FRAMEWORK FOR AN AGENT-BASED ELECTRONIC MARKET PLACE: DESIGN AND IMPLEMENTATION

Sim W.T. John, Chi C.H.
School of Computing
National University of Singapore
Singapore 119260
{jsim, chich}@comp.nus.edu.sg

Abstract

This paper describes an architectural framework for the creation of applications based on mobile software agents. As networks grow in popularity, especially the Internet, bandwidth becomes an important issue. At the same time, more and more information is made available on the Internet. This not only aggravates the bandwidth problem but also demands more user search time. Mobile software agents are likely candidate solution to this problem. However, there are currently no common framework for developing agent-based applications. This involves developing an architecture for both the agent as well as the agent execution environment that is flexible, general and able to function efficiently. It is found that many of such Internet-based applications share many commonalties. By having a framework that abstracts away the specific functionality of the application, we are able to extract the common features and thus reuse much of the general architecture. This in turn reduces system design time and makes for more robust systems.

1. Introduction

The Internet (or the Net) is a great source of information. It is also a new frontier for the business world. Over the last few years, it has grown both in popularity as well as in size. However, at the same time, the amount of information available on the Net has simply exploded. It has come to the point where a simple search on a random topic can yield literally thousands of related sites and categories. We can identify two situations at this stage.

Firstly, with the overload of information, it is becoming increasingly impossible to manually search for information. Secondly, more people are getting onto the Internet bandwagon. This creates a potentially vast market place where billions of dollars of trade can take place. In addition, it is an electronic environment, hence transactions can take place in the blink of an eye.

The above situation has resulted in a need for a tool that will free people from the tedious chore of sifting through tons of unrelated and irrelevant data to find the items that they want. And, after successfully sourcing for the required data, the tool should also be able to perform some form processing on that data. All these should be automated as far as possible, thus requiring minimal human supervision.

Consider the following scenario: Peter is to upgrade his CPU and to add more memory (or RAM) to his computer. Now, he would need to check the local yellow pages to find out the nearest computer shops (perhaps about 200-300 entries). He then has to call them up, one by one (perhaps about 30 shops), to find out if they have the items that he wants at a reasonable price. This exercise in patience can easily take 3 to 4 hours. What if Peter had a "servant" that would do all the above for him, so that he is free to attend to other things? And upon his return, his "servant" would be waiting with the prices and addresses of all shops that sell what he wants, or at least, a close match? Such a "servant" is the mobile agent. All Peter has to do is "tell" the agent his preferences for the CPU and RAM, and then send it off to an electronic market place to shop on his behalf. The agent will not only identify the shops that sell the items that Peter wants, it can even "bargain" with the sales personnel for a better deal.

This idea can be extended to situations beyond electronic shopping. For example, a user can describe to an agent the type of news that interest him. He then sends his agent to scour the Net for web pages that might be of interest to him. The agent then downloads those information into his computer, so that he can read them at his convenience, thus saving him time and network connection charges.

This paper describes an architecture to design and implement such an automated search and find scheme. The architecture unifies all the commonality of a large class of Internet-based applications and provides developers with one framework to develop a customised agent-based solution. Additional features of this solution include a negotiation and processing mechanism for more value added services.

2. Design Considerations

One major class of applications on Internet is based on a consumer-supplier architecture. There are suppliers that supply goods and services on the Internet for a fee. Consumers that make use of these goods and services then pay for them. An important requirement for this class of

applications is that a suitable match between consumer and supplier be found in the shortest time possible. In addition, it is also desirable to allow some kind of negotiations between consumer and supplier so that an even more suitable match can be found.

2.1 Security

In any Internet-based application, security is always a major concern. People want to be sure that their private matters are not readily accessible and their privacy is not compromised. If issues of money are involved as in the case of electronic commerce, maintaining tight security is all the more crucial. For this project, a trusted server is assumed. This has the following implications:

- A user has to be registered with the server before his agent has access to the facilities of the server.
- The agent does not have to be protected from the server. Currently, this is the only possible way, as the agent code has to run in the server and thus is always at the "mercy" of the server.

Agent security. An agent should not be allowed to access another agent's private data. That is, each agent should be "fire-walled" against each other. Agent's private data should be secure during transportation, between client and server and between servers.

Server security. An agent should not be allowed to damage the operation of the server. Only legitimate agents should be allowed into the server [Whi95].

Two-way authentication can be used to meet some of the above security requirements. This means that the agent can verify if the server is trusted and thus will not harm it. The server too, can authenticate the agent to verify that it is a valid agent and belongs to a registered user. This can be done using digital certificates [Pfl97].

