DECLARATION

This project, as presented in this report, is my original work and has not been presented for any other University award.

Sign: [illegible] Date: 22/06/2011

JACKSON MWITI
P58/70485/2008

This project has been submitted as partial fulfillment of the requirements for the Master of Science degree in Computer Science of the University of Nairobi with my approval as the University supervisor.

Sign: [illegible] Date: 22/06/2011

Project supervisor
School of Computing and Informatics
University of Nairobi

ABSTRACT

Users continue to be overwhelmed by massive amount of information spread across the network with which they are confronted on a daily basis. Most network operations continue to use client server technology which has effects on bandwidth. Due to this, mobile agents promise a bright future in distributed systems because of their ability to reduce network bandwidth. This project shows how mobile agent can be used in distributed environment to connect to a computer residing in a remote location allowing the user to search for files, delete, and copy files without compromising on bandwidth.

The main objective of this project is to implement a method that uses mobile agents in a networked environment. The implementation consists of a control server that disperses an agent to a remote computer allowing the user to manipulate data as if in a local computer. A comparison of the performance is made between this approach and the traditional client server model. The results show that significance performance is obtained using mobile agents.

Virtualization being a new technology, many companies are moving to this realm due to the benefits that comes with it. This project has been implemented on a virtual environment to show that mobile agents can be used in situations where operations are running on virtual platform in a distributed environment.

TABLE OF CONTENT

Acknowledgement.

My sincere gratitude to the Almighty God for good health I enjoyed during the time of doing this project. Secondly, I am indebted to my supervisor Andrew Mwaura (University of Nairobi) for his supervision and invaluable contribution he made without whom this project would not have been a success. Many thanks to members of the Panel that included Dr. Waiganjo and Mr. Ogutu for their advices and highlighting areas that needed improvements during oral presentations. Finally, I thank my family especially my Dad for the special role he played during the entire time of doing this project.

List of Tables.

List of Figures

List of Abbreviations

- ACL- Agent Communication Language
- VM-Virtual Machine
- IR- Information Retrieval
- MA-Multiagent
- RMI-Remote Method Invocation

CHAPTER 1:

INTRODUCTION

1.1 Background

Currently, distributed systems employ models in which processes are statically attached to hosts and communicate by asynchronous messages or synchronous remote procedure calls. Mobile agent technology extends this model by including mobile processes, i.e., processes which can autonomously migrate to new hosts. This basic idea results in numerous benefits including flexible, dynamic customization of the behavior of clients and servers and robust remote interaction over unreliable networks.

According to David Kotz and Robert S. Gray(1999),Rapidly evolving network and computer technology, coupled with the exponential growth of the services and information available on the Internet, will soon bring us to the point where hundreds of millions of people will have fast, pervasive access to a phenomenal amount of information, through desktop machines at work, school and home, through televisions, phones, pagers, and car dashboards, from anywhere and everywhere.

In his paper David Kotz and Robert S. Gray(1999) argues that bandwidth to many end users will remain limited by several technical factors. Many users will still connect via modem, or at best, ADSL over the old copper loop. Many other users will connect via low-bandwidth wireless networks. Most users can expect to see no more than 128 Kbps to 1 Mbps available at their desktop or palmtop, although some asymmetric cable modems may reach 10 Mbps (for downloads) Corey Grice(1999), Amitava Dutta-Roy(1999).

Mobile agents will be an essential tool for allowing such access. Mobile agents are an effective choice for many reasons, and although not all applications will need mobile agents.
A traditional file search on a set of network computers requires mapping each drive to be searched on the local file system. This is a tedious task. During the search operation, the results are frequently returned to the requestor on a continuous basis consuming valuable network resources. The wider context of this research project is to explore how agents can be used in searching of files to eliminate problems associated with traditional methods. The likely users of this project include private networks, individuals, corporations and governments.

1.2 Search Service

The WSS organizes the extracted features of a collection of documents. The Windows Search Protocol enables a client to communicate with a server that is hosting a WSS, both to issue queries and to enable an administrator to manage the indexing server. When processing files, WSS analyzes a set of documents, extracts useful information, and then organizes the extracted information so that properties of those documents can be efficiently returned in response to queries.

A collection of documents that can be queried comprises a catalog, which is the highest-level unit of organization in Windows Search. A catalog represents a set of indexed documents that can be queried. A catalog consists of a properties table with the text or value and corresponding location (locale) stored in columns of the table.

The following conditions cause the service to throttle back or pause indexing:

- High CPU usage by non-search-related processes.
- High system I/O rate including file reads and writes, page file and file cache I/O, and mapped file I/O.
- Low memory availability.
- Low battery life.
- Low disk space on the drive that stores the index.

With growing application deployment on virtualized hardware, hardware resources are increasingly shared across multiple virtual machines.

1.3 AIM

The aim of this project is to research on mobile agent mechanism and their applicability in a networked environment. It also aims to demonstrate how agent can be deployed in a remote machine to enable users in a networked environment to search for files, synchronize and traverse directories

1.4 PROBLEM STATEMENT

Searching and manipulating of files on a remote computer is a challenge facing users on every day of their operation. Users in a distributed environment by their nature applications are distributed across multiple computers continue to face challenges due to distribution of their files across network. This is because of one or more of the following reasons:

- The data used by the application are distributed
- The computation is distributed
- The users of the application are distributed

A traditional file search on a set of network computers requires mapping each drive to be searched on the local file system. This is a tedious task. During the search operation, the results are frequently returned to the requestor on a continuous basis consuming valuable network resources. Users in a network are faced with the following issues;

- network bandwidth
- Disconnected operation
- Low-latency interaction.

1.5 Motivation

Mobile agent has continued to gain popularity in filtering information, automation of process and also in buying and selling of goods in the field of ecommerce. Being a new technology, there is so much attention from researchers across the globe on agent technology. Despite all this, very little has been done in the area of distributed systems and how agents can help users in a distributed environment can benefit from them.

The basic idea of this project is to research on agent technology and its application in file operation

1.6 Mobile agents are inevitable

David Kotz and Robert S. Gray (1999) argues that not because mobile code makes new applications possible, nor because it leads to dramatically better performance than (combinations of) traditional techniques, but rather because it provides a single, general framework in which distributed, information oriented applications can be implemented efficiently and easily, with the programming burden spread evenly across information, middleware, and client providers. In other words, mobile code gives providers the time and flexibility to provide their users with more useful applications, each with more useful features.

1.7 Outcome

The outcome of this research project is a project report, academic paper and a prototype that implements an agent based remote file locator in a virtual environment.

1.8 Assumptions

- That users in a networked environment posses certain rights to access a machine located somewhere in a remote location.
- Only trusted users are allowed to execute commands outlined.
- Those computers in the network will be registered. This will enable the local machine (acting as the server) to access the registered clients.

1.9 OBJECTIVES

- To conduct comprehensive literature review on mobile agents and evaluate their application in a distributed environment.
- Study Agent mobility and identify an appropriate way in which agents can be used in locating file stored in a remote location.
- To study code migrations and its application in agent environment.
- To develop agent based application that connects to a remote machine and allow user to locate file, synchronize files.

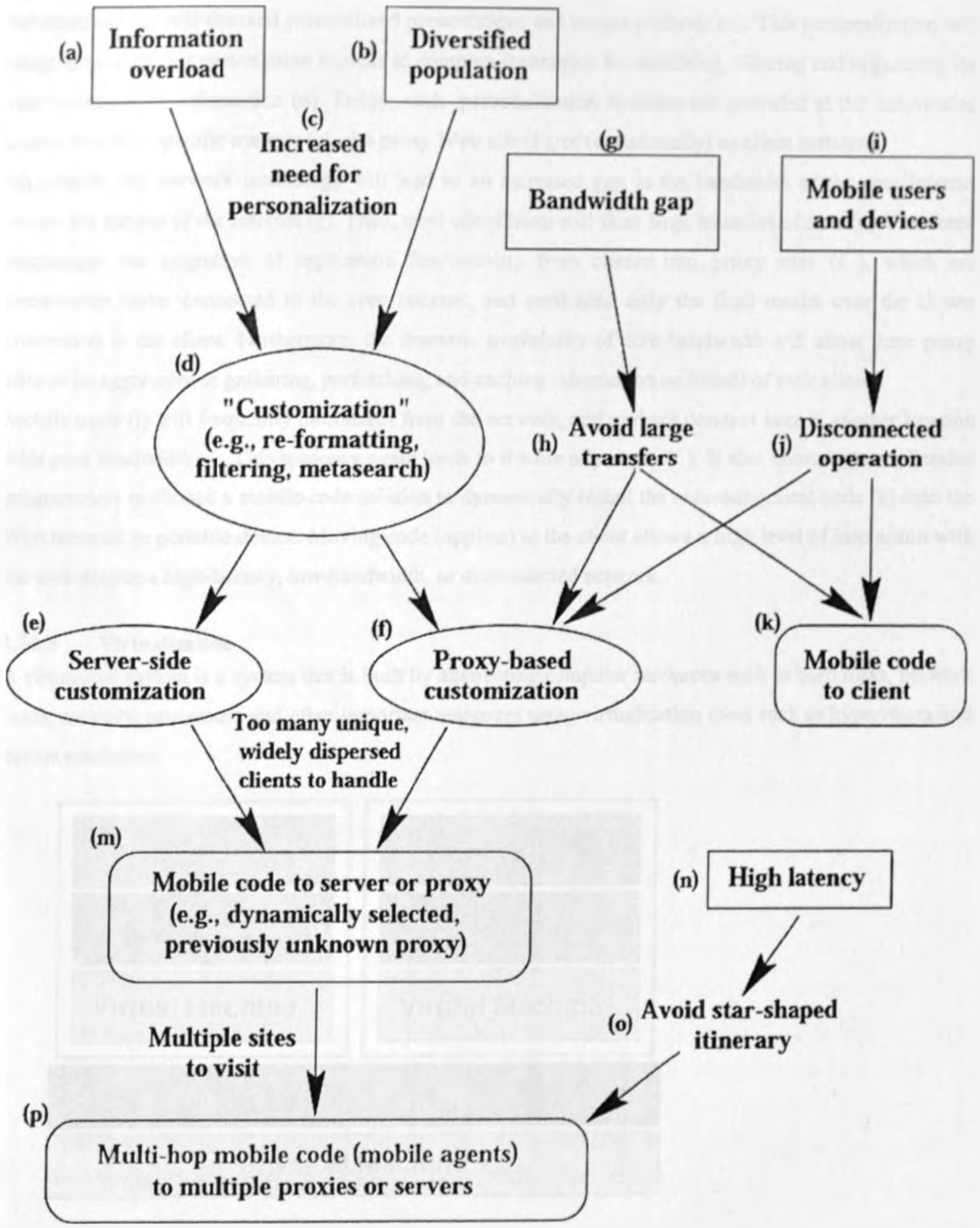
1.10 The Concepts of Agents and Mobile Agents

1.10.1 Agents

According to Michael Luck, Peter McBurney and Chris Preist (2003), Agents can be defined to be autonomous, problem-solving computational entities capable of effective operation in dynamic and open environments. Agents are often deployed in environments in which they interact, and maybe cooperate, with other agents (including both people and software) that have possibly conflicting aims. Such environments are known as multiagent systems. Agents can be distinguished from objects (in the sense of object oriented software) in that they are autonomous entities capable of exercising choice over their actions and interactions. Agents cannot, therefore, be directly invoked like objects. However, they may be constructed using object technology.

1.10.2 Mobile Agents

Mobile agents are programs that can move around on the network, while performing their duty, which may be a calculation, a database lookup or some other service.



(a) Information overload
(b) Diversified population
(c) Increased need for personalization
(g) Bandwidth gap
(i) Mobile users and devices
(d) "Customization" (e.g., re-formatting, filtering, metasearch)
(h) Avoid large transfers
(j) Disconnected operation
(e) Server-side customization
(f) Proxy-based customization
(k) Mobile code to client
Too many unique, widely dispersed clients to handle
(m) Mobile code to server or proxy (e.g., dynamically selected, previously unknown proxy)
(n) High latency
(o) Avoid star-shaped itinerary
Multiple sites to visit
(p) Multi-hop mobile code (mobile agents) to multiple proxies or servers

Figure 1: The trends leading to mobile agents

David Kotz and Robert S. Gray (1999) present the explanation of the above diagram where they say that both the amount of information available on the Internet (a), and the number and diversity of its users (b), are growing rapidly. This diverse population of users will not settle for a uniform interface to the information, but will demand personalized presentations and access methods (c) This personalization will range from different presentation formats to complex techniques for searching, filtering and organizing the vast quantities of information (d). Today, such personalization facilities are provided at the information source in a site- specific manner (e), at a proxy Web site (f), or (occasionally) as client software.
Meanwhile, the network technology will lead to an increased gap in the bandwidth of the core Internet versus the fringes of the Internet (g). Thus, most client hosts will shun large transfers of data (h). That trend encourages the migration of application functionality from clients into proxy sites (f), which are presumably better connected to the core Internet, and need send only the final results over the slower connection to the client. Furthermore, the dramatic availability of core bandwidth will allow these proxy sites to be aggressive in gathering, prefetching, and caching information on behalf of their clients.
Mobile users (i) will frequently disconnect from the network, and perhaps connect later at another location with poor bandwidth (j). This tendency again leads to the use of proxies (f). It also encourages application programmers to choose a mobile-code solution to dynamically install the necessary client code (k) onto the Web terminal or portable device. Moving code (applets) to the client allows a high level of interaction with the user despite a high-latency, low-bandwidth, or disconnected network.

1.10.3 Virtualization

A virtualized system is a system that is built by abstracting computer resources such as hard disks, network cards, memory, processors and other important resources using virtualization tools such as hypervisors and device emulators.

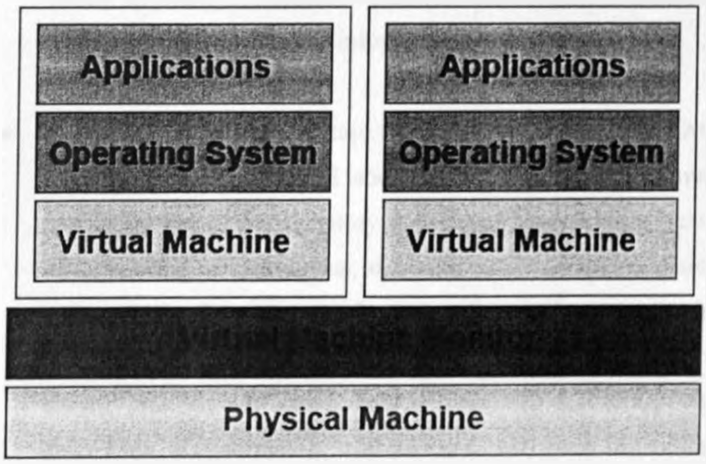


Figure 2: A virtual machine
A virtual machine monitor provides a virtual machine abstraction in which standard operating systems and applications may run. Each virtual machine is fully isolated from the rest of the virtual machines.

1.10.4 Agent technologies

Agent-based approaches have been a source of technologies to a number of research areas, both theoretical and applied. These include distributed planning and decision-making, automated auction mechanisms, communication languages, coordination mechanisms, matchmaking architectures and algorithms, ontologies and information agents, negotiation, and learning mechanisms. Moreover, agent technologies have drawn from, and contributed to, a diverse range of academic disciplines, in the humanities, the sciences and the social sciences.

1.10.5 Technological Challenges

Arising from this picture of the future of agent research, there are a number of broad technological challenges for research and development over the next decade.

