client. Client consists of Web Browser, Multimedia Player, and VisiChannel for JDBC Client [24]. Users may execute Interface Agent which is an applet on the Java-enabled web browser. Clients create objects, communicate with the ORB to convey a request for an operation invocation, and access to databases through the API of VisiBroker for Java [23,25]. Java has transformed the World Wide Web into an interactive system supporting objects but it is not a sufficient solution to the problem of creating transparently interoperating objects for client/server systems [6,15]. A platform for universal network computing can be created using Java as a programming language and mobile code system CORBA as an integration technology. The role of the stub class of CORBA is to provide proxy objects that clients can invoke methods on. The proxy object marshals and transmits the invocation request. The resulting marshaled form is sent to the object implementation using ORB's infrastructure.

4.1.3. Server. Server consists of a Web Server such as Internet Information Server 3.0, Pmart Internal, and VisiChannel for JDBC Server [24]. VisiChannel for JDBC is a multi-tiered architecture that enables a JDBC program on a client machine to access data in ODBC data sources on a server machine. Server responses to the requests of the learner. The skeleton code provides the glue between an object implementation, a CORBA server, and the ORB. The skeleton class implements the mechanism by which invocation request coming into a server can be unmarshalled and directed to the right method of the right implementation object. If the request is related to the database access, the API of VisiChannel for JDBC can be used.

4.1.4. DB Server. DB Server consists of ODBC Driver Manager, ODBC Driver, and DBMS. The ODBC Driver Manager implements the ODBC API and provides information to an application, loads ODBC drivers dynamically as they are needed, and offers argument and state transition checking. The VisiChannel for JDBC Client enables JDBC applications and applets to use the features of the ODBC drivers installed on the server machine. The server-side ODBC drivers handle direct interactions with the DBMS.

4.1.5. ORB and Operating Systems. ORB is Java ORB. CORBA can replace the current Web-based system of providing client/server services which uses HTTP/CGI [18,23,25]. Using CORBA with Java, that is Java ORB, would allow Java components to be split into client and server components making the Web client/server model even more attractive since download time would decrease. OS(Operating System) layer may be

Windows-based OS such as Windows98, 2000, and NT.

4.2. Architecture of Pmart

The overall architecture of the proposed framework Pmart is shown in Fig. 3. The electronic commerce framework Pmart consists of Pmart Internal which is independent on the specific domain and Pmart External which is dependent on the domain.

4.2.1. Pmart Internal. Pmart Internal consists of one Facilitator Agent, Buyer Agent, Seller Agent, and Information Agent. Facilitator Agent can help other agents in Pmart to communicate with each other using the functionality of Agent Name Server and the meta knowledge. Facilitator Agent consists of a global blackboard, a hierarchical knowledge, and an inference engine. Global blackboard contains the meta knowledge and the hierarchical knowledge is the knowledge about the construction of Facilitator Agent. The hierarchical knowledge is used for the construction of Facilitator Agents when the function of Facilitator is complex and complicated.

An agent can find desired agents without requiring the sender to know the locations of those agents using the function of Facilitator Agent. Each agent has to register the name, the function, and the address on the Facilitator Agent when it is created. When an agent needs some information on problem solving, Facilitator Agent informs the agent of the appropriate information using the meta knowledge stored in the Facilitator Agent.

Task agents formulate the problem-solving plans, exchange information with other agents, and proceed the autonomous problem solving procedures. Buyer Agent and Seller Agent are included in Task Agents as shown in Fig. 3.

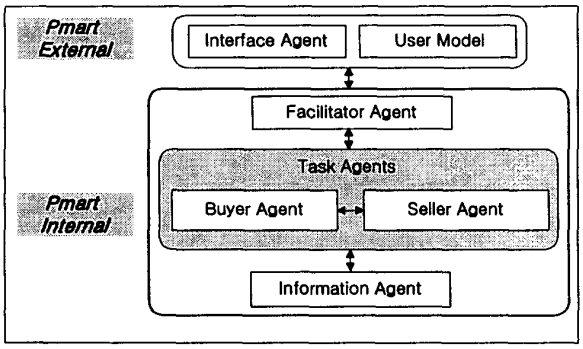

Figure 3. Architecture of Pmart

Seller Agent controls the electronic commerce service and is also a downloadable applet when the selling session begins. It has the selling and negotiation knowledge, inference engine, and the function of analyzing the seller's actions.