2.2 Generality

The design should support a wide range of uses. The architectural framework should present a common interface that allows customization to a particular application. Traditional approaches lack a unified view of such applications; this proposal of using mobile agent technology seeks to address this shortcoming. Here, are some examples of the possible types of applications:

- **Web-based Search Engine**. In this application, a user can define the types of information he is interested in, e.g. news about a particular soccer club, latest microprocessor news, any changes in a particular site etc.. He then sends the agent to "crawl" through the web in search of the required information. Traditional technology does not conveniently allow a client program to migrate its executions from server to server to continue its search from where it has left off.
- **Resource Scheduling**. A typical application of this category can be a facility reservation system in a company. A user wanting to book a particular conference room can create an agent with his requirements and send it to a central server to make its request. If that room is unavailable at the required date and time, the agent can consider other alternative rooms that also meet the specified conditions. It can then return to the user to inform him of the changes in plan and to seek final approval. Again, traditional technology lacks an elegant way to make alternative plans if the primary proposal is unfeasible.
- **Stock Market Monitoring**. The essence of this class of applications is the ability to make timely decisions. Even a time lag of less than a minute is unacceptable. Traditional approaches suffer from network latency and thus are unable to meet the real-time requirements of such operations. Agents, on the other hand, are not affected by network congestion during operation as they operate at the server side. Thus they are able to react instantaneously to situations as they arise. All the necessary logic and conditions are with the agent at the server, thus it can make well-informed decisions.

2.3 Portability

The Internet runs on many different hardware platforms and operating systems. Therefore, compatibility issues need to be addressed. A major factor in the success of such a distributed application is its ability to run on as many platforms as possible. Hence such a system should not be tied to a particular hardware/software combination.

2.4 Efficiency

Efficiency can be defined in terms of:

- **time**: the process of search-offer-evaluate-response should not take too long.
- **network requirement**: as the agent needs to be mobile, travelling to various servers, the size of the agent should be kept to a minimum.
- **resource requirement**: as there could be many agents at any one server simultaneously, the resource required by each agent at the server should be kept to a minimum.

3. Agent versus Other Technologies

We will now compare mobile agent technology against other more traditional distributed computing technologies.

We will then look at the advantages that mobile agent technology have over these traditional technologies.

3.1 Description of Remote Procedure Call

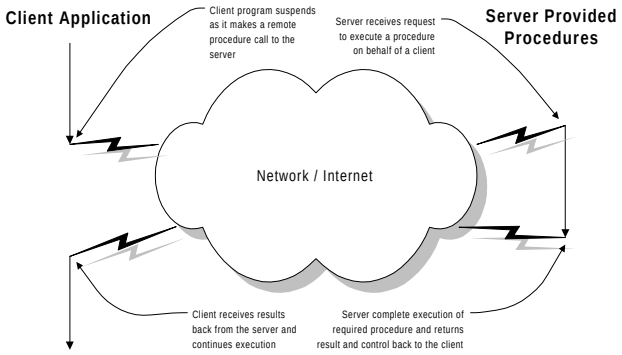

Figure 1: Remote Procedure Call (RPC)

The Remote Procedure Call (RPC) extends the traditional procedure call mechanism of pushing parameters, registers and a return address onto the stack and then performing a jump to the procedure's entry point. In the RPC case (as in Figure 1), the client and server open a communication channel between the client application and the server process. The RPC parameters are passed to an interface routine, which marshals them into a form suitable for transmission and they are then sent explicitly to the server process. The RPC packets are received by a corresponding interface routine, unpacked and passed to the server procedure. The procedure processes the parameters and (generally) produces a return RPC, which is transmitted back to the client process. Both parties must use a common interface definition (although heterogeneity of hardware and operating system software is possible).

While a local procedure call can be executed in at most a few microseconds (not including the execution time of the procedure itself), the RPC introduces overhead due to marshalling, transmission, and unpacking and has a typical latency of a few milliseconds. Like the local procedure call, the RPC is synchronous; the client process suspends, maintaining the entire process state, until it receives the return RPC from the server. Secure RPCs add authentication and encryption facilities to the client-server communication, but introduce significant overhead [Chi92]. The strength of RPC lies in its high efficiency and low latency. Its weaknesses are its need for exact interface matching, the constant need for network connection as well as the synchronous nature of its communication [Che95].

3.2 Description of Messaging

Messaging is emerging as a popular alternative to RPC for client-server communication. It is an outgrowth of both electronic mail systems and earlier distributed computing schemes in which applications communicated via files or pipes. A typical messaging mechanism is shown in Figure 2. The client application composes a message, typically of tagged or structured text, which is to be delivered to an appropriate software processor for the type of message. Messaging systems may employ a message transport service provided by an electronic mail service. The required processor type is indicated in the message header. The message is generally addressed indirectly, that is, the client may not know the explicit network address or even the identity of the destination server. The resolution of addresses is performed by intermediate steps of processing, such as post offices.

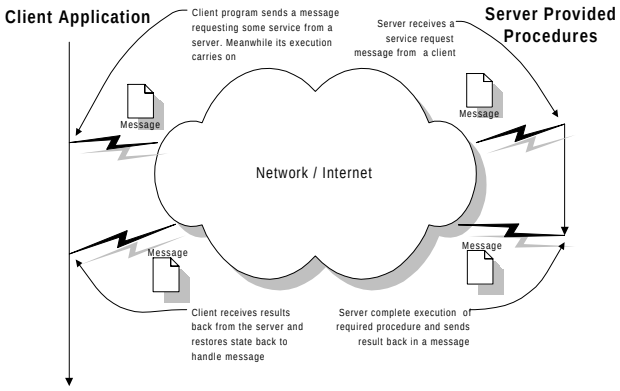

Figure 2: Messaging Mechanism

Messaging is inherently asynchronous; once the client has handed off the message contents to the messaging sub-system, it continues execution. If in the future, the client receives a response message from the server, it must restore the application state in order to process the response. For example, if the user is engaged in a dialogue with a reservation service, several iterations may be required between the client travel planning application and the Computerized Reservation Service (CRS), before the user has identified a suitable flight and seat. During these iterations, the client and the CRS must both maintain the transaction states until it is committed.