- Increase quality of agent software to industrial standard. One of the most fundamental obstacles to large-scale take-up of agent technology is the lack of mature software development methodologies for agent-based systems. Clearly, basic principles of software and knowledge engineering need to be applied to the development and deployment of multi-agent systems, but they also need to be augmented to suit the differing demands of this new paradigm.

- Provide effective agreed standards to allow open systems development. In addition to standard languages and interaction protocols, open agent societies will require the ability to collectively evolve languages and protocols specific to the application domain and to the agents involved. Some work has commenced on defining the minimum requirements for a group of agents with no prior experience of each other to evolve a sophisticated communications language, but this work is still in its infancy. Research in this area will draw on linguistics, social anthropology, biology, the philosophy of language and information theory.

- Provide semantic infrastructure for open agent communities. At present, information agents exist in academic and commercial laboratories, but are not widely available in real world applications. The move out of the laboratory is likely to happen in the next ten years, but requires: a greater understanding of how agents, databases and information systems interact; investigation of the real-world implications of information agents (for example, including the economic effects of shopbots); and development of benchmarks for system performance and efficiency. In order to support this, further needs include: new web standards that enable structural and semantic description of information; and services that make use of these semantic representations for information access at a higher level. The creation of common ontologies, thesauri or knowledge bases play a central role here, and merits further work on the formal descriptions of information and, potentially, a reference

1.10.6 SIGNIFICANCE OF THE STUDY

This project aims at coming up with a way of locating files on a remote machine by use of agent mobility to enable the enterprises eliminate issues related with traditional search method. It aims to benefit the organizations in the following ways.

a) ***Asynchronous task execution***: While the agent acts on behalf of the client on a remote computer, the client may perform other tasks.

b) ***Reduced communication bandwidth***: network bandwidth is an essential resource in any enterprise. Bandwidth has limitations and therefore need to be utilized efficiently.

c) ***Higher degree of robustness***: A dispatched agent will deal with potential problems such as unavailable servers (e.g., go to alternate sources or retry at some later time). Although mobility introduces new failure cases, in general fault tolerance is promoted because a mobile agent has the potential to react dynamically to adverse situations.

CHAPTER 2:

LITERATURE REVIEW

Helena Nunes and Sofiane Labidi, (2002) define Information Retrieval (IR) as the computer science area that deals with the representation, storage, organization, and access to information items. The main objective of IR systems is to satisfy the user information needs in the best possible way Baeza-Yates, R. and Ribeiro-Neto, B. (eds.),(1999). Helena Nunes and Sofiane Labidi, (2002) designed the MathNet Searching Tool based on mobile intelligent agents (specifically aglets) that access the resources base in the system and interact with the Learner Modeling Agent and with the Strategist Agent to give to the learner the right information, in the right time, and using the correct means.

According to Parineeth M Reddy, (2002) A software agent is an intelligent program that acts as a user's personal assistant. Software agents endowed with the property of mobility are called mobile agents. Mobile agents perform a user's task by migrating and executing on several hosts connected to the network. In his article Parineeth M Reddy,(2002) Software agents have the following properties, which distinguish them from other programs:

In his publication Parineeth M Reddy,(2002) argues that software agents employ techniques from the field of artificial intelligence, which empower them with a fair degree of intelligence and common sense. For example, the travel agent program should realize that people generally do not prefer traveling by flights that depart or arrive at the airport late in the night and the agent should avoid booking tickets on such flights. The travel agent program should be smart enough to bargain and arrange the trip so that the overall expenditure for the trip is as low as possible without compromising on the user's preferences.

As Russell and Norvig (1995) stipulate, one of the most desirable properties of an agent is its rationality. It is said that an agent is rational if it always does the action that will cause the agent to be the most successful. The rationality of an agent depends on: The performance measure that defines what is a good action and what is a bad action, the agent's knowledge about the environment and the agent's available actions.

Parineeth M Reddy, (2002) add that Software agents should provide a user friendly interface so that the user can easily interact with the agent. Agents are social entities and often communicate and collaborate with one another in order to complete their tasks. For example, the travel agent program of one user must be able to communicate with other travel agents to find out about hotels which customers disliked and avoid such hotels. He proposes an agent-based platform which allows users to receive and exchange multimedia data from distributed sources, e.g. digital libraries, image databases and video databases.

In his publication, Kunal Shah (2003), illustrates that mobility is an orthogonal property of agents. That is, all agents are not necessarily required to be mobile. An agent can remain stationary and communicate with the surroundings by conventional means like remote procedure calls (RPC) and remote object invocation (RMI) *etc*. The agents that do not or cannot move are called *stationary agents*. On the other side, a *mobile agent* is not bound to the system where it begins execution. The mobile agent is free to travel among the hosts in the network. Once created in one execution environment, it can transport its state and code with it to another execution environment in the network, where it resumes execution.

The use of mobile agents in a distributed has been supported by U.P.Kulkarni, Prof A R Yardi (2005), where he says that Distributed system employs models in which processes are statically attached to hosts and communicate by asynchronous messages or synchronous remote procedure calls. Mobile agent technology extends this model by including mobile processes, which autonomously migrate to new hosts.

2.1 Working of Mobile Agents

Parineeth M Reddy, (2002) states that mobile agent consists of the program code and the program execution state (the current values of variables, next instruction to be executed, etc.). Initially a mobile agent resides on a computer called the home machine. The agent is then dispatched to execute on a remote computer called a mobile agent host (amobile agent host is also called mobile agent platform or mobile agent server). When a mobile agent is dispatched the entire code of the mobile agent and the execution state of the mobile agents transferred to the host.

The host provides a suitable execution environment for the mobile agent to execute. The mobile agent uses resources (CPU, memory, etc.) of the host to perform its task. After completing its task on the host, the mobile agent migrates to another computer. Since the state information is also transferred to the host, mobile agents can resume the execution of the code from where they left off in the previous host instead of having to restart execution from the beginning. This continues until the mobile agent returns to its home machine after completing execution on the last machine in its itinerary.

2.2 Agent communication

An ACL provides agents with a means of exchanging information and knowledge; Michael R. Genesereth has gone as far to equate agency with the ability of a system to exchange knowledge using an ACLM.R. Genesereth and S.P. Katchpel, (1994).

Communication plays an important key rule in agent technology. Agents communicate with users, resources, and with each other to cooperate or negotiate. A common language is required to support full agent functionalities. The most famous common languages are KQML and FIPA ACL. The former was developed by US Government's Agency ARPA, the latter from the Foundation for Intelligent Physical

Agents. Both only deal with agent-to-agent communication. Much work has to be done for human computer interface issues if agents are supposed to act on behalf of a human. The agent model employs two different architectures: Purely reactive agents that operate in a simple stimulus-response fashion and that do not embed concepts of plans or beliefs; the Jade platform Jade: Java agent development framework (jade.tilab.com.) uses this kind of agents; Deliberate agents that implement the BDI model (beliefs, desires, intentions) and that embed the concept of intelligent agent; for instance the JACK (JACK Autonomous Software. www.aosgrp.com.) and JADEX (Jadex Agent System. vsis-www.informatik.uni-hamburg.de/projects/jadex.) platforms implement those kind of agents.

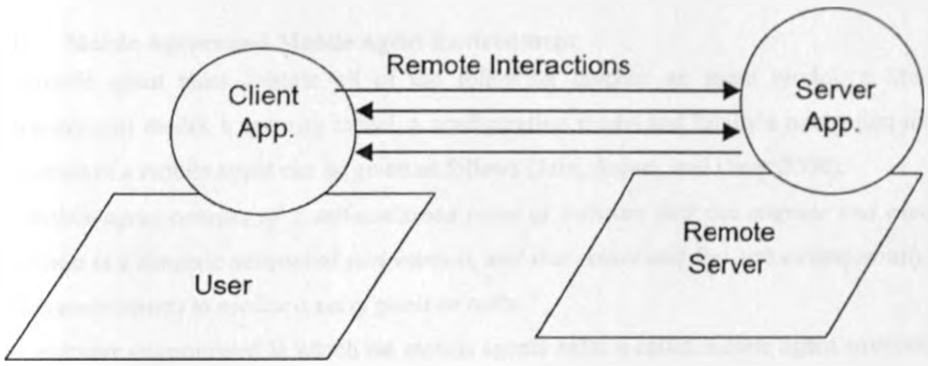


Figure 3 Communications using client-server paradigm.

In Telescript agents communicate by holding meetings. An agent can request a meeting with another agent at the same place, which is the same execution environment. The Telescript system passes the meeting request to the relevant agent. Every Telescript agent must implement the operation meeting. This is called when an agent receives an invitation to hold a meeting. The implementation of the meeting method contains the agents negotiating strategies, which may include rejecting holding a meeting under certain conditions or with certain types of agents.

Agent Tcl provides extensions to the Tcl language for agent communication. These extensions allow agents to communicate through either asynchronous message passing, or through remote procedure calls.

Java has no built in support for agent communication. In Aglets, each Java agent has a proxy object. Communication from one agent to another happens through the proxy. This is to protect the agent objects from being directly modified. The proxy object provides a set of methods for communicating to the represented object. These include requests for aglets to take actions, such as migration, cloning, destroying and suspending. There are also two methods for sending synchronous and asynchronous messages to the aglets.

2.3 Mobile agents in a distributed environment

J.Waldo, G.Wyant, A.Wollrath, and S. Kendall (1994) writes that it is argued that cases exist, especially in large-scale computer networks, where interaction among participants residing on remote hosts is rather different in terms of latency and access to application services. Hiding such differences might end up in unexpected performance problems. Therefore the location of system components such as agents in wide-area networks (e.g. the Internet) has to be actively taken into consideration. In the context of mobile agents this includes the capability to reconfigure dynamically, at run-time, the binding between software components (e.g. agents) of the application and their physical location within a computer network, as discussed in World Wide Web Consortium (W3C), (2000)

2.4 Mobile Agents and Mobile Agent Environment

A mobile agent must contain all of the following models: an agent model, a life-cycle model, a computational model, a security model, a configuration model and finally a navigation model. A working definition of a mobile agent can be given as follows (Jain, Anjum, and Umar 2000):

"A mobile agent consists of a self-contained piece of software that can migrate and execute on different machines in a dynamic networked environment, and that senses and (re) acts autonomously and proactively in this environment to realize a set of goals or tasks."

The software environment in which the mobile agents exist is called mobile agent environment. Following is the definition of mobile agent environment (Mahmoud 2001):

"A mobile agent environment is a software system distributed over a network of heterogeneous computers. Its primary task is to provide an environment in which mobile agents can execute. It implements the majority of the models possessed by a mobile agent."

The above definitions state the essence of a mobile agent and the environment in which it exists. The mobile agent environment is built on top of a host system. Mobile agents travel between mobile agent environments. They can communicate with each other either locally or remotely. Finally, a communication can also take place between a mobile agent and a host service.

2.5 The life cycle of a mobile agent

According to Kunal Shah (2003), Agent undergoes the following stages in its lifetime

1. The mobile agent is created in the Home Machine.
2. The mobile agent is dispatched to the Host Machine A for execution.
3. The agent executes on Host Machine A.

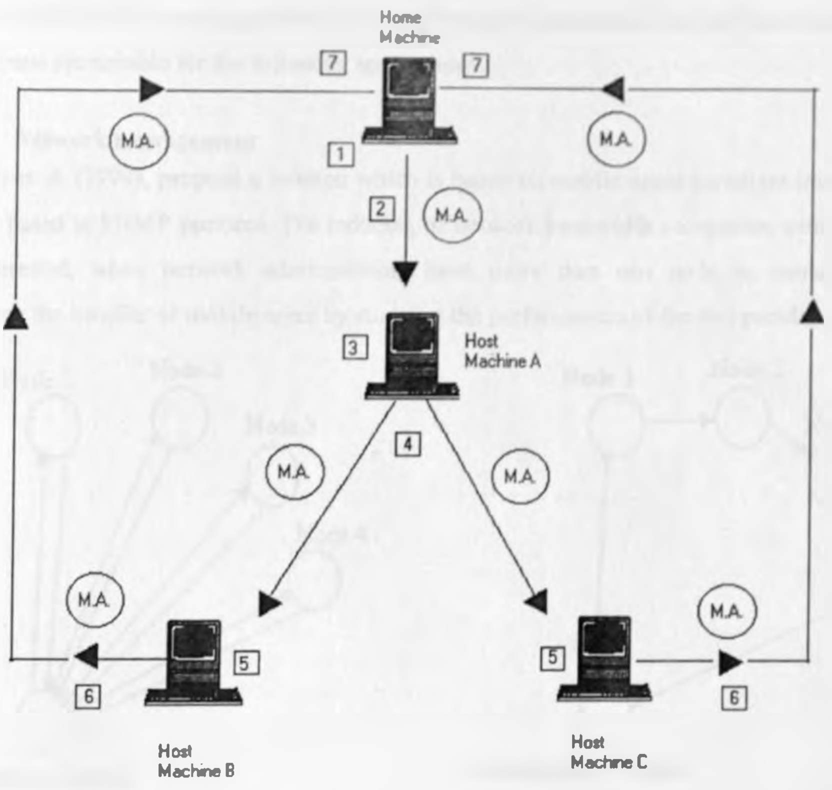


Figure 4 The life cycle of a mobile agent

4. After execution the agent is *cloned* to create two copies.

One copy is dispatched to Host Machine B and the other is dispatched to Host Machine C.

5. The cloned copies execute on their respective hosts.
6. After execution, Host Machine B and C send the mobile agent received by them back to the Home Machine.
7. The Home Machine *retracts* the agents and the data brought by the agents is analyzed. The agents are then *disposed*.

2.6 Applications of Mobile Agents

Abdelkader Outtagarts(2009)outlines that mobile agent applications in different domains such as network management, electronic commerce, energy efficiency and metering; Wireless Multimedia Sensors, grid computing and grid services, distributed data mining, multimedia, human tracking, security, affective computing, climate environment and weather, e-learning, location, recommendation and semantic web services.

Although no universally used application (normally called killer application) has been developed for them, mobile agents are suitable for the following applications.

2.6.1 Network management

Outtagarts et al. (1999), propose a solution which is based on mobile agent paradigm instead client-server paradigm based in SNMP protocol. The reducing of network bandwidth occupation with mobile agents is more interested, when network administrators have more than one node to manage. The authors demonstrate the benefits of mobile agent by studying the performances of the two paradigms.

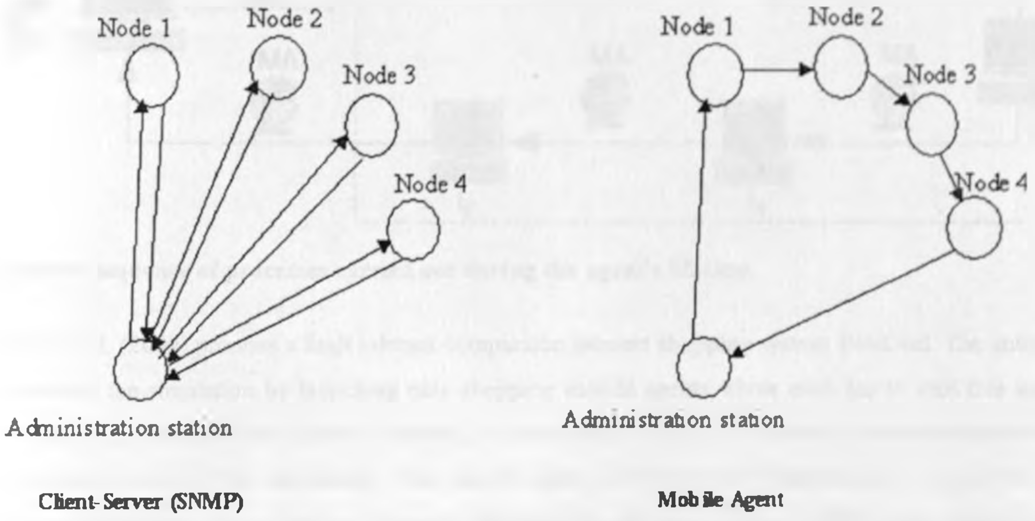


Figure 5 client-server and mobile agent application in network

Management. Outtagarts et al. (1999),Manvi et al(2009), use mobile agent to find multiple QoS paths and select a best path among them to increase call success ratio and network bandwidth. One of the most significant example of mobile agents applications is the management of commercial telecommunication networks. Currently if an optical cable in the WAN is accidentally cut, the time required to locate the problem may extend ridiculous because of the slow response of the network itself Thanh (2001). Alarms will be sent to all systems attached to this network and data protection can be lost.

