

# Handy Broker - An Intelligent Product-Brokering Agent for M-Commerce Applications with User Preference Tracking

Steven Guan, Chon Seng Ngoo and Fangming Zhu  
Department of Electrical & Computer Engineering  
National University of Singapore  
10 Kent Ridge Crescent, Singapore 119260

## Abstract

One of the potential applications for agent-based systems is m-commerce. A lot of research has been done on making such systems intelligent to personalize their services for users. In most systems, user-supplied keywords are generally used to help generate profiles for users. In this paper, an evolutionary ontology-based product-brokering agent has been designed for m-commerce applications. It uses an evaluation function to represent a user's preference instead of the usual keyword-based profile. By using genetic algorithms, the agent tracks the user's preferences for a particular product by tuning some parameters inside its evaluation function. A prototype called "Handy Broker" has been implemented in Java and the results obtained from our experiments looks promising for m-commerce use.

**Keywords:** product-brokering agent, m-commerce, profiling, genetic algorithms

# 1. INTRODUCTION

In this IT age, there is an increasing demand for more and more sophisticated software, which are capable of integrating and processing information from diverse sources. Traditional software technologies have failed to keep pace with such a demand and alternative solutions are being considered. Agent-based systems [1,2] have been proposed as a potential solution and much research has been done on this relatively new technology.

One of the potential applications for agent technology is in m-commerce. According to a study done by Frost and Sullivan<sup>1</sup>, it has been projected that electronic commerce conducted via mobile devices such as cellular phones and PDAs (Personal Digital Assistants) will become a whopping \$25 billion market worldwide by 2006. Some of the driving factors behind the m-commerce “revolution” have been attributed to the compactness and high penetration rate of these mobile devices. This along with the relatively low cost of entry for most service providers has made m-commerce the buzzword of the next century.

However, despite all the hype and promises about m-commerce, several main issues [3,4] will have to be resolved before agent technology can be fully adopted into any m-commerce system. Clumsy user interfaces, cumbersome applications, low speeds, flaky connections and expensive services have soured many who have tried m-

---

<sup>1</sup> <http://www.infoworld.com/articles/hn/xml/02/03/22/020322hnmcommerce.xml>

commerce. In fact, a usability study done in London by the Nielsen Norman Group<sup>2</sup> has found that about 70% of the participants said that they will not use a WAP (Wireless Application Protocol) enabled phone again within a year, after they have tried it for a week. Security and privacy concerns have also dampened enthusiasms for m-commerce. One of the concerns has been the fact that mobile devices such as PDAs are very easy to lose. They are also an easy prey for thieves and unauthorized personnel can have easy access to valid user IDs and passwords stored in these devices to make fraudulent transactions.

Taking all these concerns into account, it seems like good old e-commerce will remain as the preferred choice for online transactions for many years to come. Customers will only use wireless mobile devices to access the Internet if they have a good reason to do so. Therefore, in order to entice customers to participate in m-commerce, the developers will have to offer something that is unique and no consumers can live without. One of the potential “killer” applications for m-commerce could be an intelligent program that is able to search and retrieve a personalized set of products from the Internet for its user.

Currently, when a user wants to search for a particular product on the Internet, what he will normally do is to use popular search engines such as Altavista<sup>3</sup> or Yahoo!<sup>4</sup>, and enter keywords that describe the product. These search engines will process these

---

<sup>2</sup> <http://www.nngroup.com/reports/wap/>

<sup>3</sup> <http://www.altavista.com>

<sup>4</sup> <http://www.yahoo.com>

keywords and churn out a large number of links for the user to visit. On the other hand, if the user already knows of some URLs that might have the product information, he will visit these websites and hopefully get the information that he is looking for.

Although these are the more common methods of searching for information on the Internet, it may not be the best nor the most efficient ones. Neither search engines nor websites know the preference of the user and hence might provide information that are totally irrelevant to the user. For example, if the user wants to search for information about “software agents”, the search engine could return links to “insurance agents” instead. A significant amount of time could be wasted on such irrelevant information, which could have been better spent on other important tasks.

In an agent-based m-commerce, agents act on behalf of their users by carrying out delegated tasks automatically. Currently, there is no single agent that can perform all the tasks meted out by the user effectively. Like humans, specialized agents are required, which are able to work in a specific type of environment. A product-brokering agent seems to be a potential solution for this scenario. Such an agent will search for products in the background with minimal user intervention, thereby allowing the user to concentrate on other tasks. It could be programmed with the user’s preferences in mind and filter out irrelevant products automatically. The agent could also detect shifts in user’s interest and through some evolution mechanism, adjust itself accordingly to suit the user’s need.

This paper describes the design of “Handy Broker” - an intelligent agent-based system, which is capable of providing a personalized service for its user. It accomplishes this through *user profiling* [5]. The system consists of several intelligent ontology-based product-brokering agents, which are able to learn the preferences of the user over time and recommend products, which might interest the user. *Handy Broker* achieves this either by user’s feedback or through its own observation. This technique has been used successfully for specific types of agent tasks, typically those information-intensive tasks involving the World Wide Web (WWW).

Section 2 of this paper will highlight some of the related work that has been done by other researchers and a proposed design for *Handy Broker* will be presented in Section 3. A prototype has been implemented using Java and the system has passed through a series of tests. Results obtained from these tests will be discussed in Section 4. Although the results are encouraging, some limitations of the system will also be highlighted in this section. Potential applications for *Handy Broker* in m-commerce will be discussed in Section 5. Finally, the last section presents some concluding remarks along with some discussion on the possible extensions for future work.

## **2. RELATED WORK**

Personalized product-brokering agents require a profile of the user in order to function effectively. The agent would also have to be responsive to changes in the user’s interests and be able to search and extract relevant information from outside sources.

The rest of this section will highlight some of the work done by other researchers, which is closely related to our *Handy Broker*.

At MIT Media Labs, B. Sheth and P. Maes [6,7] have come up with a system that is able to filter and retrieve a personalized set of USENET articles for a particular user. This is done by creating and evolving a population of information filtering agents using genetic algorithms [8].

Some keywords will be provided by the user and they represent the user's interests. Weights are also assigned to each keyword and the agents will use them to search and retrieve articles from the relevant newsgroups. After reading the articles, the user can either give a positive or negative feedback to the agents via a simple GUI. Positive feedback increases the fitness of the appropriate agent(s) and also the weights of the relevant keywords (vice versa for negative feedback). In the background, the system periodically creates new generations of agents from the fitter species while eliminating the weaker ones. Initial results obtained from their experiments have been encouraging and showed that the agents are capable of tracking its user's interests and recommend mostly relevant articles.

While the research work at MIT requires the user to input his preferences into the system before a profile can be created, B. Crabtree and S. Soltysiak from BT Laboratories [9,10] believed that the user's profile can be generated automatically by

monitoring the user's web and email habits, thereby reducing the need for user-supplied keywords.