The communication of messaging is asynchronous; its latency in messaging is both higher and less predictable than in the case of RPC. As a result, messaging may be less effective for one-to-one communication than RPC. However, for one-to-many communication, which is typical of servers in network services, the throughput may be higher, since the client process does not need to suspend while waiting for the response. As in the RPC case, secure messaging can offer authentication and encryption. There is an equivalent overhead, but because the process is asynchronous, the overhead is less burdensome. The strength of messaging lies in its robustness, especially over wide area networks. Its weaknesses are still its dependency on the network connection, though less than in the case of

RPC. It does not have the flexibility of reacting to real-time conditions especially if they are network-related failures. Furthermore, it does not generalized well to handle a wide range of applications [Che95].

4. Description of Mobile Agent-Based Computing

Such systems are typically composed of client and server side applications. The client application enables the user to create a suitably equipped agent. That agent will then be dispatched to the server (or servers as may be appropriate) to perform whatever tasks that its dispatcher (the user) has instructed it to do. The key difference of agent-based computing is that the agent has a much higher level of autonomy as compared to competing technologies. Possible applications of this emerging technology include intelligent information searching and retrieval, electronic mail clients and electronic commerce applications [Bre97].

Agents can be created using either machine language (called compiled agents) or an interpreted language (called interpreted agents). As mobile agents usually execute on more than one server (and hence multiple hardware/software platforms), support for heterogeneity is an important consideration. Therefore, it is often preferable to express the agent in an interpreted language. There is a performance penalty for this, but since most of the processing is done at the server (which is usually much more powerful than the client), this maybe acceptable. This approach also has the advantage of late binding: it enables the agent to contain function stubs that will bind to the actual functions that are either provided by the server or other agents found at the server. Higher levels of security are also possible with the interpreted agent approach as the agent language developer has more explicit control over what system resources are available to the agent [Mic94].

4.1 Advantages of Agent Technology

In this section, we will look at some of the advantages of the mobile agent-based paradigm. It might be noticed that most of the stated advantages can also be accrued to one or more existing distributed programming paradigms. However, the point to note is that no existing technology has all these benefits presented in one common framework.

4.1.1 Low Requirement for Network Bandwidth

During data processing, there may be a lot of raw information that needs to be examined to determine their relevance. Transferring this raw information between server and client can be very time-consuming and cause excessive network congestion. Imagine having to transfer 100MB of images just to pick out one or two that meets the required specification. It is much more natural and efficient to "go" to the location where the images are stored and do a local search/ pruning. After that, only the chosen compressed images will be transferred back across the network. It obviates the need for constant costly network connections between remote computers as required in remote procedure calls (RPC) and sending/ receiving of messages as in messaging. It also provides a much cheaper alternative as we pay increasingly for network bandwidth and time.

4.1.2 High Bandwidth Communication with Server

In conjunction with the above advantage, mobile agents can support high bandwidth communication with the server without the associated high usage cost. This is because the agent is communicating with the server at the server's site itself and not over the (slower speed) network. This makes agent technology suitable for real-time processing of information where network latency is unacceptably high.

4.1.3 “Lean Clients”

The processing power and storage on the local machine may be very limited (only perhaps for processing and storing the results of a search). Therefore, requiring that the entire database be downloaded first before processing would be extremely costly in terms of network bandwidth, time as well as local system resources. This might not even be possible for mobile clients like most notebook computers, and even more so for palmtop computers.

Agents provide a better alternative: by allowing the agent to perform the searching and filtering of information at the server before downloading it to the client. In addition, the client's processing power is generally much less than that of the server. Thus a complex query can be completed much faster when executed on the server than on the client.

4.1.4 Autonomy

Agents are allowed a certain amount of autonomy. Based on some parameters supplied by the user, the agent is left to make its own decisions as to the suitability of the current situation. For example, consider a Web searching application. The user might supply the agent with a list of keywords and sites to try searching for relevant Web pages. The agent can use the list of sites as a possible itinerary and visit each site in the order of least loaded. Upon reaching each server, it then begins its search. Each way it searches could be different depending on the available methods of searching at each server. It can then intelligently extend or restrict the keywords used subjected to the number of documents requested by the user.

All these decisions are guided by parameters supplied by the user during agent creation and can be made without further consultation of the user, hence the autonomy.

4.1.5 Asynchronous Communication Support

The only time a user is actually connected to the server is when he is dispatching the agent, or receiving the returning agent and its payload. At all other times, the user does not need to be connected to the server. This requires much less network time as compared to other existing technology. In addition, during the execution of the agent, the user need not be online, as the agent is able to operate autonomously at the server side, thus further reducing network requirements.