2.6.2 Electronic commerce

Electronic commerce is another good fit for mobile agent technology. A mobile agent could do your shopping for you, including making orders and potentially even paying. Al-Jaljouli et al (2009) have implemented mobile agent in e-commerce to search and to filter information of interest from electronic markets. They describe also robust security techniques that ensure a sound security of information gathered throughout agent's itinerary against various security attacks, as well as truncation attacks.

Mobile agents can travel to different trading sites and help to locate the most appropriate deal, negotiate the deal and even finalize business transactions on behalf of their owners. A mobile agent can be programmed to bid in an online auction on behalf of the user. The user himself need not be online during the auction

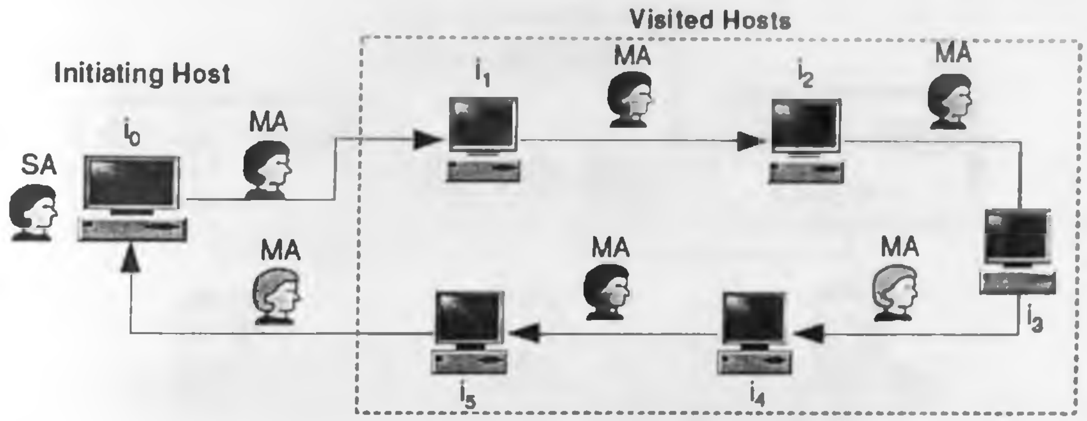


Figure 6 sequence of processes carried out during the agent's lifetime.

Nipur et al. (2009) propose a fault tolerant comparison internet shopping system BestDeal. The author has conducted the simulation by launching nine shopping mobile agents where each has to visit five supplier sites to get the best deal for different products. Performance is measured in terms of execution steps as well as execution time of the simulation. The mobile agent survives even if failure rate is more than 80% however for higher failure rate performance degraded significantly. Li et al. (2009) have studied mobile agent oriented M-commerce platform. The design and implementation of a mobile agent platform for M-commerce applications is discussed in this paper. According to the authors, the advantage of adopting mobile agents for M-commerce is to scale up to large, dynamic world market places distributed over the Internet and to ease the access and participation of mobile users.

2.6.3 Parallel Computing

Solving a complex problem on a single computer takes a lot of time. To overcome this, mobile agents can be written to solve the problem. These agents migrate to computers on the network, which have the required resources and use them to solve the problem in parallel thereby reducing
the time required to solve the problem(http://www.itswtech.org).

K. Belkhelladi, P. Chauvet and A. Schaal (2009) has developed a framework that uses mobile agents launched into different hosts on available networked PCs and cooperating among them to solve large combinatorial problems efficiently. The execution environment framework is based on the JADE technology. In addition, we define a new information exchange strategy based on a dynamic migration window method and a selective migration model. A parameters adaptation model is also proposed. This model is used to adjust different parameters/operators of the genetic algorithm executed by each mobile agent. The proposed framework has been experimented on an extended set of Earliness and Tardiness Production Scheduling and Planning Problem (ETPSP).

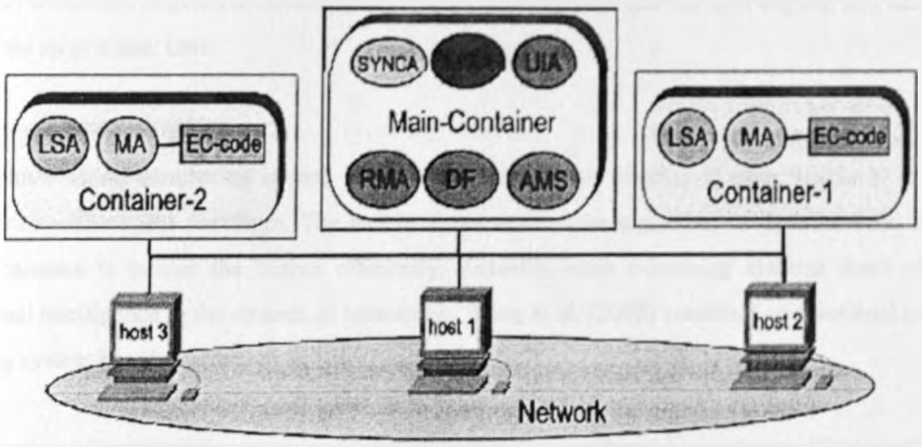


Figure 7 Framework architecture. Belkhelladi, P. Chauvet and A. Schaal(2009)

2.6.4 Data Collection

Consider a case wherein, data from many clients has to be processed. In the traditional client-server model, all the clients have to send their data to the server for processing resulting in high network traffic. Instead mobile agents can be sent to the individual clients to process data and send back results to the server, thereby reducing the network load.

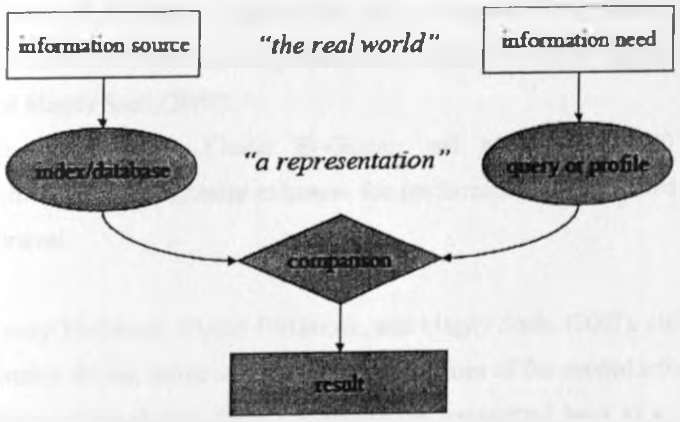


Figure 8 Model of Search

[adapted from Belkin & Croft, 1992]

2.6.5 Mobile Computing

Wireless Internet access is likely to stay slow and expensive. Power consumption of wireless devices and high connection fee deter users from staying online while some complicated query is handled on behalf of the user. Users can dispatch a mobile agent, which embodies their queries, and log off, and the results can be picked up at a later time.

2.6.6 Human Tracking

To enhance video monitoring system in the automatic human tracking system, Kakiuchi et al. (2009) introduce mobile agent paradigm. The mobile agent utilizes the algorithm of determination of neighbor video cameras to pursue the human efficiently. Actually, since e-learning systems don't consider the emotional intelligence in the context of instruction, Wang et al. (2008) construct an emotional intelligent e-learning system based on mobile.

The emotion of a student is recognized by facial expression captured by a camera. The get the student's learning psychology by analyzing the facial expression of students using two-dimension model to describe a student's emotion.

2.7 MOBILE AGENTS IN A DISTRIBUTED ENVIRONMENT

The development of distributed systems has been affected by the need to accommodate an increasing degree of flexibility, adaptability, and autonomy. The Mobile Agent technology is emerging as an alternative to build a smart generation of highly distributed systems ,Yousry El-Gamal, Khalid El-Gazzar, and Magdy Saeb,(2007).

Yousry El-Gamal, Khalid El-Gazzar, and Magdy Saeb,(2007),argue that exploiting agent-based technologies significantly enhances the performance of distributed systems in the domain of information retrieval.

Yousry El-Gamal, Khalid El-Gazzar, and Magdy Saeb, (2007), state that a mobile agent can abandon the portable device, move onto the network locations of the needed information resource and perform a locally custom retrieval task. Only the results are transmitted back to a portable device. Moreover, the mobile agent can carry on a task while the connection to the portable device is temporally lost and then continue once the link returns to send the found result.

Yousry El-Gamal, Khalid El-Gazzar, and Magdy Saeb, (2007), propose an analytical model that describes the network load and the response time in order to compare the performance of both the mobile agents and the Remote Method Invocation (RMI). RMI is an object equivalent of the classical client-server approach. In their research, they only concerned with the parameters that can be useful in the comparative evaluation.

For example, the number of requests that arrives to the server can affect the total processing time of the request.
The task is terminated once the data item is found. We assume that the client begins to send a request Breq in bytes and the servers reply by Bres if the requested data are found otherwise, a reply BNF is returned. The same process is repeated on the next server until the required data are fetched. On the other hand, the mobile agent approach visits sequentially the set of servers until it obtains the desired information.
The mobile agent compresses the data which are found at the server before transmitting it back to the client by a compression ratio s, where 0 = s = 1.

They assume that the mobile agent consists of code BC, data state BD, where BD is the sum of the bytes of the result, and BS is the execution state. The probability of finding data at server i is given by pi, where 0 = pi = 1. The migration process consists of marshalling data and state, transmitting the code, data and state to the destination, unmarshalling data and state then, resuming the agent execution. We assume that the time to marshalling and unmarshalling one byte is tm. The time to process the request at the server is tp. The time to transfer one byte from location L1 to location L2 over the direct link L1-L2 is BL 1 L 2 t - . That is, indirect routes are ignored. They emphasize that; each server will be visited only once. Therefore, there is no reusing of the same server again in one searching task. The load due to TCP header is ignored. There is no network queuing time. For sake of simplicity, they assume that the returned result has a constant size. additionally they assume a constant overhead scheduling time (ts) in the case of mobile agents. That is, the agent action is considered as a heavy task. Finally, they neglect the authentication overhead.

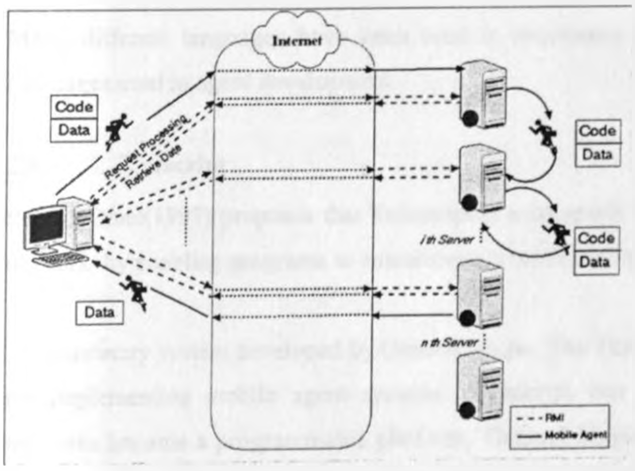


Figure 9 Illustration of a graphical scenario of a client seaching

Based on the above experiment, mobile agents performed better in a distributed environment that RMI in information retrieval.

2.8 Code Mobility

Iris Reinhartz-Berger, Dov Dori, and Shmuel Katz (2002), each operation that involves code mobility can be divided into three steps: determining the code operation targets, transferring the code, and integrating the code into the target system. In static system architecture, the target determination step can be done at compilation time. If the system architecture is dynamic, such that it is determined at run time, then the operation targets should be computed immediately prior to transferring the code.

Following the target determination, the code is transferred by applying one of the design paradigms for code mobility, which extend the traditional client-server paradigm from data to code. Once transferred, the code can be integrated with the local target system by activating an instance of it, connecting it to existing data or code, or continuing its transfer over the network.

Li Jingyue (2002) Mobile code is an important programming paradigm and opens up new possibilities for structuring distributed software systems in an open and dynamically changing environment. It can improve speed, flexibility, structure, or ability to handle disconnections and it is particularly well suited if adaptability and flexibility are among the main application requirements. It has applications in many areas, such as mobile computing, active networks, network management, resource discovery, software dissemination and configuration, electronic commerce, and information harvesting

2.9 Mobile Agent Development Environments

Many different languages have been used to implement mobile agents. The following are the main languages used in agent development.

2.9.1 Telescript

Peter DOmel(1997) proposes that Teleseript is a computer language which was designed to program the network by enabling programs to autonomously move themselves from host to host in order to do their job.

A proprietary system developed by General Magic. The Telescript language has been specifically designed for implementing mobile agent systems. Telescript was designed with the vision for the computer networks become a programmable platform. General Magic's ambition was for Telescript to become for communications what Postscript is for printing. Contrary to the name, Telescript is not a scripting language. It is a complete object oriented language. Telescript supports objects, classes and inheritance. The object oriented model and the syntax is in many way similar to that of C++. Telescript has a library of built-in classes for writing mobile agents. There are special classes for *agents* and *locations*. Agents are a base class for mobile agents. Locations are objects that represent sites. The Telescript language has a set of built-in commands for agent migration and inter agent communication. The Telescript system includes notions of which authority the agent is representing. Telescript programs are compiled into a portable intermediate representation, called *low Telescript*, analogous to Java byte code. Telescript programs can run on any computer with a Telescript execution engine. The Telescript execution engine was designed to be able to run on even small communication devices. The Telescript language has had a great influence on the development of mobile agents, and mobile agent languages. It was General Magic who first coined the term *mobile agent*.

2.9.2 Java

The Java language M. Campione and K. Walrath, (1998)] and its virtual machine seem to be very appropriate to implement an agent system. Java is platform independent, allows for serialization and persistency and comes with security built into the virtual machine. These are just some of the features needed to create an agent system.

Java is a general purpose language. Despite its relatively young age, it is already establishing itself as the de facto standard for developing internet and intranet applications. Java is an object oriented language. It uses the classes object oriented model. Its syntax is similar to that of C and C++. While Java was not specifically designed for writing mobile agents, it has most of the necessary capabilities for mobile agent programming. Java is multi-threaded. Java programs are compiled to Java byte codes, binary instructions for the Java Virtual Machine, Steven Versteeg (1997).

Java programs are able to run on any platform with a Java Virtual Machine interpreter. This makes Java programs highly portable. The Java libraries have good support for communication procedures. Java has been used as the basis for many implementations of mobile agent systems. Nearly all of the systems make use of Java 1.1's RMI (Remote Method Invocation). Some systems of note include:

- IBM's **Aglets** - under development by IBM Research Centre, Japan. An *aglet* is a mobile agent. All aglets are derived from an abstract class called Aglet. Aglets uses an event driven approach to mobile agents, that is analogous to the Java library Applet class. J. Kiniry and D. Zimmerman, (1997) Each aglet implements a set of event handler methods that define the aglets behaviour. Some of these methods are:

    - OnCreation() - called when a new aglet is created.
    - OnDispatch() - called when an aglet receives a request to migrate.
    - OnReverting() - called when the aglet receives a request from its owner to come home.
    - OnArrival() - called after an aglet is dispatched
- General Magic's **Odyssey** - A mobile agent system under development by General Magic, that attempts to achieve the functionality of Telescript, using Java.