Their approach is to extract high information-bearing words, which occurs frequently in the documents that are opened by the user. This is achieved by using ProSum<sup>5</sup>, which is a text summarizer that can generate a set of keywords to describe the document and also determines the information value of each keyword. A clustering algorithm is then employed to help identify the user's interests and some heuristics are used to ensure that the program could perform as much for the classification of interest clusters as possible, thereby minimizing the amount of user input required in the profile generating process.

However, they have not been completely successful in their experiments. The researchers admitted that it would be very difficult for the system to classify all the user's interests without the user's help. Nevertheless, they believed that their program has taken a step in the right direction by learning a user's interest with minimal human intervention.

A new product-brokering agent usually does not have sufficient information to recommend any products to the user. Hence, it has to get product information from somewhere else. A good source of information will be the Internet. In order to do that, a method suggested by G. Pant and F. Menczer [11] is to implement a population of

---

<sup>5</sup> Profile-based text summarization

web crawlers called *InfoSpiders* that searches the WWW on behalf of the user. It will gather information on the Internet based on the user's query and indexes them accordingly. It behaves much like a personalized search engine but is designed to evolve and retrieve only relevant web pages for its user.

These agents initially rely on traditional search engines to obtain a starting set of URLs, which are relevant to the user's query. The agents will then visit these websites and decode their contents before deciding where to go next. The decoding process includes parsing each web page visited and by looking at a small set of words around each hyperlink, a score is given based on their relevance to the user. The link with the highest score is then selected and the agent will then visit the linked website.

No further details have been provided on how they extract or analyze the contents of the web pages but it has been mentioned that they use neural networks, HTML and XML parsing tools that are commonly used by other web crawlers. The agent stops after they have visited a pre-determined number of web pages or when it could no longer find any relevant web pages. The user can also terminate the search any time he wishes.

### **3. DESIGN OF HANDY BROKER**

*Handy Broker* can be used to search for all kinds of products. In our application, the system will be used to search for some computer products, namely *CPU*, *Mainboard*



and *Memory*. It is possible to extend the application to search for other products. All the codes are written in Java as it is object-oriented in nature and is compatible across multiple operating systems.

Similar to the information filtering agents done by B. Sheth and P. Maes, an initial population of product-brokering agents will be created and evolved using some form of genetic algorithms. However, in this design, the *profile* of the user is not based on any keywords supplied by the user. In fact, no keywords are required to be entered by the user. Instead, each agent will have an evaluation function that will be used to calculate the *value* of each product. Products that have a higher *value* will have a higher chance of being recommended by the agent. This evaluation function has some tunable parameters, which characterizes the user's preferences for a particular category of products. Initially, these tunable parameters will be randomly generated based on some heuristics but it will evolve over time to match with the user's preferences.

In our design, some assumptions have been made about the system. One of the most important assumptions is that the user of the system is a rational person and will select a product rationally. Another important assumption is that the value, which a user places on a product, can be calculated mathematically. The product values that we are focusing on will be those that can be calculated by using some tangible attributes (e.g. *price*) of the product. The agent will not be able to calculate the intangible value (e.g.

*brand*) that is imposed on the product. If these assumptions are not met, the agent will not be able to track the user's preferences successfully.

### 3.1 Ontology

Before *Handy Broker* is able to explore the Internet and retrieve product information for the user, agents need to have some prior knowledge such as the URLs of some relevant websites, keywords or some quantifiable attributes that can be used to describe the product. It could be tedious if the user has to enter such information into the agents when he wants to search for a particular product. Imagine the amount of data he will have to enter if he wants to search for several different products.

An alternative to this is to create *product ontology* as shown in Figure 1 that already contains some prior information in it (e.g. URLs of relevant websites). Creating *product ontology* involves defining the meaning of each term that is used to describe the product, their valid range of values and their relationship with one another. This is necessary as the same term might have different meaning on different products.

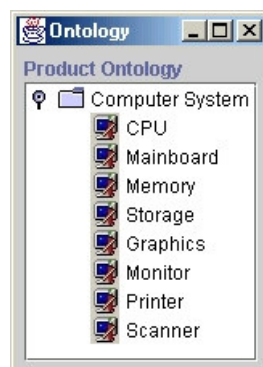


Figure 1: Screenshot of product ontology

*Product ontology* can be implemented in a tree-like structure, with the leaf nodes representing products and the parent nodes representing product categories. Each leaf node actually contains a Java class called *productInfo*, which has some prior information about the product. Different products will have different *productInfo* classes. A new product can be added as a leaf node to its parent node easily. When a leaf node is selected, relevant product information can be passed to the product-brokering agents. Currently, selection of a leaf node will pass the URL of the product's website and its attributes to the agents automatically.

## 3.2 Product-Brokering Agent

After describing how our agent obtains its product knowledge, the next stage is to define the agent itself. An agent will basically be a Java programming thread that will be running continuously in a while-loop until some terminating conditions have been met. A unique agent name will be given to each agent so that we can identify and differentiate one agent from another.

### 3.2.1 Agent's Fitness function

To calculate the fitness of an agent, a fitness function has been defined by using the following equation:

$$Fitness = \frac{\sum_{n=window\_size}^{points\ earned\ in\ the\ recent\ n\ generations}}{n} \quad (1)$$

This fitness function is basically an un-weighted (or simple) moving average of the agent's fitness, where  $n$  is the window size of the moving average. Using equation (1),

the agent's fitness is obtained by averaging the number of points earned by the agent in the current and the previous  $n-1$  generations. By varying the value of  $n$ , we can effectively control the number of generations under consideration and the points earned outside this "window" will not be considered. The rationale for this is that more emphasis should be placed on the agent's current performance instead of its past performances. As the fitness of an agent would be used to determine which agent to evolve, we do not want its past performances, which might be irrelevant now, to influence the evolution process.

An agent's fitness will always be a positive value and a new agent would start off with some default fitness. The fitness of an agent can also be a good indicator about the agent's performance. Therefore, in order to keep track of an agent's performance, each agent will have a list called *fitness\_history* and this is used to store the fitness of an agent for each generation. Hence, after an agent has been awarded some points, it will calculate its new fitness using equation (1) and inserts the value into its *fitness\_history*. Details on how an agent earns its points will be discussed in the next few sections.

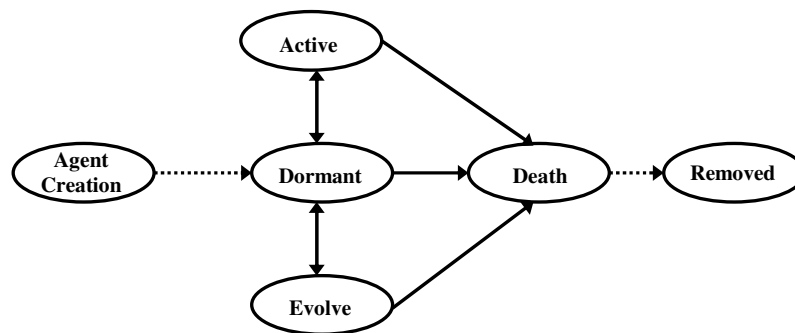
### **3.2.2 Agent's Life Cycle**

When an agent is created, it can be in any of the four different states (i.e. *Dormant*, *Active*, *Evolve* or *Death*). During agent creation, it will register itself to a database and its default state will be the *Dormant* state. This is to allow the user to keep track of all

the agents running in the system. This database also allows agents to store any product information that they have found on the Internet.