A typical scenario is this: during agent creation, the user works on his local machine and does not require a server connection. After dispatching the agent to the server, the user can log off while the agent will continue its work on the server. Then when the agent has completed its work, it can deposit its results in the user's mailbox, awaiting the user when he returns. Another benefit is that it frees up the user to perform other more necessary tasks, while the agent carries out its tasks.

4.1.6 Agent Collaboration / Competition

An important (though not necessary) aspect of the mobile agent computing paradigm is the possibility of agent collaboration. In simple terms, it is having many simple and perhaps specialized agents coming together to "discuss" the problem and to find a solution. Agents can exchange information about specific topics or situations encountered and this additional information can be used to plan further actions. An example of such a scenario is that of using software agents to monitor the network load and to provide dynamic routing of packets depending on the network congestion.

Instead of collaboration, agents can also be in competition with each other. A typical example of this would be the electronic market place. Each agent is trying to maximize its benefits by trying to outbid each other and closing the deal before its competitors. This ability of agents to collaborate or compete is what mainly sets it apart from other traditional approaches.

4.2 Comments

Existing technology can only claim to possess a subset of the advantages that mobile agent-based computing possesses. No one technology has all the above stated advantages in one common framework. Therefore, the main advantage that the mobile agent computing paradigm provides is an amalgamation of all the features achieved by existing technology. This provides a unified, consistent framework for achieving all the stated benefits.

In other words, the strengths of mobile agents may not be any individual potential class of applications, each of which could alternatively be implemented using a more traditional distributed processing scheme. Rather, mobile agents should be used to develop a family of applications. They should provide users a single, flexible way to implement and reap the benefits of the traditional distributed processing and client/server models.

5. Proposed Architecture Description

As there are many possible applications that can be designed, the proposed architecture should be flexible enough to handle the variations. It should also perform efficiently in most cases while at the same time provide adequate security features for applications that require it.

There are 4 elements to our proposed architecture. The agent execution environment (or simply the server), the agent itself, the communication protocol and the object description mechanism.

5.1 Agent Execution Environment (Server)

The agent execution environment is one that promotes interaction and collaboration between agents. It provides various necessary services to agents. In addition, it handles user requests for information such as the status of his agent (e.g. what items are left un-transacted). Main components of this environment include:

- **Gateway**: This is used to allow the entry and exit of agents from the server. It also provides authentication and verification services in collaboration with the **Security Manager** component to ensure that the agents are valid (i.e. they belong to a registered user) before they are permitted to enter and trade in the server. [Liu98]
- **HoldingArea**: This is used to hold agents that have arrived at the server. All active agents reside in their respective holding-areas. The holding-area also provides a mapping of **agentID**s to the agents themselves. This is used in the addressing scheme when agents need to communicate with each other.
- **BlackBoard**: This is where agents advertise what items they wish to transact or what services they require/offer. Agents will advertise in their respective blackboards (e.g. buyer and seller blackboards respectively). Agents then use this blackboard to identify potential customers and initiate negotiations with them.
- **Security Manager**: This is used to hold all passwords of users that are registered at the server. It is also used by the **Gateway** to authenticate incoming agents and during log-in sessions by users to the server.

- **Log**: This is used to record various significant events that occur, like the arrival and departure of agents, the successful transaction of an item etc.. The log records down the time of the event and gathers various statistics like the allocation of items, the last transacted price of an item and other information as appropriate.
- **GUI**: This is the user interface to the server. It allows the administrator to set various system parameters like the number of agents allowed in the server at any one time, whether logging is enabled and what the name of the log file is etc.

5.2 Mobile Agent

The design of the mobile agent is both reactive as well as proactive. It is reactive in that it reacts to messages sent to it, and takes necessary actions to handle that message. It is proactive in the sense that it maintains its own agenda. Thus when there are no messages to handle, it will actively seek out other suitable agents to begin negotiations with, in order to fulfill its agenda. In the following diagram (Figure 3), the agent is used in an electronic shopping application, with computer items to transact.