- ObjectSpace's **Voyager** - The Voyager system's model of mobile computing is very similar to that of Obliq. The system provides a mechanism for converting objects into a distributed objects. This allows objects at remote sites to be semantically treated in the same way as objects at the local site. Objects can be easily copied between remote sites.

2.9.3 Obliq

Obliq is an experimental language under development by Digital Equipment Corporation's Systems Research Center. Obliq is a lexically scoped, object-based, interpreted language that supports distributed computation. The language supports objects, but not classes. It uses the prototype-based model of object-oriented programming. New objects can be created directly, or cloned from other objects. Obliq uses runtime type checking. Obliq has built-in procedures for importing and exporting procedures and objects between machines. Obliq adheres to lexical scoping in a distributed context. When procedures and objects are dispatched to a remote site for execution, any references they contain point to the same objects as on the machine from which they were dispatched, A. H. Borning (1986).

The Obliq distributed semantics is based on the notions of *sites*, *locations*, *values* and *threads*. A site is a computer on the network. A location is a memory address on a site that stores a value. A value can be of a basic type or an object. Threads are virtual sequential instruction processors. Threads may be executed concurrently on the same site or at different sites. Values may be transmitted over the network. When an object is transmitted, basic values are copied exactly. Locations that the object contains are copied, such that they point to the same address on the same site, at the destination site as they did at the original site.

Obliq's semantics of network computing is fundamentally different to the other languages considered. Where as other languages see each computer as independent worlds that can communicate with each other through the network, Obliq treats the network as a single computer with sites as components.

2.9.4 Agent Tcl

Agent Tcl is a mobile agent system being developed by Dartmouth College. The Agent Tcl language is an extension of the Tool Command Language (Tcl), the language originally developed by Dr. John Ousterhout. The Agent Tcl extensions add commands for agent migration and message passing. The extra commands give Agent Tcl scripts similar mobility capabilities to Telescript. Agent Tcl uses a modified Safe Tcl interpreter to execute scripts.

2.9.5 Perl 5

Penguin is a Perl 5 module with functions enabling the sending of Perl scripts to a remote machine for execution and for receiving perl scripts from remote machines for execution. The scripts are digitally signed to allow authentication and are executed in a secure environment. Mobile agents written in Perl are restricted in that they must always restart execution at the same point. There is also no support for agents saving their state on migration. A new Agent Module v3.0 is being created to give Perl 5 more sophisticated mobile agent capabilities. The extra features include giving agents the ability to save their state on migration.

2.9.6 Python

Python is an object-oriented scripting language. The Corporation for National Research Institution, uses Python as a language for implementing Knowbot programs.

This is by no means a complete list of the languages being used for mobile agents. For a more complete list, the reader is referred to Kiniry and Zimmerman .

The languages that will be mainly considered in the following discussions are Telescript, Java, Agent Tcl and Obliq. Collectively, these languages represent most of the approaches presently taken to languages for mobile agents. Aglets will be most referred to of the Java libraries. The reason for this is the techniques associated with the other two Java libraries mentioned are represented by Telescript and Obliq.

2.10 CHARACTERISTICS OF MOBILE AGENT LANGUAGE

2.10.1 Migration

Steven Versteeg Supervisor: Leon Sterling (1996) proposes the agent language must be able to support an agent migrating. Ideally, it should be possible to suspend an agent's execution at any point, save the state, including the heap, the stack and even the registers, move the agent to another computer, and restart execution, with the agents execution state exactly restored.

Telescript has built-in support for agent migration. Agents may move to any location with the go statement. Upon the execution of this command, the agent is transported to the target site, where it continues execution from the line after the go statement. All the agents properties and the program execution state, including those of local variables in methods and the program counter, are restored exactly. The agent migration is process is handled completely by the Telescript operating system. The programmer does not need to worry about saving the relevant state information just before migration.

Agent Tcl uses a similar migration model to that of Telescript. The built-in statement for agent migration is called agent_jump. As with the Telescript go, when this statement is issued the execution environment handles the transportation of the agent, and restores the agent execution state. Since the Tcl language provides absolutely no support for capturing program state, this is an Agent Tcl extension of the language.

Java was not specifically designed for implementing mobile agents so it does not have in built-in support for migration. Saving the program state in Java is much more difficult. Java's security architecture makes it impossible to directly save the virtual machine execution state. However Java 1.1 supports class serialization. Serialization allows an entire class instance to be written to file, including the object's methods, attributes and their values. Serialization will not save the program stack, that is, the values of local variables in methods. The Java virtual machine does not allow the explicit referencing of the stack, for security reasons. Workarounds have been developed for saving the program stack state. In Aglets, each aglet implements a method called onDispatch(). This method is called when an aglet receives a request to migrate. The request may have come from the aglet itself or from another process. In this method, the programmer must define a procedure for placing everything an aglet needs to restore its state on the heap. The aglet is then serialized and transported to its destination.

There are advantages to Telescript and Agent Tcl's built-in support for agent migration. In Telescript it is possible to migrate from any point in the program, including in the middle of method calls. In Java the agent program must be structured so that everything needed to restore execution state is stored in the heap, before migration. It is left to the programmer to make sure that all variables are correctly saved. In Telescript and Agent Tcl, the implementation of agent migration is completely hidden from programmer. This is a source of error that Telescript programmers do not need to worry about.

Obliq takes a different view of agent migration. In Obliq, an agent can be written as a procedure that takes a state object as an argument. A site can make its execution engine available for threads at other sites to use. A procedure can be executed at a remote site, by passing the name of the procedure as a parameter to the execution engine. The following code fragment shows how an agent can be sent to another site for execution.

```
let state = { ... }; (define agent state)
let agent = proc(state, arg) ... end; (define agent procedure)

(get a handle to remote site execution engine)
let remoteSite = net_import("RemoteServer", Namer);

(Execute the agent at the remote site.)
remoteSite(proc (arg) agent(copy(state), arg) end)
```

2.10.2 Interface to server resources

The fundamental purpose of mobile agents is to get the program closer the source of the information. The agent implementation language must provide an easy way to access the resources on the host machine.

In Telescript, local resources are treated as another agent. There is an agent present at the server to represent the local resources. This model provides an elegant and consistent interface to local resources at different computers, but it requires writing a Telescript wrapper.

Obliq has categories different types of services provided by a site. A program may request a list of the services provided by a site in a particular category.

Agent Tcl and Aglets use a similar method to interacting with local resources to Telescript. In Aglets, an aglet is associated with an AgletContext object. This object describes the environment that the aglet is in. Through the aglet context object, an aglet is able to find out what other aglets are also in its current environment. Like in Telescript, a stationery aglet is used to represent the local computer's services.

2.10.3 Security

Security is a critical part of mobile agent systems. Karjoth, Lange and Oshima identify three security issues specific to mobile agent systems. These are: Protecting the host from the mobile agent, Protecting the mobile agent from other mobile agents, and Protecting the mobile agent from the host. Researchers have so far only found solutions to the first two issues.

Two major techniques are used to protect the host computer:

- Executing agents in an isolated environment. Agents cannot directly access any parts of the host system outside their execution environment. The agent system may grant some agents special privileges to access resources outside of their execution environment.
- Authenticating the source of mobile agents, and granting execution privileges to agents on the basis of how trusted their source is. Some agents may be denied execution altogether.

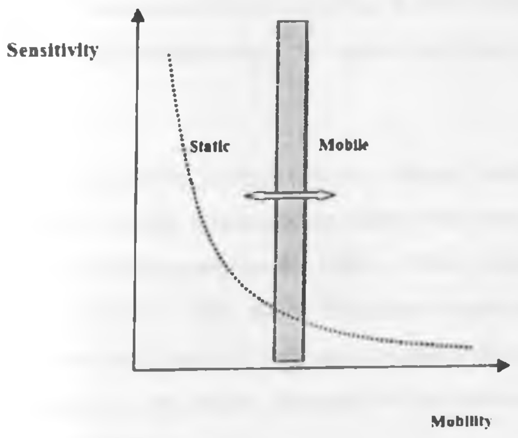


Figure 10 Degree of Mobility vs. Sensitivity of Agent

The level of security required for the application and the sensitivity of the mobile agent's code and data directly influences the degree of mobility of a mobile agent. As shown in Figure 2, as the sensitivity of the agent's task increases, the designer of the agent may decrease the degree of mobility of the agent. The shaded vertical bar represents a cut-off point for the designer to decide which agents will be static and which agents will be mobile. This decision will be made based on the available security mechanisms, performance requirements, sensitivity of the agent's code and data, maximum acceptable risk, and the level of functionality required. Java, Agent Tcl and Telescript use both of these mechanisms in their security models.

Java programs each run in their own environments. There are security mechanisms built into the Java Virtual Machine instruction set to prevent programs from accessing outside of their environment. These are:

- Type-safe reference casting.
- Structured memory access.
- Automatic garbage collection.
- Array bound checking.
- Checking references for null.

The effects of these mechanisms is that Java programs run in a sandbox. That is they are limited to the environment allocated to them by the Java Virtual Machine, and the Java byte code instruction set disallows them from directly accessing anything outside of this environment. Accesses outside of the sandbox can only be done by using some of the Java libraries, allowing disk access, network access, and printing, or by calling native methods. The Java Security Manager controls which programs are permitted access outside of the sandbox, and the nature of the outside access. For example, by default, applets are permitted to make network connections to their original source computer, but not to any other computers. The Security Manager may grant special privileges to all classes from the same author, or to just some classes.

Agent Tcl enforces runtime security checks with a technique similar to that used by the Safe-Tcl interpreter. Mobile agents are run within their own *safe* interpreters. In the safe interpreters commands that access outside resources are hidden. When an agent invokes a hidden command, it is redirected to the *master* interpreter. The master interpreter implements a security policy of what commands may be available to which agents. If the security policy allows the command for a particular agent, then the master interpreter calls the hidden command in the safe interpreter. The security policy is user-defined by the administrator of the server.

In Telescript all agents and places have an *authority* property. The authority is a class that defines the individual or organisation in the physical world that the agent or place represents. Agents and places must reveal their authority to another agent of place on request. They may not falsify or withhold their authority. The network of places is divided into *regions* under the same authority. When an agent tries to move from one region to another, the source region must prove the authority of the agent to the destination region.

The Telescript language also has *permits*. Authorities limit what agents can do by assigning them permits. Permits are used to limit what instructions agents execute, and to limit their resources to a budget. For example the agent's permit can limit its lifetime or the amount of computation it may do. Telescript was designed with electronic commerce in mind, so the same resource permits can be used to allocate agents an amount of money. If an agent ever tries to violate the conditions of its permit it is destroyed.

The Telescript language provides a very powerful and flexible framework for protecting the host computers from untrusted sources, but at the same time not getting in the way of doing business with trusted sources.

The common way for the host to authenticate incoming mobile agents is through digital signing. Most Java mobile agent systems and Agent Tcl use this method. When an agent is transported, the message containing it is signed by the sender agent server. The receiver agent server authenticates the mobile agent message on arrival. If any part of the agent message was altered in transit, the digital signature is no longer valid. The sender agent server signs the agent rather than the original author because an agent includes the program plus the state. The state will change.

Obliq has a completely different mechanism of achieving security. Obliq relies on the lexical scoping of the semantics of the language, together with strong runtime checking. When a agent is given to a remote site for execution, because of lexical scoping these agents can only access data or resources that they can reference via free identifiers, or that are given in as procedure parameters. Lexical scoping dictates that the free identifiers refer to values that are available at the client site. Hence, the only way an agent can obtain access to a server's resources is by assigning variables to resources that the server exports to the client site. The values of these variables can then be passed as parameters to the agent. Hence, the agent is only able to access server resources that the server explicitly exports.

The following code fragement illustrates. Agent 1 uses a local resource. Agent 2 is able to use a remote resource by obtaining a binding to an exported remote resource, and passing this as a parameter to the agent L. Cardelli (1995).

```
let agent1 = proc(arg)
   resource = getResource();
   use(resource)
end;
```

```
let agent2 = proc(resource, arg)
   use(resource)
end;

(get a handle to remote site execution engine)
let remoteSite = net_import("RemoteServer", Namer);

(Execute the agent1 at the remote site -- local resource is used)
remoteSite(proc (arg) agent1(arg) end)

(Get resource that the remote site exports)
resource = getResource(remoteSite)

(Execute the agent2 at the remote site, remote resource is passed as parameter)
remoteSite(proc (arg) agent2(resource, arg) end)
```

2.10.4 Cross platform

In most cases it is desirable for a mobile agent to be able to migrate across a heterogeneous network. Certainly, for a mobile agent to be used on the Internet this is a requirement. For this to be possible, the agent must be written in a language that is supported on all its potential host computers. This is one of the reasons why nearly all mobile agent systems use interpreted languages. All the languages looked at are interpreted.

Telescript, Java and Agent Tcl agents are all interpreted at execution. Interpreters for these languages exist across different platforms. (Obliq interpreters are currently only available for UNIX.) Despite this Java has a number of advantages in this area. First, Java Virtual Machine interpreters already exist on many computers. Most major operating system vendors, including Microsoft, Sun, IBM, Novell and Apple have announced that they plan to include the Java Virtual Machine as part of the next releases of their respective operating systems. Mobile agents written in Java will not require a special purpose interpreter to run. The mobile agent interpreter can be expected to be already available on most machines. Agent Tcl requires a special purpose interpreter. Telescript programs require a Telescript execution engine, a closed standard commercial product. One cannot realistically expect the Telescript execution engine to become as widely spread as Java Virtual Machine interpreters. Second, a general problem with cross platform technology is that, despite the intentions, some parts of the implementation act differently on different platforms. While this is certainly a problem with Java now, one might optimistically expect these bugs to be fixed, simply because of the magnitude of the resources involved in Java research and development.

As a sign perhaps that General Magic accepts that Java has become the cross platform standard, it is attempting to implement a Java-based equivalent of its Telescript technology.

2.11 Properties of Java that make it a good language for mobile agent programming

2.11.1 Platform-independence.

Java is designed to operate in heterogeneous networks. To enable a Java application to execute anywhere on the network, the compiler generates architecture-neutral byte code, as opposed to non-portable native code. For this code to be executed on a given computer, the Java runtime system needs to be present. There are no platform-dependent aspects of the Java language. Primitive data types are rigorously specified and not dependent on the underlying processor or operating system. Even libraries are platform-independent parts of the system. For example, the window library provides a single interface for the GUI that is independent of the underlying operating system. It allows us to create a mobile agent without knowing the types of computers it is going to run on.

2.11.2 Secure execution.

Java is intended for use on the Internet and intranets. The demand for security has influenced the design in several ways. For example, Java has a pointer model that eliminates the possibility of overwriting memory and corrupting data. Java simply does not allow illegal type casting or any pointer arithmetic. Programs are no longer able to forge access to private data in objects that they do not have access to. This prevents most activities of viruses. Even if someone tampers with the byte code, the Java runtime system ensures that the code will not be able to violate the basic semantics of Java. The security architecture of Java makes it reasonably safe to host an untrusted agent, because it cannot tamper with the host or access private information.

2.11.3 Dynamic class loading.

This mechanism allows the virtual machine to load and define classes at runtime. It provides a protective name space for each agent, thus allowing agents to execute independently and safely from each other. The class-loading mechanism in extensible and enables classes to be loaded via the network.