While an agent is still “alive”, it will toggle between the *Dormant*, *Evolve* and *Active* states, depending on certain circumstances. If the agent is killed by the user, it will switch to the *Death* stage and de-register itself before it is removed from the system.

Figure 2 shows how the four different states are related:



**Figure 2: Agent's life cycle**

#### *i. Dormant*

The agent is not doing any task at this moment. It is waiting for the user to give it instructions. Note that this will be the default state of the agent once it has been created. In this state, the user can modify the agent's parameters before starting the agent.

### *ii. Active*

The agent has received some instructions and is currently performing some tasks for the user. The types of tasks performed by the agents will be discussed further in Section 3.2.3.

### *iii. Evolve*

The agent has received some user feedback regarding its performance. It is now analyzing this feedback and making the appropriate adjustments. The evolution process will be discussed in greater detail in Section 3.5.

### *iv. Death*

The agent has been killed and it can no longer perform any task for its user. Note that it is still present in the system and will only be removed when instructed by the user. This allows the user to recycle any information that he might find suitable.

## **3.2.3 Agent's Task**

Once the user has passed some instructions to an agent, it will switch to the *Active* state and activates the appropriate tasks. As the agent might need to perform different types of tasks simultaneously, these tasks are implemented as independent and self-contained programs, which are separated from the agent. Therefore, instead of implementing several agents from scratch to perform different tasks, we only need to implement a basic agent and a few task programs. What the basic agent needs to do is

to call the appropriate task program, pass some information to it and the task programs will handle the rest.

For our application, the agent's task has been designed specially to parse information from a website called *Hardwarezone.com*<sup>6</sup>. It is a website hosted in Singapore and it displays up-to-date information of various computer products in a tabular form. The task program allows the agent to establish a connection to the website and download the HTML document onto a local computer. The program then parses the document and extracts relevant information for the agent by looking for specific tags within the HTML document. In our application, the program will be able to extract information such as the product description, price, performance and also the name of the shop that is selling this product.

### **3.2.4 Agent's Knowledge**

After the agent has obtained relevant product information from the website, it will need a place to store this piece of information. As mentioned in Section 3.2.2, when an agent is created, it will register itself to a database. A Microsoft Access database (Figure 3) is used in our application. Within the database, a table will be created for each agent to store the information (i.e. *product description*, *price*, *performance*, and *shop name*) that it has retrieved from the website. There is also a column called *prod\_value*, which contains the value that the agent has placed on that particular

---

<sup>6</sup> <http://www.hardwarezone.com>

product. Details on how the agent derives this value will be discussed in the next section.

Description	Performance	Price	Shop	prod_value
INTEL Pentium-	1.85	322	Costronic	2.78322378179
INTEL Pentium-	1.8	292	Costronic	2.76829132959
INTEL Pentium-	1.65	205	Costronic	2.71500137257
AMD Athlon XP	1.65	211	Costronic	2.69801617168
INTEL Pentium-	1.9	388	Costronic	2.69624502862
INTEL Pentium-	1.7	247	Costronic	2.69596342297
INTEL Pentium-	1.6	199	Costronic	2.63212811680
INTEL Pentium-	1.9	418	Cybermind	2.61131902415
AMD Athlon XP	1.47	115	Costronic	2.61028894199
AMD Athlon XP	1.45	110	Costronic	2.58449989341
INTEL Pentium-	1.9	452	Bliss	2.51506955243
AMD Athlon XP	1.7	312	Costronic	2.51195707996
INTEL Pentium-	1.7	317	Costronic	2.49780274589
AMD Athlon 1.4	1.4	115	AGENUINE	2.47048710267
INTEL Pentium-	1.6	270	Laser	2.4311365729
INTEL Pentium-	1.9	488	Sysnet	2.41315834707
INTEL Pentium-	2.05	595	Costronic	2.40983096778
AMD Athlon XP	1.335	92	Costronic	2.40578104577
INTEL Pentium-	1.8	438	MediaPro	2.35498477453
AMD Athlon XP	1.54	259	Laser	2.34244595988
AMD Athlon 1.3	1.33	111	io Data	2.34200873061
INTEL XEON 1.	1.9	520	IMS	2.32257060897
AMD Athlon XP	1.47	228	Bell	2.29040099184
AMD Athlon 1.2	1.25	89	Bliss	2.2445142699

Figure 3: Screenshot of an agent's database

In addition to this, each agent will also store its data in a *global* database. The *global* database is similar to the database as shown in Figure 3 but it will contain all the products that have been retrieved by the agents in the system.

### 3.2.5 Product Recommendation

Before recommending a product to the user, the agent should be able to evaluate which product would fit the user's requirements best. A proposed method is to use some quantifiable attributes such as *cost* and *performance*, to evaluate products. An example of an evaluation function could be based on the following equation:



$$product\_value = perf\_weight * performance - cost\_weight * cost \quad (2)$$

Equation (2) tries to model the two types of factors that can influence a user's choice. The first attribute (*performance*) represents the performance of a product while the second attribute (*cost*) represents the cost of a product. It has been assumed that the better the product, the higher will be its *performance* and a better product usually results in a higher *cost*. From equation (2), it can be seen that a product with a higher *performance* and/or a lower *cost* will result in a higher *product\_value*.

The two weights – *perf\_weight* and *cost\_weight*, represents the weights that the user could give to each attributes. These two parameters are actually used to represent the user's preferences and are incorporated inside the agent. If *perf\_weight* has a higher value, it means that the user place more emphasis on the performance of the product. Likewise, if the user has a higher value for *cost\_weight*, it means that the user is more concerned about the cost of the product. Note that, for different products, a different set of attributes and weights could be defined under equation (2) and all these could be defined inside the *product ontology*.

When an agent is created, these two weights will be initialized based on some heuristics and would be used to calculate the value of each product found in the agent's database. The agent will then rank the products according to their values and selects the top three products to be presented to the user. The value of *perf\_weight* and *cost\_weight* will be allowed to change when the agent undergoes evolution.

### 3.2.6 Agent's GUI

It will be useful if the user is able to observe what is happening inside an agent when required. To facilitate this, *Handy Broker* provides a simple GUI (Figure 4) that shows information such as the name of the agent, its current status, and products recommended. It would also allow the user to change some of the parameters inside an agent.