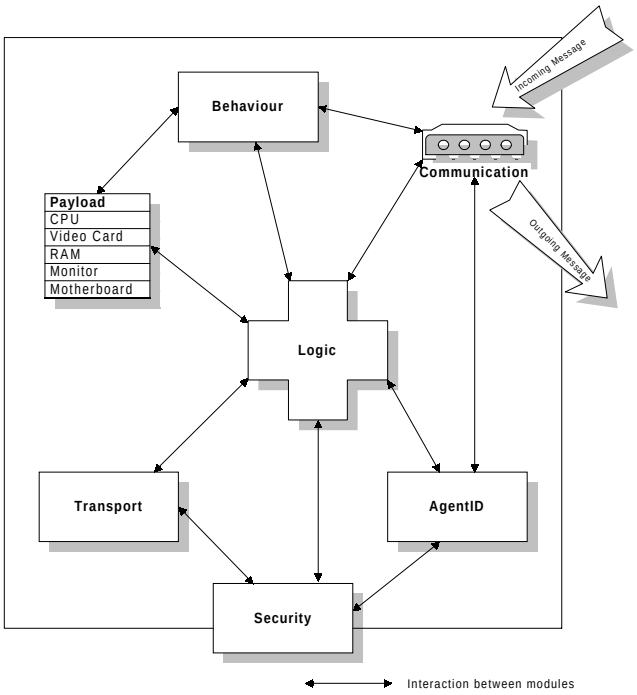

Figure 3: Agent Structure Used in Electronic Shopping

- **Logic**: This is used to control what the agent does next, in response to a message. It also determines the proactive nature of the agent. This component comes from the server and is thus trusted code. As this determines what the agent can do in the server, depending on its type, the user is unable to harm other agents or the server. Extensive testing can also be done on an off-line server to minimise programmer errors. In addition, by simply providing a different type of **Logic** component, the agent can easily assume a completely new role. For example by changing the **Logic** component to one that searches the web for information of interest to its user, the agent becomes a Web-crawler. (Of course, the method of evaluating the suitability of what the agent finds (i.e. the **Behaviour** component) also has to be appropriately modified to evaluate the new data item.) Therefore, only the components need change, but not the architecture, as long as they adhere to the original interface. This makes the agent generic.
- **Behaviour**: This is used to evaluate the various offers that are sent to the agent. The user supplies various parameters that determine whether a particular item is what the agent wants. For example, the user might indicate that he prefers an AMD microprocessor to an equally clocked Intel one. Thus during the evaluation of an offer, the agent will place more weight on an AMD processor as compared to an Intel one. [Chi98]
- **Valuation**: This is used to return the **Behaviour** component's assessment of a given **Template**. For most purposes, 3 types of **Valuation** can be returned. The first one is the evaluated **Template** that matches the requirements sufficiently closely and thus a successful match is found. The next possibility is that the evaluated **Template** may be too unrelated to be further considered and thus a match failure ensures. The last possibility is that the **Template** is not too far from the requirements and perhaps negotiations between the involved agents can result in a compromise and then a successful match can be found. The proposed types of **Valuation** are: ValuationSuccess, ValuationFailure and ValuationRetry. ValuationRetry is used to return a counter-offer to the current offer. This counter-offer is then relayed to the other agent.
- **Payload**: This is used to carry a list of items the user is interested in. For example in an electronic shopping application, it can be viewed as both an electronic shopping cart and shopping list. It provides facilities to skip to the next item in the event that there are no suitable offers for the current one. This prevents the agent from getting "stuck" waiting for a suitable offer for the current item while there could be other offers for its other items.
- **Transport**: This is used to facilitate the transportation of the agent from the client to the server as well as between servers. It also stores the itinerary of (possible) servers that the agent can visit to fulfil its tasks. [Liu98]

- **Communication**: This is used to handle all communication-related matters of the agent, like the queuing and processing of messages. It also tracks the status of communication (e.g. last message sent, the sender ID) between the agent and all other agents it has dealings with. This can be used by the **Behaviour** component to try and obtain a more globally optimal deal. This component processes multiple communication on a time-sharing basis and allows the agent to find a suitable candidate faster for greater efficiency.
- **EstablishedComm**: This is used to track the status of communication with a particular agent. Each communication with a particular agent is stored in this component. This consists of the last message sent and the last message received.
- **Security**: This is used to authenticate requests to modify the internal status of the agent (e.g. change the agent's password, change an agent's component) or requests that causes the agent to perform certain special actions (e.g. moving to another server). Any changes to the components (e.g. assigning a new **AgentID**, or changing a new **Payload** object) require the agent's password. This is to prevent other agents from sending illegal harmful messages to it.
- **AgentID**: This is used to uniquely identify an agent in a particular server. All references to an agent are made via its **AgentID**. Hence an agent is isolated from other agents as a further security measure.

5.3 Communication Facilities

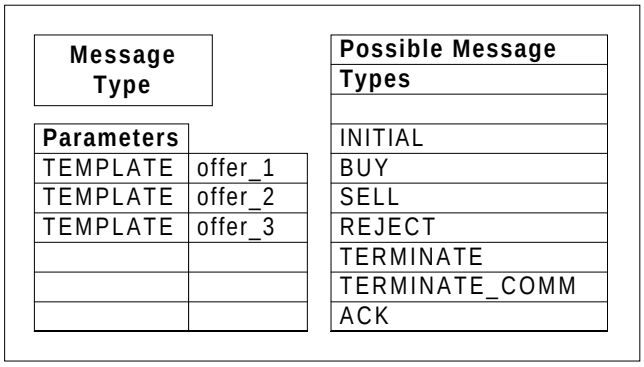

Figure 4: Message Object Structure

This is used to carry information between agents, between agent and server and (possibly) between servers. A message is made up of a *message type* followed by a list of *parameters*. This is much like a machine instruction, with an opcode (message type) and a list of parameters. The set of valid message types are found in the message component. The design of this object is very generic so that it can handle a wide variety of different messages, in support of the possible various applications. Each parameter tag is a STRING while the actual value corresponding to that tag is an OBJECT. A typical message object structure is shown in Figure 4.