2.11.4 Multithread programming.

Agents are by definition autonomous. That is, an agent executes independently of other agents residing within the same place. Allowing each agent to execute in its own lightweight process, also called a thread of execution, is a way of enabling agents to behave autonomously. Fortunately, Java not only allows multithread programming, but also supports a set of synchronization primitives that are built into the language. These primitives enable agent interaction.

2.11.5 Object serialization.

A key feature of mobile agents is that they can be serialized and deserialized. Java conveniently provides a built-in serialization mechanism that can represent the state of an object in a serialized form sufficiently detailed for the object to be reconstructed later. The serialized form of the object must be able to identify the Java class from which the object's state was saved, and to restore the state in a new instance
Objects often refer to other objects. Those other objects must be stored and retrieved at the same time, to maintain the object structure. When an object is stored, all the objects in the graph that are reachable from that object are stored as well.

2.11.6 Reflection.

Java code can discover information about the fields, methods, and constructors of loaded classes, and can use reflected fields, methods, and constructors to 5 operate on their underlying counterparts in objects, all within the security restrictions. Reflection accommodates the need for agents to be smart about themselves and other agents.

2.12 Virtualization

Robert Rose(2004), defines virtual machine concept allows the same computer to be shared as if it were several. IBM defined the virtual machine as a fully protected and isolated copy of the underlying physical machine's hardware R. J. Creasy,(1981).
Today's x86 computer hardware was designed to run a single operating system and a single application, leaving most machines vastly underutilized. Virtualization lets you run multiple virtual machines on a single physical machine, with each virtual machine sharing the resources of that one physical computer across multiple environments. Different virtual machines can run different operating systems and multiple applications on the same physical computer

2.12.1 Virtual Infrastructure

A virtual infrastructure lets you share your physical resources of multiple machines across your entire infrastructure. A virtual machine lets you share the resources of a single physical computer across multiple virtual machines for maximum efficiency. Resources are shared across multiple virtual machines and applications. Your business needs are the driving force behind dynamically mapping the physical resources of your infrastructure to applications—even as those needs evolve and change.

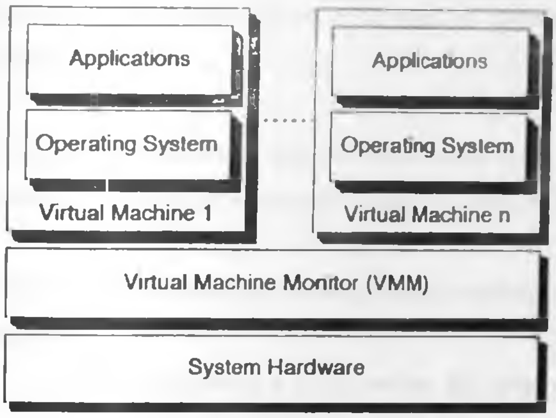


Figure 11 Virtual Machine Monitor - Virtual Machine Relationship

According to Robert Rose(2004), the VMM provides a virtual processor and other virtualized versions of system devices such as I/O devices, storage, memory, etc. The VMM also provides isolation between the virtual machines it hosts so that problems in one cannot effect another.

2.12.2 Virtual Machine Monitor

Robert Rose(2004), in his article, states that the idea of a Virtual Machine Monitor (VMM) goes hand-in-hand with virtual machines. The VMM is the software component that hosts guest virtual machines. In fact, the VMM is often referred to as the host and the virtual machines as guests. The VMM is a software layer that abstracts the physical resources for use by the virtual machines.

The CORBA Platform

The CORBA platform is based on the concept of interoperable objects, providing inter-connection support common to all the objects. Thus, client components use the basic services (naming, persistence, events, trading, etc.) and common facilities for component collaboration. Some of the CORBA features are: interoperability among objects, client request transparency, portability, object reuse, etc. The four key elements of this architecture are:

a) Object Interconnection Support - ORB (Object Request Broker): The client objects are separated from mechanisms used for communication, activation, or storage of the server objects.

b) Object Services: Packed as components with well defined interfaces, these services extend the Object Interconnection Support capabilities. The following services may be adopted: naming, acknowledgement of events, persistence, management of life cycle, transactions, concurrence control, relationships and externalisation.

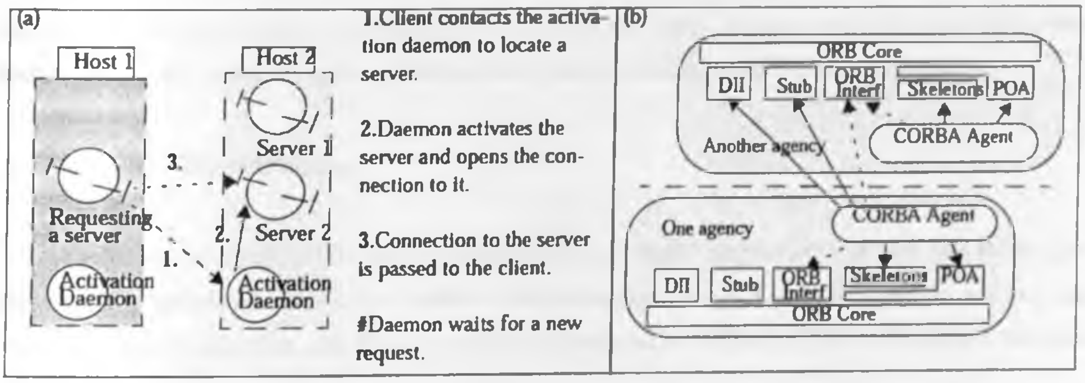
c) Common Facilities: These collections of components define the engagement rules for application objects.

d) Application Objects: These are specific objects for user application and are built based on the services provided by Object Interconnection Support, common facilities, and object services

The CORBA platform has several advantageous features, besides interoperability between objects, as follows:

- When requesting a given service, the client need not concern himself about the location of the service provider since interaction occurs abstractly and for that reason the client does not know if the requested object is a local or a remote one;
- The object concept reduces the number of codes that need to be written since it incorporates concepts like polymorphism inheritance;
- The objects are application-independent, reusable and portable and can therefore be used in heterogeneous systems;
- The objects are also self-contained, i.e., they include behavior, interaction and relationships, and can perform actions.

Location and migration distribution transparencies are considered in the context of multi agent systems. The extension of mobile agent facility architecture is proposed with the addition of an availability service and a transparency service. Transparent mobility of services is explored in applications using overall load balancing of resources.



1.Client contacts the activation daemon to locate a server.

2.Daemon activates the server and opens the connection to it.

3.Connection to the server is passed to the client.

#Daemon waits for a new request.

Figure 12 . (a) Client / Server binding with an activation daemon. (b) CORBA Agents:

CORBA Naming Service. The Naming Service [Vogel97] locates object implementations and thus it is a fundamental service for distributed object systems. It provides an extra layer of abstraction for the identification of objects, allowing readable object identifiers and persistent identification mechanism, i.e., objects can bind themselves under the same name regardless of their object reference.

Mobile Agent Security

Despite its many practical benefits, mobile agent technology results in significant new security threats from malicious agents and hosts. The primary added complication is that, as an agent traverses multiple machines that are trusted to different degrees, its state can change in ways that adversely impact its functionality.

Wayne Jansen, Tom Karygiannis (2000) , Threats to security generally fall into three main classes: disclosure of information, denial of service, and corruption of information. There are a variety of ways to examine these classes of threats in greater detail as they apply to agent systems.

- **Masquerading**

When an unauthorized agent claims the identity of another agent it is said to be masquerading. The masquerading agent may pose as an authorized agent in an effort to gain access to services and resources to which it is not entitled. The masquerading agent may also pose as another unauthorized agent in an effort to shift the blame for any actions for which it does not want to be held accountable. A masquerading agent may damage the trust the legitimate agent has established in an agent community and its associated reputation.

- **Unauthorized Access**

If the agent platform has weak or no control mechanisms in place, an agent can directly interfere with another agent by invoking its public methods (e.g., attempt buffer overflow, reset to initial state, etc.), or by accessing and modifying the agent's data or code.

Modification of an agent's code is a particularly insidious form of attack, since it can radically change the agent's behavior (e.g., turning a trusted agent into malicious one). An agent may also gain information about other agents' activities by using platform services to eavesdrop on their communications.

- **Eavesdropping**

A host provides the hardware and other resources that an agent executes on. A host has fairly good opportunity to eavesdrop since it can monitor every execution the agent makes. The host not only can access the agent's code, state, and data, it can also infer information of the agent from the request the agent makes.

- **Alteration**

A host can alter the code, the state, or the data of an agent requesting service on that host. Since the agent normally visits a series of hosts to carry out its tasks, we may not be able to track down who made the alteration if we do not detect it in a timely manner.

The mobile agents are under so many possible attacks from the host/hosts, can mobile agent protect actively and effectively protect itself from the host/hosts? Since mobile agents are at the mercy of the host/hosts for resources, some researchers believe mobile agents cannot effectively protect themselves from host(s)' attacks. Although the research for effective methods is still in infancy, we believe it is possible to protect mobile agents from a host/hosts.

The protection of an executable program in an insecure environment is more important for the mobile agent. Because the mobile agent is comprised of the code and state information needed to carry out some computation. Mobility allows an agent to move, or hop, among agent platforms.

Because the mobile agent has the loose roaming itineraries in many agent applications, many traditional security means cannot work very well.

Because attackers can tamper the agent from the three ways as following: data , code and state.

Software solutions

Obfuscation:
Obfuscation, proposed by Hohl (www.informatik.uni-stuttgart.de)has for object the generation from an ordinary agent A another agent A' equivalent in functions but difficult to analyse. The idea consist in the violation of software engineering rules like the use of GOTO instead of recommended loops, the use of insignificant variable names, the replacement of procedure calls by the procedure body and the insertion of useless code Hamed Aouadi and pr Mohamed Ben Ahmed (2006).

Execution traces:
In this approach every site visited by the agent generates a trace of the agent execution. This trace contains every code line, every variable and every external variable read by the agent. Before the migration of the agent, they calculate the hash of the trace (with a hash function like, sign that hash and associate the result to the agent. A light version of this solution consists in the agent code split in white segments and black segments. The trace is calculated only on the black segments where the agent interacts with the visited platform. Hamed Aouadi and pr Mohamed Ben Ahmed (2006)

Threat Class	Threat Subclass	Suitable Countermeasures
Integrity Attack	Integrity interference	Trusted execution environment Encryption Reference states
	Information modification	Tamper resistant hardware Trusted execution environment Detection objects Itinerary recording8 Anonymous itinerary Partial result encapsulation & authentication Cryptographic tracing
Availability Refusal	Denial of service	Trusted execution environment Server replication Path histories Cryptographic tracing
	Delay of service	Trusted execution environment Path histories Server replication
	Transmission refusal	Trusted execution environment Server Replication
Confidentiality Attack	Eavesdropping	Trusted execution environment Using a mobile agent system Encryption Environmental key generation

Table 1 Threat Classes and Corresponding Suitable Countermeasures Table

CHAPTER 3:

METHODOLOGY

3.1 Introduction

This chapter presents the various methodologies that were employed in doing this research project. It explores in details how the project was carried out. It shows the major stages that were involved the activities carried out on every stage. The project aims to find out how mobile agent can be used in a networked environment to help users in locating, retrieving and synchronizing of files.

3.2 Information Gathering

Materials contained in this projected were collected from journals that provided vital information for literature review section. Internet played a major role in the research because in the area of mobile agents, very few books are available. Information was also obtained from texts books which were gotten from the university library.

- **The Prototype**

The findings of the research were implemented in a prototype that was developed in a virtual environment. The section below outlines how major activities were carried out.

- **Virtualized Environment**

The first step in this project involved setting up a virtual environment comprised of the following a components.

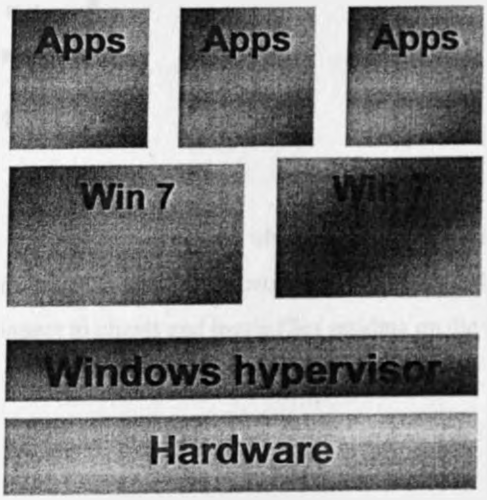


Figure 13: Virtual Machine environment

Windows hypervisor. This will be the host operating system which other guest operating system will run on top of it. For this project I will use windows 7 home edition

The hypervisor is the single most security critical element in the system. It is the hypervisor that provides isolation between different virtual machines.

Win Xp Operating system - this will the guest operating system which will be running in virtual box. This the operating that will have the java installed which will support the agent to execute

Hardware. This is the actual computer that the hyper visor sits on. The laptop will be the hardware in this case.

Setting of agent environment.

The research was carried out by conducting literature review on mobile agents . Agent communication was explored to know how agents communicate.

Agent system model was studied to get an overview of an agent in a networked environment . This was done to understand the security of agent

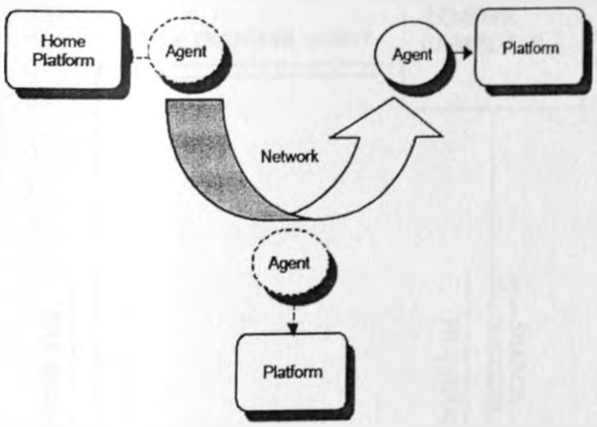


Figure 14: Agent System Model

Testing

Testing was to be done on a virtual environment whereby the control server was residing on a remote location and the client also residing on a remote location. To ensure that the application was working, the server was to successfully connect to clients and locate files residing on the client machine

CHAPTER 4:

4.1 ANALYSIS AND DESIGN

This chapter outlines the design of the prototype, the components of the prototype and the interfaces that the user will be interacting with. The conceptual model comprises of the user, graphical user interface, the network and remote computer.