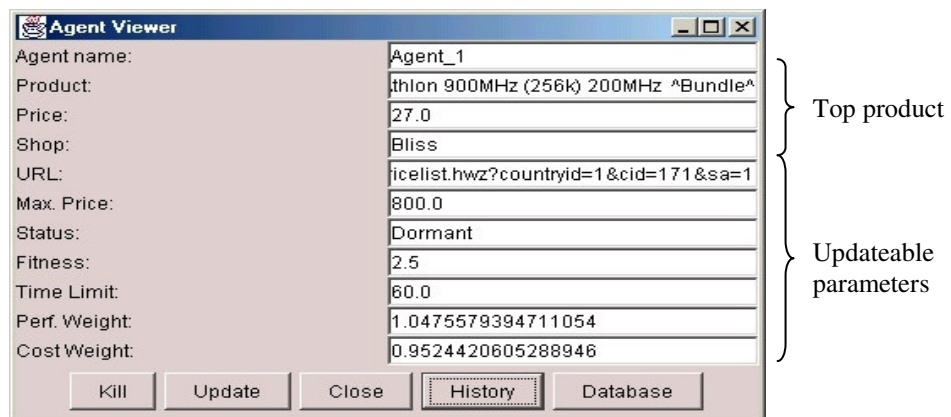


Figure 4: Agent's GUI

## 3.3 Monitoring Tools

*Handy Broker* also provides a monitoring tool for the user, which allows him to observe and control the behavior of his agents as a group. This tool will be the main interface between the user and his agents. From this interface, the user has full control on the lifecycle of his agents. The user can choose from a list of products provided in the *product ontology* and enter some parameters (e.g. number of agents) before starting a search.

Once all the parameters have been entered into the system, an appropriate number of agents will be created to search for the product on the Internet. While the program is

running, the user can start or stop the agents any time by using this tool. A text message will also be provided to allow the user to track the progress of his agents in the system. To provide feedback on the agent's performance, the user can do so by clicking on the *Result* button. A screen shot of the implemented system is shown in Figure 5.

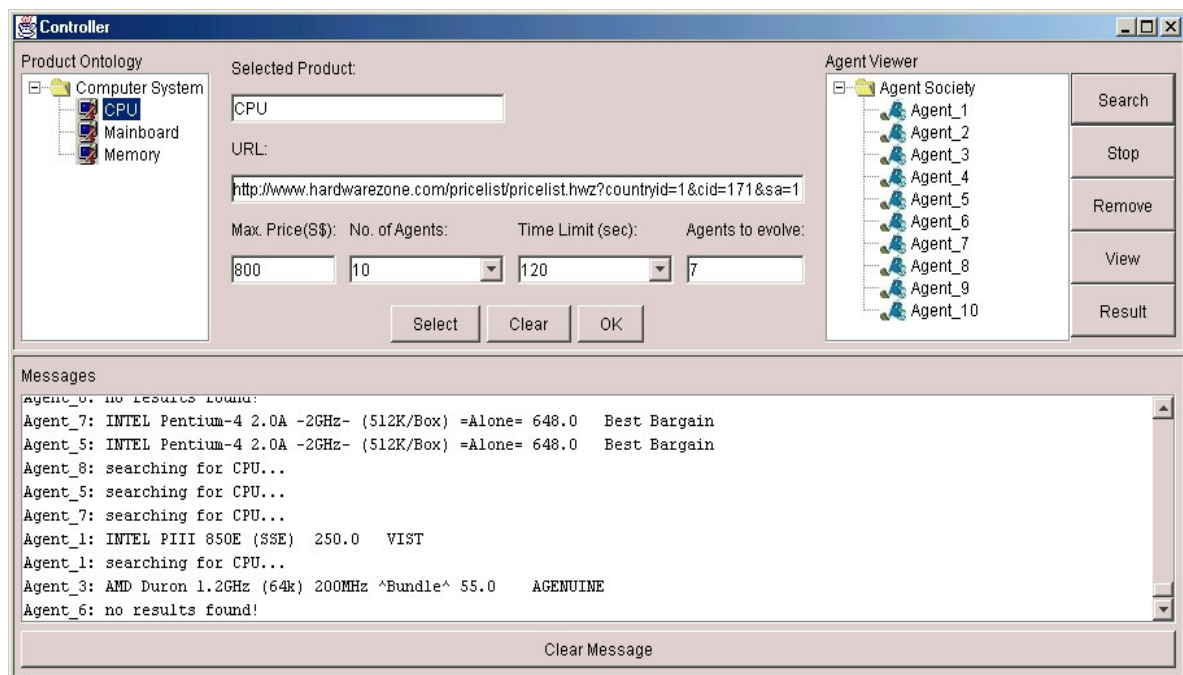


Figure 5: Monitoring tool in action

### 3.4 User Feedback

During user feedback, each agent in the system will select the top three products in its database and adds them into a *recommended* list. A sorting function will be implemented to allow the user to sort the list according to his preferences. If the user cannot find any product that he fancies in this list, he can look at the *global* list, which contains all the products that have been retrieved by the agents in the system. When

the user sees a product that he likes, he can select the product by clicking on it and all the agents in the system will be informed about the user's selection.

The agents will take note of the product that the user has selected and searches for that product inside its own database. At this stage, each agent would have already assigned a product value to each product in its database. To determine the number of points to be awarded to an agent, the products will be ranked in an ascending order according to this value. Hence, products with a higher value will be located at the bottom of the table. The agent will then determine the position of the user-selected product and take note of its row number. The formula to calculate the exact number of points to be given to an agent is as follows:

$$\text{points awarded} = \frac{\text{row number of user selected product}}{\text{total number of products}} \times \text{maximum points} \quad (3)$$

As an example, Figure 6 shows an agent's product list after it has been sorted in ascending order.



No.	Description	Price	Shop
1	AMD Athlon XP 1900+ (1.6GHz) 266MHz =Bundle=	288.0	Bliss
2	INTEL Pentium-4 1.8GHz(256K/Box) =Bundle=	289.0	Costronic
3	INTEL Pentium-4 1.7GHz(256K/Box) =Alone=	325.0	Sysnet
4	AMD Athlon MP 1500+ (1.33GHz) 266MHz *Alone*	330.0	MediaPro
5	INTEL PIII 1.13GHz (Tulatin) -Box- *Alone*	340.0	MediaPro
6	INTEL PIII 1.26GHz (Tulatin) -Box- *Alone*	399.0	MediaPro
7	INTEL Pentium-4 1.8GHz(Box) =Bundle=	399.0	Video-Pr...
8	INTEL Pentium-4 1.9GHz(Box) =Alone=	418.0	Cybermi...
9	INTEL Pentium-4 1.8GHz(Box) =Alone=	438.0	MediaPro
10	INTEL PIII 1.26GHz (Tulatin) -Box- ^Bundle^	445.0	Taknet
11	INTEL Pentium-4 1.9GHz(Box) =Bundle=	452.0	Bliss
12	AMD Athlon MP 1900+ (1.6GHz) 266MHz =Bundle=	490.0	Superpet
13	AMD Athlon MP 1800+ (1.53GHz) 266MHz *Alone*	495.0	IMS