5.4 Object Description using Templates

This is used to represent items that are of interest to the user (and hence the agent). It contains the properties that characterise the item. For example, in an electronic commerce application for the sale and purchase of computer parts, a CPU can be defined by its clock speed, whether it has MMX extensions, its brand etc. Usually, the server will provide these templates and the user simply fills in the appropriate values for the item's properties. This design is also very generic, relying on STRING tags to identify the applicable properties of the item and its associated values. Thus it is left to the **Logic** and **Behaviour** components to interpret those values.

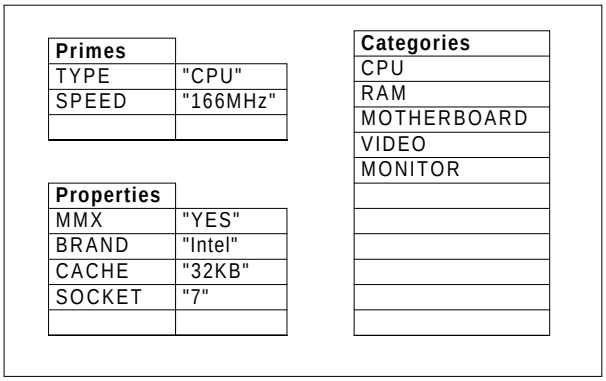

Figure 5: Template Object Structure in Computer Parts Commerce Application

6. Benefits of Architecture

6.1 Logic Component Supplied by Server

The typical approach to mobile agent design is to have all processing logic within the agent itself. However, this approach has some drawbacks:

- **Agenda of Agents:** The agenda of the agent could be questionable, thus posing a security risk. The agent could be a rogue agent and cause damage to the server or to other agents. This could have been done intentionally (as in a case of a virus or a Trojan horse) or unintentionally (due to errors in processing logic).
- **Incompatibility Between Agents:** As the system develops, there will be newer versions of agents with added functionality. Hence, when agents communicate with each other, they have to perform version checking

and to revert to the lowest common denominator. This nullifies the advantage of having a newer agent version with better processing capabilities.

- **Size of Agents:** The size of the agent will be considerably larger with all that extra code it is carrying around. This makes agent transportation more expensive both in terms of network bandwidth as well as time.

By making the **Logic** component available at the server, the following benefits can be obtained:

- **Lower Security Risk:** As the processing logic of the agent comes from a trusted source, an agent can be considered far more secure.
- **Rapid Availability of New Functionality:** Whenever new functionality is made available at the server, all agents requesting that **Logic** component will have immediate access to the new functionality. This also means that there will not be incompatibilities between agents, as all agents will have the latest version.
- **Reduced Agent Size**: Having much of the processing logic provided at the server will mean that the size of the agent can be significantly reduced and will comprise mainly of its payload and core functionality, like transportation functions.

6.2 Behaviour Separated from Logic

By having the **Behaviour** component of the agent separated from the **Logic** component, the user is able to tailor the evaluation of various offers to suit his needs. The "What to do" (which is determined by the **Logic** component) for the same type of agent will always be the same, hence it is more efficient to have only one common copy which is made available at the server. However, the "How much is this item worth" is very dependent on the individual user's preference and situation, thus it should be separated from the **Logic** component. What we have now is the common **Logic** component coming from the server and the **Behaviour** component from the user (or the client program).

6.3 Modular design

By putting various functionality of the agent and server into various (to varying degrees of) independent components, we are able to modify functionality relatively easily (as long as they conform to the pre-defined interfaces). This allows components to be upgraded as and when necessary to provide for new functionality or, as bug fixes. This follows closely to the idea of functional adapters, where a certain component (**Logic** and **Behaviour** components in this case) can be switched after the agent has been created and therefore can take on different roles.

7. Sample Implementation

In order to illustrate the suitability and potentials of the proposed architecture, a sample application was implemented. It is an electronic market place used for the trading of computer parts.

7.1 The Big Picture

This implementation is one of the components in the overall scheme of our electronic commerce project. The following diagram shows how this electronic market place fits into the whole concept of electronic commerce. Essentially, this is a sort of searching problem, albeit a complex one, with multiple "fuzzy" parameters. Fuzzy because the target items may not be exactly what a client wants but a set of possible choices.