CONCEPTUAL MODEL

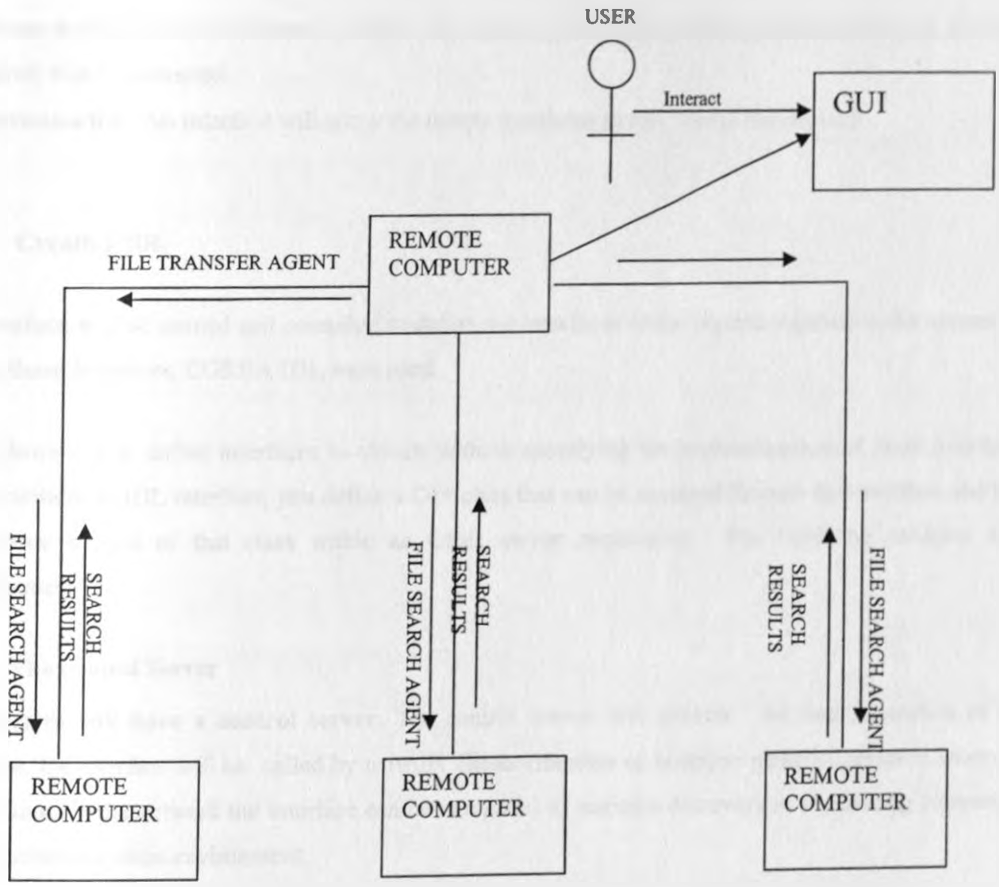


Figure 15: Conceptual Model

The user will interacts with the application through a graphical user interface (GUI) provided, through IDL, from this interface the user will be able to see the following modules, Commands- this interface will allow the user to see the client machines that have been registered by the control server. The user cannot access a computer that has not been registered. It should noted that client machines should be registered ant this can be set in the registry so that whenever the computer is switched on, it is directly registered to avoid users registering their machines every time they switch them on.

Download file-This interface will allow the user to connect to the desired computer and be able to see the directories that exist in the client machine. The interface will contain a button that the user to open a folder and select the desired file.

Upload file- will allow user to upload files to a desired destination

Search- allows the user to search for files that exist in a given coimputer

Traverse directory- this interface will allow the user to go through the directories that exist in the client computer that is connected.

Synchronize file- this interface will allow the user to synchronize two files in the network

4.2 Creating IDL

Idl interface will be created and compiled to define the interfaces to the objects required in the system. To define these interfaces, CORBA IDL were used.

IDL allows you to define interfaces to objects without specifying the implementation of those interfaces. To implement an IDL interface, you define a C++ class that can be accessed through that interface and then you create objects of that class within an Orbix server application. The following modules were incorporated.

4.3 The control Server

The system will have a control server. The control Server will provide the implementation of the interface, the interface will be called by network clients (monitor or bonet) to make its presence known to other clients in the network the interface can be compared to resource discovery or monitoring component in a distributed system environment.

The control Server will maintain a list of connected clients in an array. When the clients call this method with their identifiers as arguments the control Server will add the client identifiers to the list (array) of connected clients

The system will also provides a method through which the network clients can queries the list of connected clients, a method useful when the network users want to know which clients are available.

The controlServer returns the list of connected clients as an array to the calling client, done by using the corba's out direction utility.

4.4 The search.

This module will be the one to perform the search. The search File method searches the network clients (bonets) for files whose names match the search file (file to be searched). The monitor will send the search string to each and every connected client. The connected clients then queries all the directories in the system starting from the root and then sends back the details of the files that match the search string. the request is asynchronous i.e the monitor doesn't have to wait for the search result once it has send the request

4.5 interface

The interface will provides access to a remote client shell, it will executes the provided string arguments then sends back the results to the monitor

The method will makes use of java.lang library utility to interface and pass commands to the shell. For perfomance enhancements the multithreading concept is used.

The method will make use of two threads:

I. executingThread
II. supervisingThread

The executing thread is in charge of executing the given command and handling any errors that might arise

The supervisingthread will monitor the executing thread to detect nonresponsive threads, these threads are then terminated where necessary i.e when blocked.

4.6 Listing Directories

This method will enable listing of the contents of directories

The method receives two arguments (directory name and direction of traversal either up/down). On up it lists the contents of the parent directory while on down it lists the contents of the supplied directory.

When it receives an empty directory argument(" ") it queries for the root directories the method returns asterisks(*) to designate the contents of an empty directory.

The method lists the contets of the supplied directory the method also incoporates means by which the requesting clients determines whether the returned argument is a file or directory, enabled by use of the flags parameter

4.7 Download

The method will use java.io library utility to read the contents of a local file (respect to the bonet) into a string then transmit this result to the requesting client, the requesting client then use this string sequence as an input stream to build the requested file. The method calls the copy in method with the string stream argument (converted input stream) to complete the download request.

4.8 Synchronize:

The synchronize method will be used to maintain consistencies between files in remote computers by synchronizing their contents based on the selected criterion. The ctriterion used could be by time last modified or by their sizes. The most recent time means the most consistent. Likewise to size the largest means that content has been added which means it is the latest thus the most suitable.

4.9 Defining IDL Interfaces

IDL interface describes the functions that an object supports in a distributed application. Interface definitions provide all of the information that clients need to access the object across a network

4.10 Pattern Used

Master-Slave

The Master-Slave pattern defines a scheme whereby a master agent can delegate a task to a slave agent. The key idea of the master-slave pattern is to use abstract classes, Master and Slave, to localize the invariant parts of delegating a task between master and slave agents: dispatching a slave back and forth to other destinations, initiating the task's execution, and handling exceptions while performing the task.

Master and slave agents are defined as subclasses of Master and Slave, in which only varying parts such as what task to perform and how the master agent should handle the task's result are implemented.

4.11 The Monitor interface

The Monitor provides the interface to the system; the monitor is in charge of connecting to remote clients, passing a message/command to remote clients and getting back the status from remote clients. Thus its methods are concerned with display of status and any feedback information from remote clients.
Both methods are concerned with outputting information to the Monitor's screen

CHAPTER 5:

IMPLEMENTATION

5.1 Introduction:

This section provides an overview of how implementation was done including the requirements. Details of both software and hardware requirements are given under this chapter.

Virtual machine technology is gaining in popularity daily. When installed, users can run multiple operating systems on a single computer and can quickly switch between these applications with the click of a mouse button.

5.2 System Requirements:

The Physical Computer Requirements:

The minimum system requirements for the host computer (which is the computer system that you are physically installing the Virtual Machine software) is as follows

Supports any Pentium ***compatible processor with at least 400MHz (1 GHz is recommended) with L2 cache.***

- At least 20 MB of hard disk space
- DVD or CD-ROM Drive
- Monitors using at least Super VGA (800x600) resolution.
- Can be installed on any of the following operating systems:

  ✓ Vista Enterprise
  ✓ Vista Ultimate
  ✓ Vista Business
  ✓ Windows Server 2003 Standard Edition
  ✓ Windows Server 2003 Standard x64 Edition
  ✓ XP Professional

- **Supported processors:**
  AMD Athlon/Duron
  Intel Celeron
  Intel Pentium II
  Intel Pentium III
  Intel Pentium 4

Intel Core Duo

Intel Core2 Duo

The Guest Operating System Requirements:

Caution. When installing guest operating systems keep in mind that they can quickly fill up disk space and use up available memory. Below, the table below gives the amount of memory and disk space that is needed to run the guest operating system:

Guest Operating System	Minimum Hard Disk Space	Minimum Memory
Windows 98, Windows 98 Second Edition	500 MB	64 MB
Windows Millennium Edition (Windows ME)	2 GB	96 MB
Windows 2000 Professional	2 GB	96 MB
Windows XP Home Edition	2 GB	128 MB
Windows XP Professional	2 GB	128 MB
Windows Vista Enterprise	15 GB	512 MB
Windows Vista Business	15 GB	512 MB
Windows Vista Ultimate	15 GB	512 MB
OS/2	500 MB	64 MB

Table 2: showing requirements for operating system.

5.3 Setting up Virtual Environment: VMWare

To create the environment, you need to first install VMWare on the computer. Once that is done, you need to load an OS into the VMWare.

<u>Agent Design</u>

Once the configuration is set, choose the Operating System that is going to be installed in this VM. This doesn't install the OS, but just sets the environment compatible to install the OS that you have chosen from the list.

<u>Setting network in vm ware</u>

Once the OS path is set, move on to Network Settings where you have four options. Setting the network type is most important when it comes to how you would like to use this VM later, though VMWare is created for ease of use and hence you could change the network settings at any point of time even after the installation is done.

There are four modes of network types to choose as shown below

Bridged mode

NAT'ted mode

Host-only mode

No Network Connection

For this project, I used Bridged Mode.

Bridged Mode:

In this mode, the guest operating system would have direct access to the external network of the host on which VMWare is running. The guest will have its own IP address on the external network.

NAT'ted Mode:

In this mode, the guest operating system would connect to the external network using the host IP address.

Host-only Mode:

In this mode, the guest operating system would connect to the virtual network within the VMWare, and hence this mode can be used for different VM's to connect with each other inside the virtual environment.

No Network Connection:

In this mode, as the name suggests there will be no network connection from this guest OS.

5.4 Corba/ multiagent

The interface client provides the following methods

Execute (in string command, in string monitorName): which receives two string parameters: command –the command to be executed by the bonet. bonetName –the name of the monitor requesting the operation, this name is useful to communicate the results back to the monitor.

Arguments list_directory(in string directory, in string dirrection): returns arguments: a sequence of strings (array), i.e. the contents of the given directory.

Void download(in string filename, in string dnldServer, in string destination_host, in string destination_dir): downloads a specified file to a specified host, the additional parameters are useful in logging details.

void copy_in(in string input_stream ,in string path,in string dnldServer): this method is invoked by a remote host when to complete a download request.

Void searchFile(in string fileName,in string monitorName): a method that searches for and returns all the files whose name contains or similar to the specified fileName parameter. The results are communicated back to the monitor enabled by the monitorName parameter.

Arguments list_roots(in string hostname): interface's method that returns string sequence of root directories the hostname parameter specifies the calling host. Used for logging and completing the request

void listdir(in string path ,out arguments dirlist, out flags fl): interface's method that lists the contents of a directory specified by path, the method uses out parameters to relay back the results to the calling host.

Void rename(in string oldName, in string newName, in string requestingHost): interface's method that renames the file oldName to newName. The requesting host is used for logging and sending back any errors that may occur in the process.

Void delete(in string deleteFile, in string requestingHost): interface required to delete a file remotely. The details – requestingHost and the deleted file - are logged.

Void getprops(in string fileName, out long size, out boolean canread , out boolean canwrite, out boolean isFile, out long lastModified): a method called to obtain the properties of a network file. the method uses the out parameters for message passing.

Void synchronize (in string criteria, in string hostA, in string dirA, in string hostB, in string dirB): a method to synchronize a pair of network files. The method uses the specified criteria i.e. by last modified or by size to synchronize the files.

void queryFolder (in string queryHost, in string dirName, out flags size, out booleanseqs isFile, out flags modified, out arguments ls, out arguments canonical): a method to obtain the details of a file or folder used when synchronizing files.

NB
All of the methods above are contained in the bonet module and each participating client in the network can be seen as a bonet, thus all the clients in the network are capable of providing the described services.

CHAPTER 6:

6.1 EXPERIMENTAL RESULTS

The experiment was conducted in order to test the following, Agent can be able to move in the network and be able to access files residing on a remote machine, and the mobile agent performs better than client server. Two different types of experimentations were done. In one of the experiments the response time was measured against the number of users, while the other experiments tested the file size against response time. All these experiments were conducted under the following conditions.

- The computer had a CPU of 2.13 GHz with 1.96 amount of RAM
- Microsoft window seven was the operating under which the agent was running on
- Virtual Box 4.0 was the Virtualization platform under which the operating system was running on.
- The two computers were both connected with Ethernet cable with speed of 100Mbs

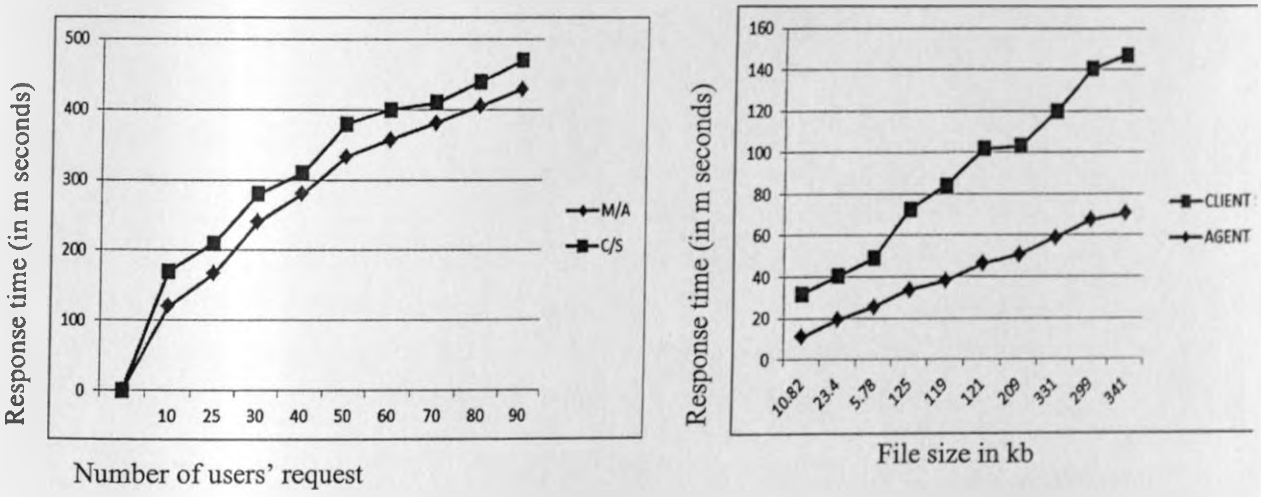


Figure 16: Comparison between client server and agent based remote file locator

From the above results, mobile agent seems to perform better than client server method which seems to experience delays. For this case mobile agent are far better that client server method which to take more bandwidth that client base. Mobile Agents transparently use the network to accomplish their tasks, while taking full advantage of resources local to the many machines in the network. They process data at the data source, rather than fetching it remotely, allowing higher performance operation. They use the full spectrum of services available at each point in the network, such as GUIs at the user and database interface on servers. They make best use of the network as they travel.

The client server seems to be taking too much time to respond as compared to agent based method. This can be attested by the fact that Client makes a request by sending a message to the server and the

Server unwraps the message, decodes it and processes the request and sends the reply in the form of a similar message back to the Client. This operation overwhelms the network whereby bandwidth is greatly consumed. Whereas the mobile transport the code to do the processing locally, client server model continue to transmit data over the network for processing.

CHAPTER 7:

7.1 CONCLUSIONS AND FUTURE WORK

Mobile agents have been found to be beneficial in distributed environment because of many benefits they posses. The main objective of this project was to research on mobile agents an application that allow user to connect to a remote machine and be able to manipulate data. This application was supposed to allow the user in a given location to connect to a remote computer through a mobile agent and be able to manipulate data. This has been implement using java programming language which offers many advantages when dealing with mobile agents. The application that was developed allowed copying of files, deletion, synchronization and renaming of files that reside on a remote machine.