Figure 6: Screenshot of an agent's database after sorting

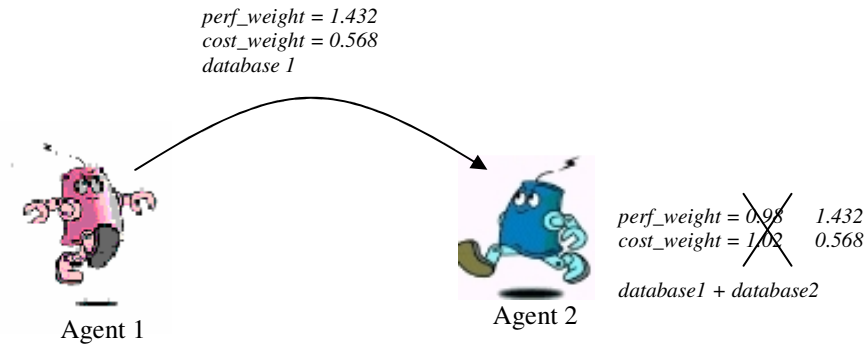
In this example, there are a total of 13 products found inside the agent's database and the product with the highest product value is located in row 13. This will be the top product inside the agent's database. However during feedback, the user might have actually chosen the product in row number 7 instead. Therefore, assuming that the maximum amount of points awardable is 5, the number of points that the agent has earned in this case will be:

$$points\ awarded = \frac{7}{13} \times 5 = 2.692$$

Using this example, it can be seen that if the ranking of the agent- and user-selected products are far apart, the agent will actually receive less points. Besides, if the user-selected product is not inside the agent's database, the agent will not receive any point at all!

### 3.5 Evolution Process

The fitness of an agent will be used to decide which agent will undergo the evolution process. In conventional genetic algorithms, the agent with a higher fitness will have a higher chance of survival as compared to an agent with a lower fitness. However, in this application, there will be a slight variation in the algorithm. Instead of killing the weaker agents, we allow weaker agents to continue and refine by copying over the database of the fittest agent and merging it into its original database to form a larger database. Let *Agent 1* be the fittest agent and *Agent 2* be the weaker one. Figure 7 shows what happens between the agents during this evolution process.



**Figure 7: Evolution process**

However, the parameters acquired by the weaker agent in this evolution process might not necessarily be the optimal. Therefore, the weaker agent will try to adjust its newly acquired *perf\_weight* and *cost\_weight* to better reflect the user's requirements. First, it will use the newly acquired parameters to re-evaluate all the products found inside its new database. Then the agent will select the best product based on these new parameters. If it is the same as the user selected product, no further changes will be required but some small and random mutations in the parameters will be allowed.

However, that will not be the case usually. In this case, the agent will compare the *performance* and *cost* attributes of the products that are selected by the user and the agent. Let  $p1$  and  $p2$  denotes the *performance* of the products selected by the user and agent respectively. Also let  $c1$  and  $c2$  denotes the *cost* of the products selected by the user and agent. Four possible scenarios will have to be considered and they are:

i.  $p1 > p2$  and  $c1 > c2$

The user has selected a product that has a much better performance but more expensive than what the agent has suggested. The agent can deduce that the user places more emphasis on the performance rather than the cost of the product. Therefore, it will increase its *perf\_weight* and reduce its *cost\_weight*.

ii.  $p1 < p2$  and  $c1 < c2$

The user has selected a product that is of a lower performance but cheaper than what the agent has suggested. The agent can deduce that the user places more emphasis on the cost rather than the performance of the product. Therefore, it will reduce its *perf\_weight* and increase its *cost\_weight*.

iii.  $p1 < p2$  and  $c1 > c2$

The user has selected a product that is of a lower performance and more expensive than what the agent has suggested. The agent will be confused over such a selection and will prompt the user if it should carry on the evaluation. If the user still wants the agent to carry on, it will either reduce its *perf\_weight* or *cost\_weight*. This might happen when the user has placed some form of intangible attributes/values on the product, which are not present inside the agent's evaluation function.

iv.  $p1 > p2$  and  $c1 < c2$

The user has selected a product that has a higher performance and cheaper than what the agent has suggested. This scenario will not arise during evolution. Looking back at equation (2), a product with a higher performance and/or a cheaper product will result

in a higher value being assigned to that product. Since using  $p1$  and  $c1$  will definitely result in a higher product value as compared to  $p2$  and  $c2$ , this scenario will not happen.

The evolution process as described in this section is actually quite similar to the use of the reproduction and crossover operator to clone the fitter agents and then use the mutation operator to mutate some of the parameters within the agents.

## 4. SYSTEM EVALUATION

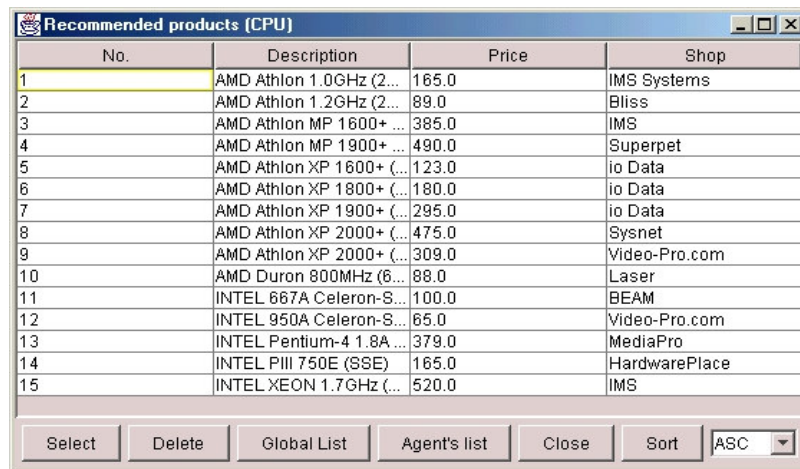
To evaluate the performance of the implemented system, some experiments have been conducted to see if the agents are able to track the user's preference. All the agents in these experiments are running inside a single PC. The PC used in this experiment is a Pentium 4 with 384MB of memory and operating under the Windows ME environment with JDK1.3.0. The system connects to the Internet via a 56.6kbps modem.