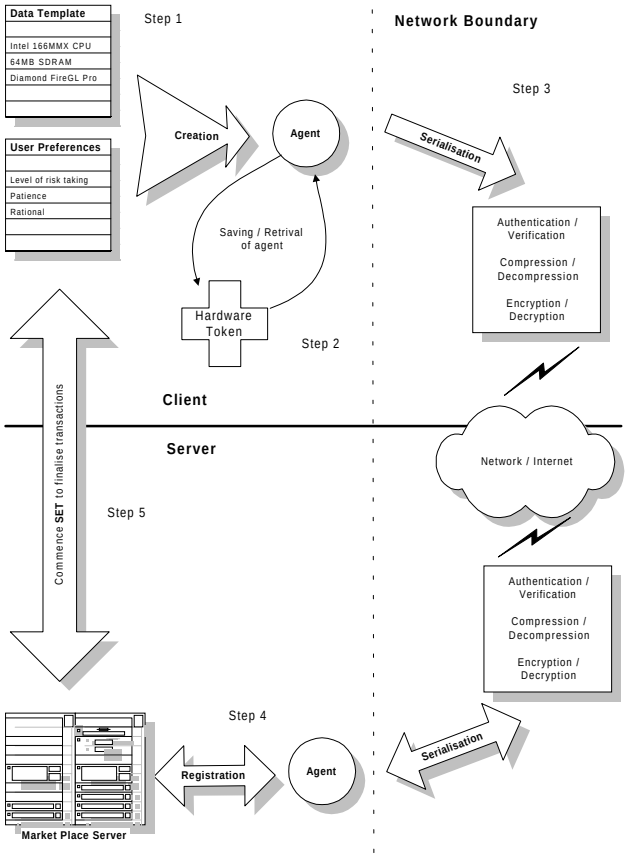

Figure 6: The Big Picture of Electronic Commerce

In addition, the search process involves bargaining with other agents in order to obtain a better deal for the items of interest. All this is done without the need of human intervention except for the finalisation of the transaction. This is the significant difference between this

system as compared to other electronic commerce system. The following diagram (Figure 6) illustrates this new concept.

The whole process begins with a user. Let us consider the scenario where a user wants to buy some computer parts, namely a Pentium 166MHz CPU, a Tseng Labs ET6000 Video card and 32MB SDRAM. The purchase process will then go through the following steps:

1. The user first creates an agent and instructs it on what are the items he wishes to buy. He can also instruct the agent on some of the possible market place servers to try when looking for potential sellers of the items he is interested in. In the client software, he can also simulate the trading behaviour of the agent to determine if that is the behaviour that he wants [Chi98].
2. After agent creation, the user has to deploy his agent. There are two ways to do so. Firstly, he can deploy his agent via the network [Liu98]. Secondly, he can save a copy of the agent in a token like a smart card and carry it around with him till he finds a suitable server to deposit his agent [Ngw98].
3. Each agent has to be registered before being allowed into the server to trade. The working of the server is the focus of this project and will be covered in more detail in the next section.
4. Trading is a two step process. The first step is to identify potential agents that it (the agent) can deal with. It looks for agents that carry the required category of items (i.e. CPU, Video card and RAM). Next, it then begins bargaining with the group of short-listed agents to try and find a suitable price to which both agents can agree on. In the event that the agent is quite unsuccessful in this particular server, it can decide to move to another server to "try its luck". The user leaves the decision of "where to go next" to the agent, depending on prevailing conditions (like server/ network loads, success rate at current server etc.)
5. Finally, after finding all the items required or reaching some user-specified time-out value, the agent can either return home or simply terminate itself. The user then uses the SET (Secured Electronic Transactions) [Vis97] protocol to perform payment online and the whole process is completed.

7.2 Implementation Description

With the overall concept described in the last section, let us examine how the market place server works in a little more detail. The following diagram (Figure 7) illustrates the working of the server.

When an agent arrives at a server, the server will advertise the agent's **Payload** of items to be transacted on a **BlackBoard** and then place it into a **HoldingArea**. The agent will then actively look for possible candidate agents to begin negotiations with. Using the earlier example, the agent wants to buy a CPU, a Video card and some RAM. It will look up the Seller **BlackBoard** (where seller agents advertise what items they have on sale) under the appropriate categories (CPU, VIDEO and RAM in this case) for the items it wants.

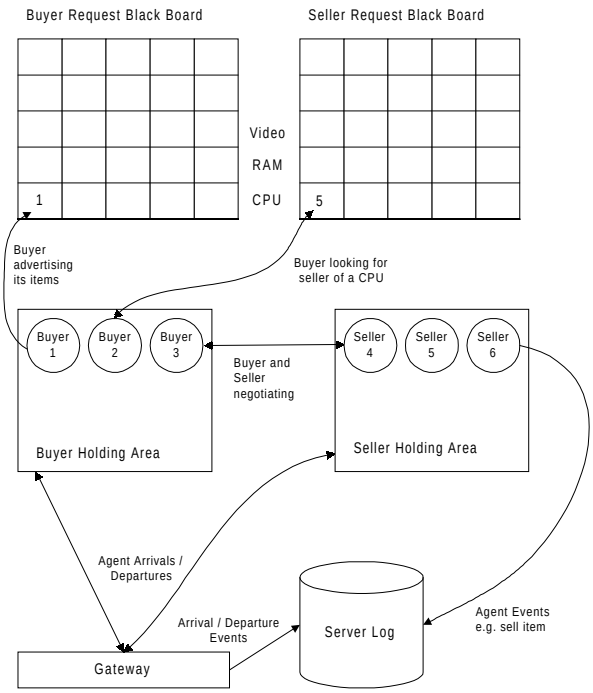

Figure 7: Market Place Server

When it identifies a potential candidate agent, it will initiate negotiations with it by sending it an offer for the item that it is interested in. The receiving agent will evaluate the offer and then return a response. This process of search-offer-evaluate-response will continue until certain terminating conditions are met. Examples of such conditions are when the agent has successfully bought or sold all the items it wants; or after much unsuccessful transactions it could decide to give up and try some other market place. The **Logic** component of the agent is responsible for deciding what action to take, given the current situation.