Application areas of mobile agent were reviewed in this project. Researches carried by scholars were reviewed to find out what they have done. Application areas of mobile agents have been discussed in details and proposed frameworks discussed. The application areas featured are, Networking, where it was found that mobile agents are being employed in network management by network administrators because of the benefit that mobile agent offers. Another application area discussed is on E-commerce. It was found that mobile agent is being employed on the internet to do shopping, including making orders and potentially even paying. Mobile agent is being employed in parallel computing. It was found that these agents migrate to computers on the network, which have the required resources and use them to solve the problem in parallel thereby reducing.
Other application areas of mobile agent discussed are mobile computing, data collection and human tracking. This section of literature review achieves the objective set for this project.

An evaluation on mobile agents and their applications in a distributed environment was conducted and it was found that an increasing degree of flexibility, adaptability, and autonomy in distributed environments, mobile agents plays a big role in such environments. This project found that mobile agent can carry on a task while the connection to the computer is temporally lost and then continue once the link returns to send the found result. Due it advantage mobile agents perform well in distributed environment than other technologies. Mobile agent development environment was also researched where languages used in agent development were discussed.

A study on agent mobility was conducted where it was found that code mobility can be divided into three steps: determining the code operation targets, transferring the code, and integrating the code into the target system. Due to agent property of transferring code instead of data, this can be employed in file searching whereby agent can be sent into a remote machine to do local searches while freeing the client to do other operations. Code mobility is a property that makes mobile agent appropriate in doing searches as compare to client server paradigm

Agent communication was also highlighted in this project o shed light on Agents. It was found that agent communicate with users, resources, and with each other to cooperate or negotiate. Common languages that agent uses for communication were tackled in this section of literature review.

The agent based application that connects to a remote machine and allow user to locate file, synchronize files has been developed to demonstrate agent mobility in a remote location. This application was able to connect to a remote machine and allow user to access files, synchronize files and so on. Based on testing and analysis performed in this project, it can be concluded that mobile agent can be used in distributed environment to enable users perform their operations as if they were on a local machine while saving on bandwidth.

7.2 Contribution made and achievements.

Through the prototype this project was able to prove that mobile agents can used together with CORBA framework in a distributed environment to tap benefit of CORBA while incorporating mobile agent benefits.

The research demonstrated that it is possible to develop mobile agent that can operate in a virtual environment which allows user to share the resources of a single physical computer across multiple virtual machines for maximum efficiency

This work demonstrated that it is possible to develop mobile agents based on java that are able to move to a remote computer and manipulate data based on user needs

Through experimentation the research findings found that mobile agents are able to save on bandwidth compared to client server architecture

7.3 Future Research

Though this project is complete, it has not fully covered mobile agents especially in terms of security. Users in a distributed environment continue to treat mobile agents with caution because of the security issue surrounding them. I therefore recommend further research in this aspect because security remains a challenge. This project based it implementation in a virtual environment as a platform for development. Performance model to predict performance of mobile agent in a virtualized environment need to be researched because during my research, I have not come across any model that address performance prediction of virtual machine. If you can be able to predict the performance of a virtual machine and how this performance affects mobile agents then this will be a yard stick to come up with mobile agents that performs better.

References

1. Abdelkader Outtagarts , (2009). BestDeal by using Mobile Agent, International Conference on Information Management and Engineering
2. Abdelkader Outtagarts, (2009).Mobile Agent-based Applications : a Survey. Nozay, France
3. Amitava Dutta-Roy, (1999), Bringing home the Internet. IEEE Spectrum,
4. B.Schulze, (1998). Service Migration and a Transparency Service. GMD FOKUS,German
5. *Braubach,* (2006). Jadex Agent System
6. *Campione, M. & Walrath, K.* (1996). The Java Tutorial: Object-Oriented Programming *for the Internet.* Reading, Massachusetts, Addison-Wesley
7. Carlos Arias Mendez University of Magallanes, (1999) .Agent Migration Issues in CORBA Platforms
8. Corey Grice, (1999).When will data change the wireless world?, CNET NEWS.COM
9. David Kotz and Robert S. Gray, (1999), The Future of the Internet .Thayer School of Engineering Dartmouth College Hanover, New Hampshire 03755 Mobile code:
10. Do, van Thanh,(2001), Using Mobile Agents in Telecommunications, Proceedings of the International Workshop for Internet Bots: Systems and Applications
11. Felicio et al., (2009) . Platform as Support for Distributed Virtual Environments
12. Fritz Hohl,(1997).An approach to solve the problem of malicious hosts
13. H Kakiuchi at al., (2010), An Algorithm to Determine Neighbor Nodes for Automatic Human Tracking System,.
14. Kiniry, J and D. Zimmerman, (1997), Special Feature: A Hands-On Look at Java Mobile Agents,
15. Belkhelladi ,K , Chauvet . P and Schaal .A. (2009) A mobile agent framework to support parallel computing -Application to Multi-product Planning and Scheduling Problems
16. Kunal Shah(2003) Performance analysis of mobile agents in wireless internet applications using simulations.
17. Li Jingyue (2002) Code Mobility Overview (Essay for DIF 8914)
18. Ioan M. R. and Paula S. L.,(2008).Using Mobile Agents and Intelligent Data Analysis Techniques for ClimateEnvironment Modeling and Weather Analysis and Prediction, 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing.
19. Genesereth M.R. and Katchpel S.P, (1994).Software Agents," Comm. ACM, Vol. 37, No. 7, 1994, pp. 48–53, 147.
20. Mihai Barbuceanu et al., (2004). Mobile Agents Intelligent Assistants Agent Technology: Enabling Next Generation Computing by Building Agents toServe Customers
21. Outtagarts, A., Kadoch.(1999). Client-Server and Mobile Agent: Performances Comparative Study in the Management of MIBs. Proceedings of the First International Workshop on Mobile Agents for Telecommunication Applications

22. Parineeth M Reddy(1997), On the Internet Languages for Mobile
23. Al-Jaljouli R., Abawajy J.,(2009). Agents Based e-Commerce and Securing Exchanged Information
24. PATEL R. B., GARG .K., (2005). A Comparative Study of Mobile Agent and Client-Server Technologies in a Real Application retrieval
25. Robert,et al., (2004). A note on distributed computing Survey of System Virtualization Techniques. Microsystems Laboratories, Nov
26. Manvi S.S. and Venkataram .P., (2007).Mobile agent based approach for QoS routing
27. Kulkarni U.P. et al ., (2005),Agent Tracking: A Reliable Agent Forwarding Mechanism
28. Wayne Jansen et al., (2000). Mobile Agent Security. Gaithersburg, MD : U.S. Dept. of Commerce
29. Autran G., X. Li, (2009), Implementing an Mobile Agent Platform for M-commerce, 33rd Annual IEEE International Computer Software and Applications Conference, 2009, p. 50-45.

APPEDIX

Interface specification

All the interfaces were defined in an IDL file pc.idl, the file contains all the interface specifications used by the different system modules – controlServer, bonet, Monitor.

The controlServer comprised two interfaces

```
interface server{
    typedef sequence<string> clients;
    void register(in string hostname);
    void get_connected_clients(out clients connected_clients);

};
```

i. Interface through which remote clients can register themselves on the network
ii. Interface through which the connected clients can query the control server to find out the names of registered clients, useful when binding to these interfaces.

The bonet interfaces

The bonet, a remote client module that is responsible for most of the remote functions operations that the system performs. All the clients that the system "sees" must run the bonet daemon.

...

```
interface client{
     typedef sequence<string> arguments;
           typedef sequence<long> flags;
           typedef sequence<boolean> booleanseqs;
     void execute(in string command,in string monitorName);
           arguments list_directory(in string directory,in string dirrection);
           void download(in string filename,in string dnldServer,in string destination_host, in string
destination_dir);
           void copy_in(in string input_stream ,in string path,in string dnldServer); // check invalid path
           void searchFile(in string fileName,in string monitorName);
           arguments list_roots(in string hostname);
           void listdir(in string path,out arguments dirlist,out flags fl);
           void rename(in string oldName,in string newName,in string requestingHost);
           void delete(in string deleteFile,in string requestingHost);
           void getprops(in string fileName,out long size,out boolean canread,out boolean canwrite,out
boolean isFile,out long lastModified);
           void synchronize (in string criteria, in string hostA,in string dirA,in string hostB, in string dirB);
           void queryFolder (in string queryHost, in string dirName, out flags size, out booleanseqs isFile,
out flags modified, out arguments ls,out arguments canonical);
```

The interface client provides the following methods:

execute(in string command, in string monitorName): which receives two string parameters: command –the command to be executed by the bonet. bonetName –the name of the monitor requesting the operation, this name is useful to communicate the results back to the monitor.

arguments list_directory(in string directory, in string dirrection): returns arguments: a sequence of strings (array), i.e. the contents of the given directory.

void download(in string filename, in string dnldServer, in string destination_host, in string destination_dir): downloads a specified file to a specified host, the additional parameters are useful in logging details.

void copy_in(in string input_stream ,in string path,in string dnldServer): this method is invoked by a remote host when to complete a download request.

void searchFile(in string fileName,in string monitorName): a method that searches for and returns all the files whose name contains or similar to the specified fileName parameter. The results are communicated back to the monitor enabled by the monitorName parameter.

arguments list_roots(in string hostname): interface's method that returns string sequence of root directories the hostname parameter specifies the calling host. Used for logging and completing the request.

void listdir(in string path ,out arguments dirlist, out flags fl): interface's method that lists the contents of a directory specified by path, the method uses out parameters to relay back the results to the calling host.

void rename(in string oldName, in string newName, in string requestingHost): interface's method that renames the file oldName to newName. The requesting host is used for logging and sending back any errors that may occur in the process.

void delete(in string deleteFile, in string requestingHost): interface required to delete a file remotely. The details – requestingHost and the deleted file - are logged.

void getprops(in string fileName, out long size, out boolean canread , out boolean canwrite, out boolean isFile, out long lastModified): a method called to obtain the properties of a network file. the method uses the out parameters for message passing.

void synchronize (in string criteria, in string hostA, in string dirA, in string hostB, in string dirB): a method to synchronize a pair of network files. The method uses the specified criteria i.e. by last modified or by size to synchronize the files.

void queryFolder (in string queryHost, in string dirName, out flags size, out booleanseqs isFile, out flags modified, out arguments ls, out arguments canonical): a method to obtain the details of a file or folder used when synchronizing files.

NB
All of the methods above are contained in the bonet module and each participating client in the network can be seen as a bonet, thus all the clients in the network are capable of providing the described services.

The Monitor interface

The Monitor provides the interface to the system; the monitor is in charge of connecting to remote clients, passing a message/command to remote clients and getting back the status from remote clients. Thus its methods are concerned with display of status and any feedback information from remote clients.

Both methods are concerned with outputting information to the Monitor's screen.

IMPLEMENTATION OF THE INTERFACES:

The IDL interfaces were implemented by writing servant classes with java language, for each interface defined in corba a servant class was written in java and compiled, IDL to java mappings played a crucial role. The procedure works as follows:

When for each interface named for instance foo is declared in a corba-IDL file, the idl-to-java compiler creates an abstract class _fooImplBase which must be implemented by extending the class, the concrete class (one extending the abstract class) must implement all the methods defined in that particular interface.

Example:
controlServer functionality is defined by the interface server.

```
...

interface server{
  typedef sequence<string> clients;
  void register(in string hostname);
  void get_connected_clients(out clients connected_clients);

};
```

Using the above example the idl-to-java compiler creates an abstract class _serverImplBase shown below.

```
/**
* pc/_serverImplBase.java .
* Generated by the IDL-to-Java compiler (portable), version "3.1"
* from pc.idl
* Sunday, April 4, 2010 12:03:07 PM PDT
*/


 private static java.util.Hashtable _methods = new java.util.Hashtable ();
 static
 {
   _methods.put ("register", new java.lang.Integer (0));
   _methods.put ("get_connected_clients", new java.lang.Integer (1));
 }
$rh)
 {


   switch (__method.intValue ())
   {
     case 0:  // pc/server/register
     {
       String hostname = in.read_string ();
       this.register (hostname);
```

```
        out = $rh.createReply();
        break;
      }

      case 1:  // pc/server/get_connected_clients
      {
        pc.serverPackage.clientsHolder connected_clients = new pc.serverPackage.clientsHolder ();
        this.get_connected_clients (connected_clients);
        out = $rh.createReply();
        pc.serverPackage.clientsHelper.write (out, connected_clients.value);
        break;
      }

} // class _serverImplBase
```

Having the declared all the interfaces that the system needs as described in the preceding section, the servant classes were then implemented in java major challenge being posed by the narrow primitive data types offered by corba IDL, this lead us to resolve the problem using enumerated types declaration. The enumerated data types though proved useful added complexities to the system coding more so complexities arising from idl-to-java mappings.

The section that follows describe in detail how the system implements its functionalities, enumerated below:

controlServer

register
get_connected_hosts

monitor

printResult
printSeachReslt

bonet

execute
list_directory
download
copy_in
searchFile
list_roots
listdir
rename
delete
getprops
synchronize
queryFolder

The controlServer

The controlServer provides the implementation of this interface, the interface is called by network clients (monitor or bonet) to make its presence known to other clients in the network the interface can be compared to resource discovery or monitoring component in a distributed system environment
Implementation:
The controlServer maintains a list of connected clients in a array. When the clients call this method with their identifiers as arguments the controlServer adds the client identifiers to the list(array) of connected clients

```
public class serverServant extends _serverImplBase{
        String[] conPcs=new String[0];
        public void register(String str){
                String[] currPcs;
                if(conPcs.length<1){
                        conPcs=new String[1];
                        conPcs[0]=str;
                }
                else{
                        currPcs=new String[conPcs.length+1];
                        currPcs[currPcs.length-1]=str;
                        for(int i=0;i<currPcs.length-1;i++){
                                currPcs[i]=conPcs[i];
                        }
                        conPcs=null;
                        conPcs=new String[currPcs.length];
                        conPcs=currPcs;

                }

        }
```

Connecting client

The system provides a method through which the network clients can queries the list of connected clients, a method useful when the network users want to know which clients are available.
The controlServer returns the list of connected clients as an array to the calling client, done by using the corba's out direction utility.

```
public void get_connected_clients (pc.serverPackage.clientsHolder connected_clients){
                connected_clients.value=conPcs;
        }
    }
```

The monitor
how the system implements monitor's printResult.
The funciotn is called by a remote client to communnicate back the results of an execution.

```
class monitorInterfaceGui extends _monitorImplBase{
        private String monitorName=null;
        Monitor monitorRef;
        public monitorInterfaceGui(String monitorN,Monitor gui){
                monitorName=monitorN;
                monitorRef=gui;
        }
```

```
    public void printResult(String output){
            monitorRef.appendText(output);
    }

...

//monitor

final JTextArea tx=new JTextArea(20,20);

public void appendText(String str){
                tx.append("\n"+str);
 }
```

Implementation;

The method simply prints out the supplied string arguments to a text area in the monitors GUI

printing Seach Reslt.

This method is specific to the search method implemented by the bonets. It is used by the bonets to communicate back the search results.