### 4.1 Product Recommendation

In this experiment, a group of twenty randomly generated product-brokering agents are instructed to search for one of the products on the Internet. The product chosen for this experiment is *CPU*. For this part of the experiment, the user wants to get the best *CPU* possible and he does not care about the price. After instructing the agents to search for the product, the system is allowed to run on its own for about ten minutes so that the agents can retrieve sufficient products before the user gives feedback. After



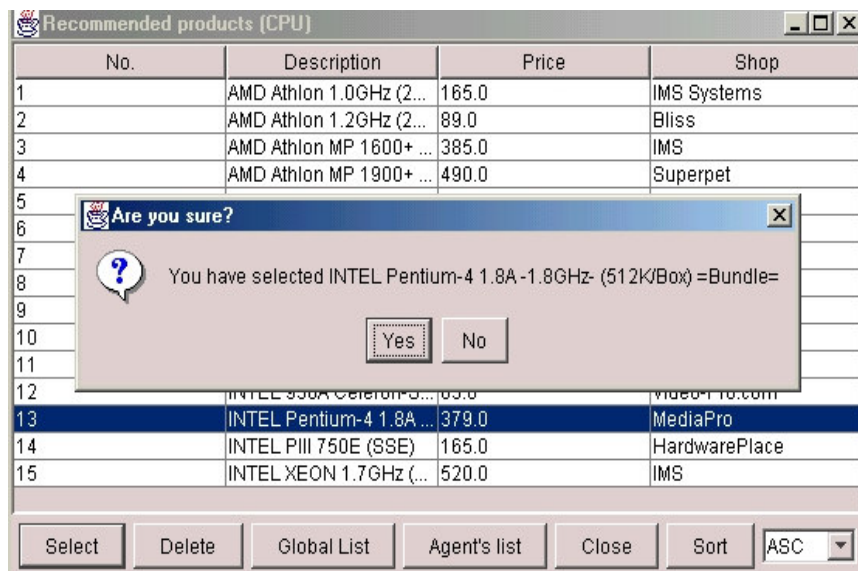
ten minutes, the user clicks on the *result* button and the recommended list is as shown in Figure 8.



No.	Description	Price	Shop
1	AMD Athlon 1.0GHz (2...	165.0	IMS Systems
2	AMD Athlon 1.2GHz (2...	89.0	Bliss
3	AMD Athlon MP 1600+ ...	385.0	IMS
4	AMD Athlon MP 1900+ ...	490.0	Superpet
5	AMD Athlon XP 1600+ (...)	123.0	io Data
6	AMD Athlon XP 1800+ (...)	180.0	io Data
7	AMD Athlon XP 1900+ (...)	295.0	io Data
8	AMD Athlon XP 2000+ (...)	475.0	Sysnet
9	AMD Athlon XP 2000+ (...)	309.0	Video-Pro.com
10	AMD Duron 800MHz (6...	88.0	Laser
11	INTEL 667A Celeron-S...	100.0	BEAM
12	INTEL 950A Celeron-S...	65.0	Video-Pro.com
13	INTEL Pentium-4 1.8A ...	379.0	MediaPro
14	INTEL PIII 750E (SSE)	165.0	HardwarePlace
15	INTEL XEON 1.7GHz (...)	520.0	IMS

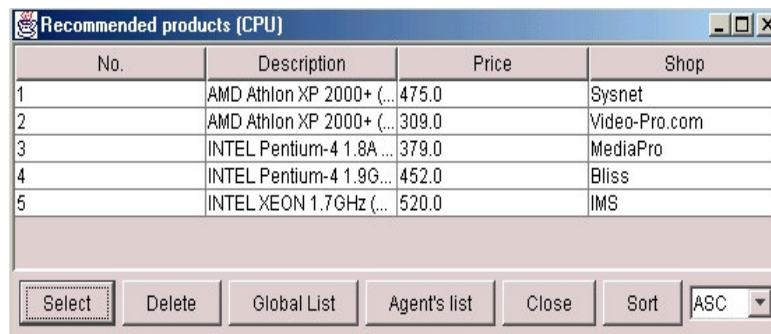
**Figure 8: Recommended list of products**

From the recommended list, the user selects the current best product in row 13, which happens to be Pentium 4 1.8GHz as shown in Figure 9.



**Figure 9: User selection**

While the feedback is been made, the system continues to search for products in the background. After making a few similar selections, the agents evolved and re-evaluated their lists. The new recommended list is now as shown in Figure 10. The list now only shows the best *CPUs* retrieved by all the agents.



No.	Description	Price	Shop
1	AMD Athlon XP 2000+ (...)	475.0	Sysnet
2	AMD Athlon XP 2000+ (...)	309.0	Video-Pro.com
3	INTEL Pentium-4 1.8A ...	379.0	MediaPro
4	INTEL Pentium-4 1.9G...	452.0	Bliss
5	INTEL XEON 1.7GHz (...)	520.0	IMS

**Figure 10: Recommended list after feedback**

When the user is satisfied with what the system has learned, he allows the system to carry on searching the Internet for new products on its own. After some time has passed, the agents have found an even better performing *CPU* and it is reflected in the agent's recommended list, as shown in Figure 11.



No.	Description	Price	Shop
1	AMD Athlon XP 2000+ (1....)	475.0	Sysnet
2	AMD Athlon XP 2000+ (1....)	309.0	Video-Pro.com
3	INTEL Pentium-4 1.8A -1...	379.0	MediaPro
4	INTEL Pentium-4 1.9GH...	375.0	Video-Pro.com
5	INTEL Pentium-4 1.9GH...	452.0	Bliss
6	INTEL Pentium-4 2.0A -2...	545.0	Global IT Mart
7	INTEL XEON 1.7GHz (25...	520.0	IMS

**Figure 11: New products recommended by the agents**

Two other scenarios have also been tested on the system. One of them is to search for the cheapest *CPU* available while the other is to find a right mix of *performance* and *cost* for the user. The steps used in these scenarios are similar to those used in the first part of the experiment and the results obtained are valid.

## **4.2 Tracking user's preference**

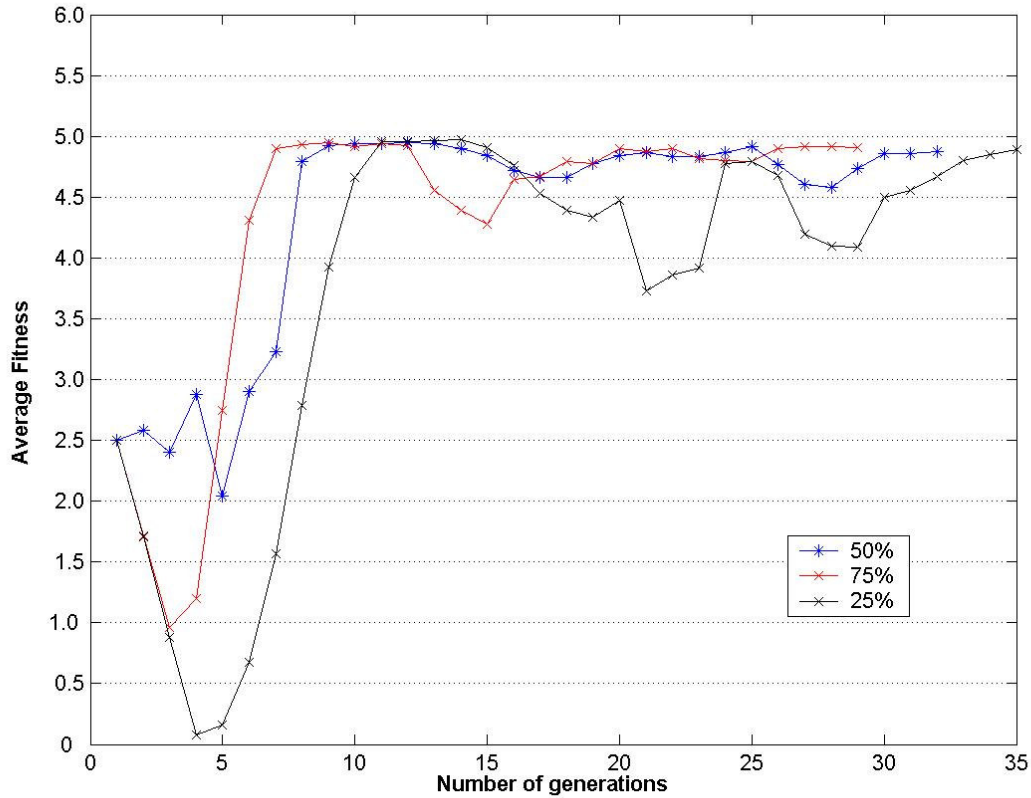
In this experiment, the objective is to test if the system is able to detect a change in the user's preference. If it is able to do so, we need to determine how fast the system will be able to respond to these changes. This could be observed by looking at the average fitness of all the agents in the system. The average fitness of all the agents should remain high if the system is able to track and respond to the changes effectively.