After completing its job, it can then return to its user with the results. It will then be up to the user if he is satisfied with the results. If he is satisfied with what the agent has found, he can finalise any payments via the SET protocol (as defined by VISA and MasterCard) [Vis97]. It should be noted that human intervention is required for the actual confirmation of transaction and the initiation of

payment. This will relax the security requirements of the market place and make it more cohesive.

8. Conclusion

This paper proposed an architectural framework for the construction of a popular class Internet applications. Various design issues in the development of such applications have been raised. The shortcomings of some existing paradigms at providing developers with a unified framework for such development work area were also highlighted. This was followed by a solution in the form of the mobile software agent paradigm. More details of the proposed architecture and its associated benefits were provided and a short description of a sample implementation was given.

Agent technology is still in its infancy when compared to the client-server model or RPC. More work needs to be done before serious adoption of this emerging technology can take place. Prior works include IBM's Aglets [Osh96] and its Workbench (agent creation/ execution environment) [Lam96], XPect: A Framework for Electronic Commerce [And97] and General Magic's Telescript [Whi96]. A survey [Kin97] of existing mobile agent system has yielded some interesting results. Plangent [Aki97] is another approach to the development of intelligent agents, while some aspects of agent security are addressed in this paper [Gun97]. So, areas that need more work include agent creation and communication protocol standards (e.g. are KQML [Fin93]), encoding of user requirements, agent execution environment standards and various security issues. Until these issues are adequately addressed, mobile agent computing will remain within the confines of the laboratory. However, the potential benefits to be reaped with the deployment of mobile agent technology are huge and thus warrant further research.

References

[Aki97] Akihiko Ohsuga, Yasuo Nagai, Yutaka Irie, Masanori Hattori, and Shinichi Honiden, "Plangent: An Approach to Making Mobile Agents Intelligent," *IEEE Internet Computing*, Vol. 1, No. 4, July/August 1997.

[And97] Jean-Marc Andreoli, François Pacull, and Remo Pareschi, "XPect: A Framework for Electronic Commerce," *IEEE Internet Computing*, Vol. 1, No. 4, July/August 1997.

[Bre94] Bret Summers, "Agents: Not just for Bond," *JavaWorld*,. April 1997. [http://www.javaworld.com/javaworld/jw-04-1997/jw-04-agents.html]

[Che95] Chess D, C. Harrison and A. Kershenbaum, "Mobile Agents: Are they a good idea?" *Technical Report. IBM T. J. Watson Research Centre N.Y.*, March 1995.

[Chi92] Chi Ren Tsai and V. D. Gligor, "Distributed Systems and Security Management with Centralized Control," Proce*edings of the Spring 1992 EurOpen/USENIX Workshop*, April 1992, pp. 137-146.

[Chi98] Chionh Wei Peng, "Agent Negotiation for Electronic Commerce," *Honour Year Project Report, Department of Information Systems and Computer Science, National University of Singapore*, March 1998.

[Fin93] Tim Finin and Jay Weber et. el., "Specification of the KQML Agent," *Communication Language*, June 15 1993.

[Gun97] Gunter Karjoth, Danny B. Lange and Mitsuru Oshima, "A Security Model for Aglets," *IEEE Internet Computing*, Vol. 1, No. 4, July/August 1997.

[Kin97] Joseph Kiniry and Daniel Zimmerman, "Survey: Special Feature: A Hands-On Look at Java Mobile Agents," *IEEE Internet Computing*, Vol. 1, No. 4, July/August 1997.

[Lan96] Danny B. Lange and Daniel T. Chang, "IBM Aglets Workbench (Draft): A White Paper," *http://www.trl.ibm.co.jp/aglets/whitepaper.htm*, 1996.

[Liu98] Liu Jian, "Compact Mobile Software Agents," *Honour Year Project Report, Department of Information Systems and Computer Science, National University of Singapore*, March 1998.

[Mic94] Michael R. Genesereth, Steven P. Ketchpel, "Software Agents," *Communications of the ACM*, Vol. 37, No. 7, July 1994.

[Ngw98] Ng Wei Kai, Bennet, "Smart-card Driven Framework to Agent-based Electronic Commerce," *Honour Year Project Report, Dept. of Information Systems and Computer Science, National University of Singapore*, March 1998.

[Osh96] Mitsuru Oshima and Gunter Karjoth, "IBM's Aglets Specification (Draft)," [*http://www.trl.ibm.co.jp/aglets/spec_alpha5.html*], 1996.

[Pfl97] Charles P. Pfleeger, Chapter 4 : Protocols and Practices, in *Security in Computing*, 1997.

[Vis97] VISA and MasterCard, SET Specification Version 1.0, May 31, 1997.

[Whi95] John White, "Telescript Technology: The Foundation of the Electronic Market Place," *General Magic White Paper*, 1995.

[Whi96] John White, "Mobile Agent White Paper," [*http://www.genmagic.com/agents/Whitepaper/whitepaper.html*], 1996.