Buyer Agent is also downloadable applet when the buying session begins. It has the buying and negotiation knowledge, inference engine, and the function of analyzing the buyer's actions. Both Seller Agent and Buyer Agent communicate with Facilitator Agent to update the User Model in the end of negotiation.

Information Agents provide intelligent access to a heterogeneous collection of information sources. And these agents have models of the associated information resources and strategies for source selection, information access, and conflict resolution.

The communication between Buyer Agent, Seller Agent, and the Facilitator Agent uses Java Event and EventListener interface technique. The argument object that is passed with the Events is new object called a Message which is modeled after a standard KQML message packet. We have modified the concept of the Marketplace [2] in the agent communication.

4.2.2. Pmart External. Pmart External consists of Interface Agent and the User Model which is the variant part of Pmart when the domain is changed.

Interface Agent interacts with the users to acquire information about users' profile, responses, and goals. It observes and imitates the user's behavior, adapts based on user feedback, and can ask for advice from other agents assisting other users. This agent is a first downloadable Java applet when the session begins and exchanges information with Information Agent to fetch/store the contents of the User Model from/into databases.

User Model has knowledge of the information about buyers and sellers, and the purchase history of the negotiation. User Model is placed in the server-side and the client-side at the same time for the efficiency of the communication. Since the frequency of the communication among Interface Agent, Buyer Agent, and User Model is very high, we can place these components in the client-side together during the buying/selling session for the efficiency. In the end of session, however, the contents of User Model is copied into that of the server-side for consistency.

Interface Agent may be created by applets and other agents may be created by applications.

4.3. Implementation of Pmart

The implementation environment is as follows:

Server : Windows NT 4.0 Server
Client : Windows 98
Web Server : Internet Information Server3.0
Data Base Server : NT Oracle 7.3
Web Browser: Communicator 4.7 / Expleror 5.0
Development Software
- JDK1.2.2 - Symantec Visual Cafe
- VisiBroker for Java - VisiChannel for JDBC

In the proposed electronic commerce framework Pmart, *n*-web clients, including buyers and sellers, make

request for the services of multiple web servers and DB servers through Java ORB, VisiBroker for Java 3.2. In this approach, instead of invoking a method on a remote object, the code for the class is transferred across the network, run locally and then the method is invoked on a local object instance. Here we choose Inprise's VisiBroker for Java 3.2 as Java ORB.

VisiChannel for JDBC requires that the following software be installed on our client machine before installing the VisiChannel for JDBC software: Java development environment (such as the JDK) and a Java-enabled web browser. In addition, the VisiChannel for JDBC Server machine must have the following software: Java runtime environment (such as the JDK), ODBC Driver Manager and ODBC drivers, and appropriate client and network libraries for our database.

The ODBC Driver Manager implements the ODBC API and provides information to an application, loads ODBC drivers dynamically as they are needed, and offers argument and state transition checking.

We have implemented a prototype of Pmart. The distributed environment of Pmart in Fig. 2 was developed by AI Lab of Pukyong National University [5]. We have developed Pmart architecture in Fig. 3 and modified the concept of Marketplace [2] in the agent communication modeled after a standard KQML. Among Pmart External, the implementation of Interface Agent is partially implemented and is going on using Visual Cafe. Fig. 4 shows a typical operation model of Pmart.