An initial population of twenty randomly generated agents is created and the response of the system is observed by changing the number of agents to evolve in the population. For each test case, the same set of test data are used and will be described in the next few sections.

### **4.2.1 Gradual changes in user's preference**

In this part of the experiment, the user starts by selecting the best *CPU* available. After a few selections, the user will gradually choose cheaper and cheaper *CPUs*. The experiment stops after all agents begin to recommend the cheapest *CPU* available. Figure 12 shows the average fitness of the agents, when the user gradually changes his

preferences. The percentage values in the figure indicate the percentage of population allowed to evolve during each generation.



**Figure 12: Tracking gradual changes in user's preferences**

The results obtained from this experiment have shown that the system is capable of tracking gradual changes in the user's preferences. Although some “dips” are observed during the experiment, the average fitness of the agents in the system remains high while the user changes his selection. These dips could happen because some of the agents might not have in their database the products selected by the user. Therefore, these agents do not receive any points and could “pull down” the average fitness quite significantly.

### 4.2.2 Abrupt changes in user's preference

In this part of the experiment, the user makes two abrupt changes in his preferences. Initially, the user starts by selecting the best *CPU* available. After a while, he abruptly changes his preferences by selecting the cheapest *CPU* and then reverts back to his original selection. Figure 13 shows the average fitness of the agents when the user changes his preferences abruptly.

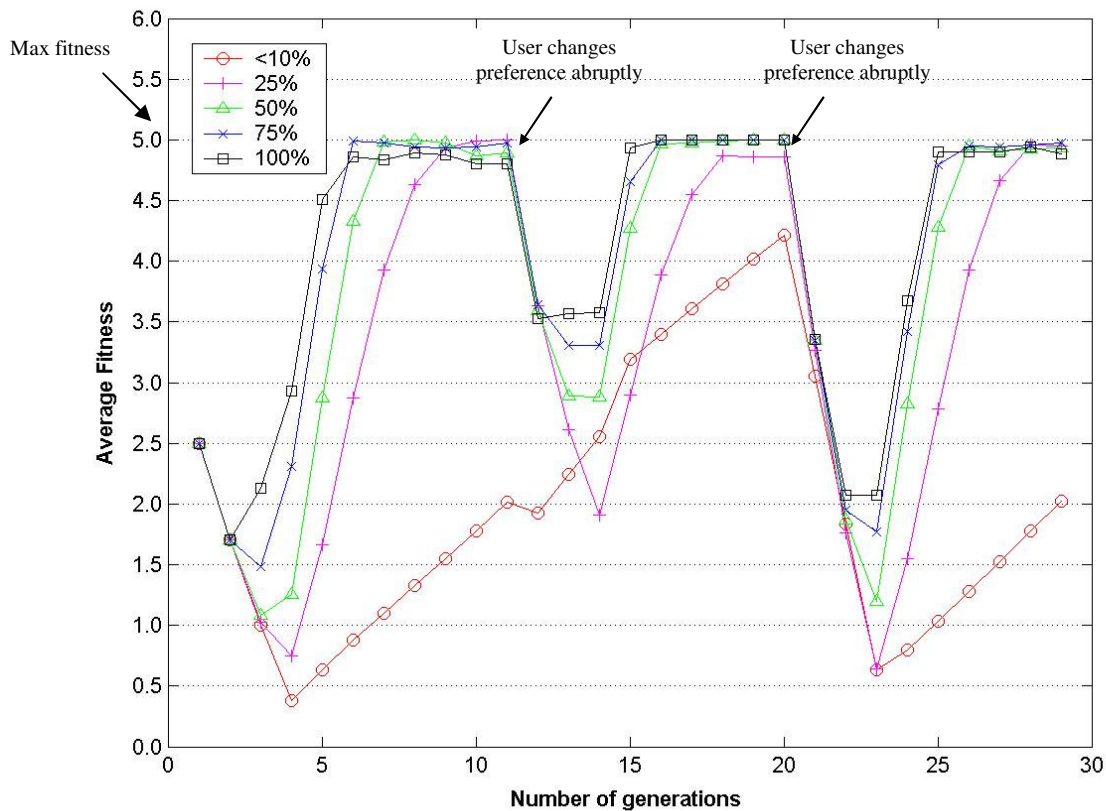


Figure 13: Tracking abrupt changes in user's preference

Interestingly, the results obtained suggested that as we increase the proportion of agents to evolve, the response of the system would be much better. The best result is obtained when 100% of the population is allowed to evolve. It can be seen that the

system in this scenario attains the maximum average fitness in a shorter time as compared to others. Also, when the user makes an abrupt change in his selection, the average fitness of the system does not drop as much compared to the rest and it recovers much faster. The system could not track the user's preference when less than 10% of the population is allowed to evolve.

Although allowing a larger proportion of agents to evolve will result in a faster response. It might cause the system to converge to a solution pre-maturely. This has been observed when 100% of the population is allowed to evolve. The type of products recommended by the agents is not as diverse as compared to when only 50% of the population is allowed to evolve.

Hence, there is a tradeoff between them. From the experiments, it has been observed that allowing 50% of the population to evolve will be a reasonable compromise between the response time and the diversity of products recommended.

### **4.3 Limitations of the System**

Although preliminary test results have shown that *Handy Broker* is quite successful in tracking the user's preferences after a few evolutions, it has been noticed that sometimes, it does not follow the user's selection. A possible explanation for this could be due to the mismatch of the evaluation function that is used by the agent to calculate the perceived value of a particular product. Currently, the function only takes into account the performance and cost of a given product. However in reality, there

might be other product attributes, which could also affect the user's decision. Some of them could also be intangibles, which might be difficult to represent in the function.