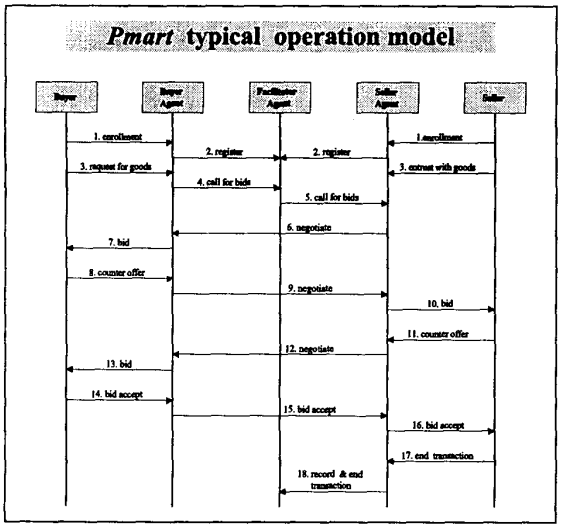

Figure 4. *Pmart* typical operation model

In Fig. 4, both agents Buyer Agent and Seller Agent must register with the Facilitator Agent before they can have any interactions with other agents in the electronic commerce. When the users make their own agents and select the Start option, a single Facilitator Agent is created. Seller Agent advertises the items they have to sell by sending messages to the Facilitator Agent. Buyer Agents initialize their own buying lists. The first sales negotiation takes place when one of the Buyer Agents wakes up and takes an item from its buying list. It then asks the Facilitator Agent to recommend a Seller Agent who has advertised its ability to sell that item. If more than one seller has advertised an item in the global blackboard, the Facilitator Agent provides all information of sellers to the Buyer Agent. And then, Buyer Agent begins the negotiation with Seller Agent based on our negotiation algorithm, *Negotiate()*. Buyer Agent or Seller Agent could make a counter offer if they are not satisfied by the current negotiation.

4.4. Characteristics of Pmart and other systems

Electronic commerce framework Pmart has the characteristics of an intelligent agent since it is based on the multiagent architecture such as autonomy, social ability and pro-activeness. Pmart is autonomous: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.

The second characteristic of Pmart is the social ability: agents interact with other agents via some kind of agent-communication language or the methods of other agents. Here we used KQML-like Java methods.

The third characteristic of Pmart is pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative. Pmart also pro-actively seeks out potential buyers or sellers and negotiates with them on their owner's behalf using the contents of User Model.

If we compare Pmart and other systems which are based on MAUT and sophisticated negotiation strategies, we can consider Tete-a-Tete developed in MIT. In this system, shopping agents assist shoppers during negotiations by providing them with a level of decision support to help them determine which merchant offering best meets their needs. This decision support is based on the MAUT. In this sense it is similar to Pmart. But this has some problems basically related to inferring the probabilities in the decision-making problem under uncertainty. Whereas in Pmart, we can use special knowledge: MAUT in the first negotiation phase, and if that fails even after several trials we could apply general knowledge heuristically in the second negotiation phase. In such a model, we have proposed a kind of *case-based reasoning* to observe behavior in previous similar situations. The generalization knowledge of transaction behaviors is achieved using successful purchase history so far and simple heuristics. Since the successful history database, CaseBase, is pertaining to all customers' buying/selling behaviors, it is meaningful data. Therefore, we could expect more successful negotiation result due to special knowledge as well as general knowledge. Other systems such as Kasbah, AuctionBot, Magnet, SICS

MarketSpace do not have sophisticated negotiation strategies like Pmart.

5. Conclusion and the future work

Today's first generation shopping agent is limited to comparing merchant offerings usually on price instead of their full range of value. Even in the full range comparison, there is not a good model which considers the overall features in the negotiation process appropriately. Therefore the negotiation model needs to be extended to include negotiations over the more attributes.

In this paper, we have suggested an agent-mediated electronic commerce framework called *Pmart* based on the multiagent to negotiate over prices, product features, warranties, and service policies. And we have proposed a negotiation model which is based on special knowledge and general knowledge at the same time. The former is based on the MAUT and the latter is based on the previous successful purchase history and the simple heuristics.

Pmart framework provides the software reuse and the extensibility based on the object-oriented and software components technologies. Therefore, we could make different types of electronic commerce systems by simply changing the contents of Pmart External that is variant part of Pmart.

Future implementation may be based on mobile agent platforms such as Voyager or SMART and utilize a more sophisticated collection of information agent. In the future, we will study the general auction framework, which supports our negotiation algorithm.

References

[1] Amazon.com. http://www.amazon.com/exec/obidos/subst/home/ home.html/

[2] J.P.Bigus and J.Bigus, *Constructing intelligent Agents with Java*, John Wiley & Sons, New York, NY, 1998.

[3] A.Chavez and P. Maes, "Kasbah: An Agent Marketplace for Buying and Selling Goods," *Proceedings of the first International conference on the Practical Application of Intelligent Agents and Multi-agent Technology (PAAM96)*, London, UK, 1996, pp. 75-90.

[4] A. Chavez et al., "A Real-Life Experiment in Creating an Agent Marketplace," In *Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-agent Technology (PAAM97)*, London, UK, 1997, pp. 159-178.

[5] M. Chung et al., Multiagent-based Distance Learning Framework using CORBA, In *Proceedings of the IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA99), Nassau, Bahamas, 1999, pp. 224-228.*

[6] G.Cornell and C.S.Horstmann, *Core Java, 2nd Edition,* CA; Sun Soft Press, CA, 1997.

[7] J. Eriksson et al., "SICS MarketSpace - An Agent-Based Market Infrastructure," *Lecture Note in Artificial Intelligence 1571, Agent Mediated Electronic Commerce,* 1998, pp. 41-53.

[8] G.Gigerenzer et al., *Simple Hueristics That Make Us Smart,* Oxford University Press, New York, 1999.

[9] G.Gigerenzer and P.M.Todd, "Fast and Frugal Heuristics: The Adaptive Toolbox" In *Simple Hueristics That Make Us Smart,* Oxford University Press, New York, 1999. pp.3-34.

[10] G.Gigerenzer and D.G.Goldstein, "Betting on One Good Reason The Take The Best Heuristics," In *Simple Hueristics That Make Us Smart,* Oxford University Press, New York, 1999. pp.75-95.

[11] R.H. Guttman and P. Maes, "Agent-Mediated Integrative Negotiation for Retail Electronic Commerce," *Lecture Note in Artificial Intelligence 1571, Agent Mediated Electronic Commerce,* 1998, pp. 70-90.

[12] R.H.Guttman, Merchant Differentation through Integrative Negotiation in Agent-mediated Electronic Commerce, MS thesis, Media Art and Sciences, MIT, 1998.

[13] R.H.Guttman et al., "Agent-mediated electronic commerce: a survey," *Knowledge Engineering Review*, Vol. 13, No. 2, 1998, pp. 147-159.

[14] R.L.Keeney and H.Raiffa, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, John Wiley & Sons, New York, NY, 1976.

[15] S.C.Lewandowski, Frameworks for Component-Based Client/Server Computing, *ACM Computing Survey,* Vol. 30, No. 1, 1998, pp.3-27.

[16] L.Martignon and U. Hoffrage, "Why Does One-Reason Decision Making Work?" In *Simple Hueristics That Make Us Smart,* Oxford University Press, New York, 1999. pp.119-140.

[17] OnSale:Live Online Auction House. http://www.onsale.com/

[18] R.Orfali and D.Harkey, *Client/Server programming with JAVA and CORBA 2nd ed.*, John Wiley&Sons, Inc., NY, 1998.

[19] D.C.Parkes et al., "Accounting for Cognitive Costs in On-Line Auction Design," *Lecture Note in Artificial Intelligence 1571, Agent Mediated Electronic Commerce,* 1998, pp. 25-40.

[20] S.Russell and P.Norvig, *Artifiial Intelligence : A Modern Approach*, Prentice Hall Inc., NJ, 1995.

[21] E. Steinmetz et al., "Bid evaluation and Selection in the MAGNET Automated Contracting System," *Lecture Note in Artificial Intelligence 1571, Agent Mediated Electronic Commerce,* 1998, pp. 105-125.

[22] United Computer Exchange. http://www.use.com/

[23] VisiBroker for Java 3.2 Programmer's Guide, Visigenic Co.,ftp://ftp.visigenic.com/*private /vbj/ vbj32/vbj32.html*

[24] VisiChannel for JDBC Administrator and Programming Guide Release 1.1, http://www.inprise.com/*techpubs /visibriker/vc4jdbc/html/jdbc2.htm*

[25] A. Vogel and K. Dubby, *JAVA Programming with CORBA, 2nd Edition*, Wiley Computer Publishing Co., NY, 1998.

[26] P.R.Wurman et al., "The Michigan AuctionBot: A Configurable Auctioon Server for Human and Software Agents," In *Proceedings of the Second International Conference on Autonomous Agents (Agents98)*, Minneapolis, MN, 1998, pp.301-308.