## **5. M-commerce Applications**

The proposed design of a product-brokering agent has been implemented using Java in a desktop computer. However, mobile devices such as phones and PDAs tend to have a smaller screen, slower processors as well as limited memory. Hence, this will pose some serious constraints when we transfer the software into these mobile devices. There is also a serious lack of standardization as these mobile devices use different OS platforms, which makes it difficult for the developers to create a single program that can run on all devices.

After taking all these into consideration, a possible solution is to use a software that is compatible across multiple operating platforms. A good candidate is Java, which has been used to implement the system as mentioned in this paper. Java is a platform-independent software technology, allowing the same code to be run on any system. This greatly reduces the time and cost of software development and is particularly attractive for Internet or local network applications. However, the disadvantage of Java as compared to other programming languages, such as C, is its less efficient and slower program execution. Faster processors and more memory are needed to compensate for this. This results in higher cost, and, for wireless applications, shorter battery life. However, this disadvantage has been slowly reduced by the introduction of more efficient JIT (just-in-time) compilers. Currently, the developers of Java have

also introduced some highly optimized and micro version of the Java software to cater for small devices such as cellular phones and PDAs.

As for hardware, Philips Research<sup>7</sup> has developed a co-processor that improves the execution of Java embedded software by a factor up to 10. This obviates the need for powerful processors and/or more memory that this programming language often requires, while maintaining the advantage of enabling fast and economic product development and easy integration with the Internet. The type of invention supports the use of Java and related languages in applications ranging from smart cards to mobile phones and set-top boxes.

A PDA is an ideal device for m-commerce applications. It tends to have a larger screen and a more powerful processor as compared to a cellular phone but is less bulky than a laptop. Making an existing application viewable in any wireless device, a process known as transcoding, is among one of the biggest challenges of m-commerce. In order to fit into the screen of a PDA, the GUI implemented in this paper will have to be scaled down to the appropriate size. A possible solution is to use scrollbars that allows the user to scroll the GUI. A possible screenshot of a PDA with the GUI is shown in Figure 14.

---

<sup>7</sup> <http://www.philips.com.sg/news.shtml#5January>





Figure 14: Screenshot of a PDA with the implemented GUI

The PDA selected for our application is the Compaq iPAQ Pocket PC H3870<sup>8</sup>. It has one of the largest viewable screens in the market and also an integrated Bluetooth for wireless links to Bluetooth-enabled cellular phones. This device also supports the Java Virtual Machine, which will allow our software to be integrated into the PDA easily.

## 6. CONCLUSIONS AND DISCUSSIONS

This paper has demonstrated that by using genetic algorithms and an evaluation function, it is possible to design and implement an intelligent product-brokering agent for m-commerce applications.

<sup>8</sup> ® iPAQ Pocket PC H3870 is a registered product of Compaq.

A prototype *Handy Broker* has been implemented using Java and the preliminary results obtained from the experiments have been encouraging. However, there are some limitations in the current prototype that might hamper the system's performance. More research will have to be done before a truly robust system can be made ready for m-commerce.

One possible improvement to the current work will be to allow the agents to be distributed in a network instead of being hosted entirely by the same computer. As the host computer might not have sufficient resources (processing power, bandwidth, etc.) to support all the agents in the system, it will be advantageous if some of the agents can be hosted by another computer. For m-commerce applications, this would mean the agents could now be hosted by a commercial Internet Service Provider (ISP).

An m-commerce user would not want to spend large amount of money on maintaining a wireless connection to an ISP or a phone company. Likewise, it is unrealistic for mobile devices such as cellular phones and PDAs to be always "online". Currently, some ISPs do provide some form of storage spaces for their subscribers to store files inside their servers. In extension to this, an ISP could now also offer to host agents that have been created by/for their subscribers with a reasonable fee. These agents could perform their tasks inside these servers and report to its user when he re-establishes another connection with the ISP.

However, allowing agents to be distributed over the network will raise some issues, which the developer should look into before the system could be implemented. Since the agents are distributed, some form of communication protocol and *ACL* (Agent Communication Language) will have to be designed and incorporated into the system. One way is to upgrade the monitoring tool that has been implemented in this paper so that it can communicate with remote agents using some *ACL* via a socket connection. Currently, there exist some high-level agent languages such as *KQML* (Knowledge Query and Manipulation Language) [12] and *FIPA ACL* [13], which has been developed for inter-agent communication.

Security issues also arise when agents are hosted by other computers. There is now a need for us to distinguish between agents that are sent by different users. In m-commerce, security is of paramount importance. Sensitive and private information of the user will have to be safeguarded from other hostile entities. This is especially important in the case of mobile agents. As they travel from host to host, we have to prevent them from being intercepted and its contents “core-dumped” by hostile hosts. Malicious agents could also masquerade as the original agent and trick an unsuspecting user into giving up his personal information.

## REFERENCES

- [1] Nwana, H.S., and Ndumu, D.T., An introduction to agent technology, *BT Technology Journal* 14(4), 55 – 67, 1996.

- [2] Aylett, R., Brazier, F., Jennings, N., Luck, M., Preist, C., and Nwana, H.S., Agent systems and applications, *The Knowledge Engineering Review* 13(3), 303-308, 1998.
- [3] Nwana, H.S., and Ndumu, D.T., Research and development challenges for agent-based systems, *IEE Proceedings on Software Engineering*, 144(1), 1997.
- [4] Morris, S., and Dickinson, P., *Perfect M-Commerce*, Random House Business Books, 2001.
- [5] Soltysiak, S., and Crabtree, B., Automatic learning of user profiles – towards the personalisation of agent services, *BT Technology Journal* 16(3), 110-117, 1998.
- [6] Maes, P. Agents that reduce work and information overload, *Communication of the ACM* 37(7), 31-40, 1994.
- [7] Sheth, B., and Maes, P., Evolving agents for personalized information filtering, *Proceedings of the Ninth Conference on Artificial Intelligence for Applications*, 345-352, 1993.
- [8] Zhu, F.M., and Guan, S.U., Evolving software agents in e-commerce with gp operators and knowledge exchange, *Proceedings of the 2001 IEEE Systems, Man and Cybernetics Conference*, 2001.
- [9] Crabtree, B., and Soltysiak, S., Identifying and tracking changing interests, *International Journal of Digital Library* 2(1), 38-53, 1998.
- [10] Soltysiak, S., and Crabtree, B., Knowing me, knowing you: practical issues in the personalization of agent technology, *Proceedings of the International*

Conference on the Practical Applications of Agents and Multi-Agent Systems, 467-484, 1998.

- [11] Pant, G., and Menczer, F., MySpider: evolve your own intelligent web crawlers, *Autonomous Agents and Multi-Agent Systems* 5(2), 221-229, 2002.
- [12] Finin, T., Fritzson, R., McKay, D., and McEntire, R., KQML as an agent communication language, *Proceedings of the 3rd International Conference on Information and Knowledge Management*, 1994.
- [13] FIPA Specifications, Available at <http://www.fipa.org/repository/fipa2000.html>