Implementation:
The method call

```
public void printSeachReslt (String fileName){
          monitorRef.appendSearchFile(fileName);
   }

...

//monitor

DefaultListModel searchresult;

public void appendSearchFile(String file){
                searchresult.addElement(file);
 }
```

The bonet
how the bonet implements the interface execute:

The interface provides access to a remote client shell, it executes the provided string arguments then sends back the results to the monitor
Implementation:
The method makes use of java.lang library utility to interface and pass commands to the shell. For perfomance enhancements the multithreading concept is used.
The method makes use of two threads:

III. executingThread
IV. supervisingThread

The executingThread is in charge of executing the given command and handling any errors that might arise The supervisingthread monitors the executing thread to detect nonresponsive threads, these threads are then terminated where necessary i.e when blocked.

```
class executingThread extends Thread{
        String cmd=null;
        int progressBar=0;
        String remoteMonitorName;
        pc.monitor remoteMonitor;
        String[] args;


        public executingThread(){

        }
        public executingThread(String comand,String monitorName){

                cmd=comand;
                remoteMonitorName=monitorName;
                pimpHelper ph=new pimpHelper(new args_holder().args);
                remoteMonitor=ph.connectMonitor(monitorName);
        }
        public void setArgs(String[] args){
                this.args = args;
        }
        public void run(){
                try{
      Runtime run=Runtime.getRuntime();
      Process pr=run.exec(cmd);
      //pr.waitFor();

      BufferedReader br=new BufferedReader(new InputStreamReader(pr.getInputStream()));

      String line=" ";
      while((line=br.readLine())!=null){
        remoteMonitor.printResult(line);
        progressBar++;
        //sleep();
      }
                }
                catch(Exception e){
                        remoteMonitor.printResult("exception caught "+e);
                }
        }

class supervisingThread extends Thread{

        int benchmark=0;
        executingThread supervisedThread=null;
        public supervisingThread(executingThread excT){
                supervisedThread=excT;
        }
        private boolean isNotProgresing(){
                return benchmark==supervisedThread.progressBar;
        }
        public void run(){
                while(true){
                        try{
                                sleep(10000);//
                                if(isNotProgresing()){
                                        supervisedThread.interrupt();
                                }
                                benchmark+=1;
                        }
                        catch(Exception e){

                        }
                }
        }
}
```

How the sysem implements bonets interface list_directory

This method lists the contents of directories

Implementation:
The method receives two arguments (directory name and direction of traversal either up/down). On up it lists the contents of the parent directory while on down it lists the contents of the supplied directory. When it receives an empty directory argument(" ") it queries for the root directories the method returns asterisks(*) to designate the contents of an empty directory.

```
public String[] list_directory (String directory,String direction){
                String lst[];
                String directoryName = directory;
                if(direction.equals("up")){
                        File file = new File(directoryName);
                        try{
                                String parnt = new File(file.getParent()).getCanonicalPath();
                                File prFile = new File(parnt);
                                directoryName = new File(prFile.getParent()).getCanonicalPath();

                        }
                        catch(Exception e){
                        }
                }
                if(directoryName.equals(" ")){
                        System.out.println("called \n\n");
                        File[] rootFiles = File.listRoots();
                        lst = new String[rootFiles.length];
                        for(int i=0;i<rootFiles.length;i++){
                                try{
                                        lst[i] = rootFiles[i].getCanonicalPath();
                                        System.out.println(rootFiles[i]+" \n\n");
                                }
                                catch(Exception e){
                                        // ***** perror()!!!!
                                }
                        }
                }
                else{
                        String comment = ("received a dir request \t  " +directoryName);
                        log(comment);
                        System.out.println(comment);
                        lst = null;
                        File file = new File(directoryName);
                        if(file.isDirectory()){
                                String[] tmplst = file.list();
```

```
                                        lst = new String[tmplst.length];
                                        try{
                                                for(int i=0;i<tmplst.length;i++){
                                                File canPath = new File(file,tmplst[i]);
                                                        lst[i] = canPath.getCanonicalPath();
                                                }
                                        }
                                        catch(Exception e){
                                        }
                                }
                                else{
                                        lst = new String[1];
                                        lst[0] = "*";
                                }
                        }
                        return lst;
        }
```

How the bonet implements the download method

The method uses java.io library utility (bufferedReader) to read the contents of a local file (respect to the bonet) into a string the transmit this result to the requesting client, the requesting client then uses this string sequence as an input stream to build the requested file. Unfurtunately the method does not work for all character sets.
The method calls the copy_in method with the string stream argument (converted input stream) to complete the download request.

```
public void download(String filename, String dnldServer,String destination_host, String destination_dir){
                        System.out.println("\n\n");
                        String comment = (" \n Got download request from  " + destination_host + "\n :: for file
"+filename + "\n  ::destination "+ destination_dir+ "\n\n" );
                        System.out.println(comment);

                        log(comment);

                        int BLKSIZ = 8192;
                        args_holder args_source = new args_holder();
                        pimpHelper ph=new pimpHelper(args_source.args);
                        client remoteclient=ph.connectClient(destination_host);
                        String destinatioPath ;
                        File copy_file = new File(filename);
                        try{
                                if(copy_file.isFile()){
                                        destinatioPath = destination_dir+"\\"+copy_file.getName( );
                                        FileReader fr = new FileReader( copy_file );
                BufferedReader in = new BufferedReader(fr);
                        System.out.println("bufered reader created !");
                        StringBuffer sb = new StringBuffer( );
                                                char[] b = new char[BLKSIZ];
                                                int n;
                                                // Read a block. If it gets any chars, append them.
                                                while ((n = in.read(b)) > 0) {
                                                        System.out.println(n);
                                                        sb.append(b, 0, n);
                                                }
                                                // Only construct the String object once, here.
                                                String outFile = sb.toString( );
                                                //System.out.println(outFile);
```

```
                                                        remoteclient.copy_in(outFile,destinatioPath,dnldServer);
                                }
                        }
                        catch(Exception e){
                                System.out.println(e);
                        }
                }

public void copy_in (String input_stream, String path,String dnldServer){
                        String comment = ("got copy in command from ::  "+ dnldServer + "for :: " + path + "\n"
);
                        System.out.println(comment);
                        log(comment);
                        try{
                                OutputStream out = new FileOutputStream(path);
                                for(int i=0;i<input_stream.length();i++){
                                        out.write(input_stream.charAt(i));
                                }
                                out.close();
                        }
                        catch(Exception e){

                        }


                }
```

How the system implements the searchFile method

The searchFile method seaches the network clients (bonets) for files whose names match the search file (file to be searched). The monitor sends the search string to each and every connected client. The connected clients then queries all the directories in the system starting from the root and then sends back the details of the files that match the search string. the request is asynchronous i.e the monitor doesn't have to wait for the search result once it has send the request

Implementation:

```
public void searchFile (String fileName, String monitorName){
                        String comment = ("received search command for "+ fileName);

                        log(comment);
                        System.out.println(comment);

                        File[] drives = File.listRoots( ); // Get list of roots
                        searchThread[] searchs = new searchThread[drives.length];
                        try{
                                for (int i=0; i<drives.length; i++){
                                        searchs[i] = new
searchThread(drives[i].getCanonicalPath(),fileName,monitorName);
                                }
                        }
                        catch(Exception e)

                        }
```

```
        }

class matcher{
        public boolean match(String str1,String str2){
                boolean retVal = false;
                int s1_length = str1 length();
                int s2_length = str2.length();
                int margin = s2_length - s1_length;
                if(str1.length()>str2.length()){
                        retVal = false;
                }
                else{
                        int pointer = 0;
                        while((pointer+s1_length)<s2_length+1){
                                String subStr2 = str2.substring(pointer,pointer+str1.length());
                                if(str1.equalsIgnoreCase(subStr2)){
                                        retVal = true;
                                        break;
                                }
                                pointer++;
                        }
                }
                return retVal;
        }
}
```

How the system implements the bonets interface list_roots

The method when called queries the system for root directories. It makes us of the java file io library utilities to obtain the root directories of the system

Implementation:

```
public String[] list_roots (String hostname){
                String[] root_dirs = null;
                String comment = ("find root directories called by "+ hostname );
                log(comment);
                System.out.println(comment);
                        File[] rootFiles = File.listRoots();
                        root_dirs = new String[rootFiles.length];
                        for(int i=0;i<rootFiles.length;i++){
                                try{
                                        root_dirs[i] = rootFiles[i].getCanonicalPath();
                                        System.out.println(rootFiles[i]+" \n\n");
                                }
                                catch(Exception e){
                                        // perror();
                                }
                        }
                        return root_dirs;
        }
```

How the system implements the bonets interface listdir

The method lists the contets of the supplied directory the method also incoporates means by which the requesting clients determines whether the returned argument is a file or directory, enabled by use of the flags parameter

implementation:

```
public void listdir (String path, pc.clientPackage.argumentsHolder dirlist, pc.clientPackage.flagsHolder fl){
                String lst[];
                String directoryName = path;
                String comment = ("received a dir request \t  " +directoryName);
                System.out.println(comment);
                        File file = new File(directoryName);
                                String[] tmplst = file.list();
                                int[] flags = new int[tmplst.length];
                                lst = new String[tmplst.length];
                                try{
                                        for(int i=0;i<tmplst.length;i++){
                                                File canPath = new File(file,tmplst[i]);
                                                lst[i] = canPath.getCanonicalPath();
                                                if(canPath.isDirectory()){
                                                        flags[i] = 3;
                                                }
                                                else{
                                                        flags[i] = 4;
                                                }
                                        }
                                }
                                catch(Exception e){
                                }
                        dirlist.value = lst;
                        fl.value = flags;
        }
```

How the system implements the bonet's interface rename

The method is implemented by the bonet to provide ways in which network clients can rename a remote file. The method uses the java.io file library utilities to perform the operation. The method is enclosed in try catch block to capture any errors (file io errors).

Implementation:

```
public void rename (String oldName, String newName, String requestingHost){
                        String comment = (" :: received a rename request from :"+ requestingHost);
                        System.out.println(comment);
                        try{
                                File f = new File(oldName); // backup of this source file.
                                // Rename the backup file to "junk.dat"
                                // Renaming requires a File object for the target.
                                f.renameTo(new File(newName));
                        }
                        catch(Exception e){

                        }
        }
```

Sugested improvements:
The method performs the rename operations on remote hosts, security - access control list- needs to be implemented to check wheather the remote user has the previledges to perform the operation

How the system implements delete:

The method is implemented by the bonet to provide ways in which network clients can delete a remote file. The method uses the java.io file library utilities to perform the operation. The method is enclosed in try catch block to capture any errors (file io errors).

Implementation:

```
public void delete (String deleteFile, String requestingHost){
                String comment = (" :: received a rename delete from :"+ requestingHost);
                System.out.println(comment);
                try{
                        // Construct a File object for the file to be deleted.
                        File bkup = new File(deleteFile);
                        // Quick, now, delete it immediately:
                        if (!bkup.delete( )){
                                System.out.println("** Deleted " + deleteFile);
                                bkup.delete();
                        }
                        else
                        System.err.println("Failed to delete " + deleteFile);
                }
                catch (SecurityException e) {
                System.err.println("Unable to delete " + deleteFile +"(" + e.getMessage( ) + ")");
                }
                catch(Exception e){

                }
        }
```

Sugested improvements:
Like the rename method , security - access control list- needs to be implemented to check wheather the remote user has the previledges to perform the operation.

How the system implements getprops

The method getprops is implemnted by the bonet. It queries the supplied files in the system for their properties. These properties include the date the file was last modified, whether the fiile can be written to, whether it's a file or folder. The method makes use of mappings provided by corba to get the requested results using out parameters.

implementation:

```
        File f = new File(fileName);
                size.value = (int)f.length();
                lastModified.value = (int)f.lastModified();
                canread.value = f.canRead();
                canwrite.value = f.canWrite();
                isFile.value = f.isFile();
        }
```

How the system implements the bonet's synchronize:

The synchronize method, seeks to maintain consistencies between files in remote computers by synchronizing their contents based on the selecred criterion. The ctriterion used could be by time last modified or by their sizes. The most recent time means the most consistent. Likewise to size the largest means that content has been added which means it is the latest thus the most suitable. To perform the opration the bonet must query the properties of a remote file and make the comparison neccesarily needed to make the synch decision.

Implementation:

```
public void synchronize (String criteria, String hostA, String dirA, String hostB, String dirB){
                        String comment ;
                        boolean by_modified = false;
                        if ( criteria.equals(" By Time Last Modified "))
                                by_modified = true;
                        System.out.println("\n\n\n by_modified = " + by_modified + "\n\n\n");
                        System.out.println("------------------"+ criteria  + hostA  +dirA  + hostB  +dirB);

                        args_holder args_source = new args_holder();
                        pimpHelper ph=new pimpHelper(args_source.args);

                        client remoteclient=ph.connectClient(hostB);


                        String lst[]= null;
                        File file = new File(dirA);
                        System.out.println("\n\n\n\n ---------------- dirA  "+ dirA+ "\n\n\n\n");
                        String[] tmplst;
                        int[] sizeA = null;
                        boolean[] is_fileA = null ;
                        int[] modifiedA = null;
                        if(file.isDirectory()){
                                tmplst = file.list();
                                sizeA = new int[tmplst.length];
                                is_fileA = new boolean[tmplst.length];
                                modifiedA = new int[tmplst.length];
                                lst = new String[tmplst.length];
                                try{
                                        for(int i=0;i<tmplst.length;i++){
                                                File dir_list = new File(file,tmplst[i]);
                                                lst[i] = dir_list.getCanonicalPath();
                                                if(dir_list.isDirectory()){
                                                        is_fileA[i] = false;
                                                }
                                                else{
                                                        is_fileA[i] = true;
                                                }
                                                sizeA[i] = (int)dir_list.length();
                                                modifiedA[i] = (int)dir_list.lastModified();
                                                is_fileA[i] = dir_list.isFile();
                                        }
                                }
                                catch(Exception e){
                    }
            }
            else{
                    try{
                            int s = (int)file.length();
                    sizeA = new int[1];sizeA[0] = s;
                    boolean b = file.isFile();
                    is_fileA = new boolean[1]; is_fileA[0] = b;
                    int m = (int)file.lastModified();
                    modifiedA = new int[1]; modifiedA[0] = m;
                    String str = file.getName();
                    lst = new String[1] ; lst[0] = str;
                    }
                    catch(Exception e){
                            System.out.println(e);
                    }
            }


            pc.clientPackage.flagsHolder size = new pc.clientPackage.flagsHolder();
            pc.clientPackage.booleanseqsHolder is_file = new
pc.clientPackage.booleanseqsHolder();
            pc.clientPackage.flagsHolder modified = new pc.clientPackage.flagsHolder();
            pc.clientPackage.argumentsHolder ls = new
pc.clientPackage.argumentsHolder();
            pc.clientPackage.argumentsHolder canonical = new
pc.clientPackage.argumentsHolder();
            remoteclient.queryFolder (hostA, dirB, size, is_file, modified, ls, canonical);
            System.out.println("ok !! called querty" );
            for(int i =0;i<ls.value.length;i++){
                    String fileN = ls.value[i];
                    if(contains(fileN,lst)!= -1){
                            //check criteria
                            int k = contains(fileN,lst);
                            if(by_modified){
                                    //by date modified
                                    if(modified.value[i] >modifiedA[k]){
                                            System.out.println("modified remote :: " +
modified.value[i] + "modified local ::" + modifiedA[k] );
                                            //download if file or Mkdir() and
synchronize iteratively;
                                            if(is_file.value[i]){
                                                    //---------download
                                                    File par = new File(dirA);
                                                    String destn = par.getParent();

        remoteclient.download(canonical.value[i], hostB,hostA,destn);
                                            }
                                            else{
                                                    boolean created = new
File(dirA,ls.value[i]).mkdirs();
                                                    String canP = dirA + ls.value[i];
                                                    // call synchronize
                                                    pc.clientPackage.flagsHolder size2
= new pc.clientPackage.flagsHolder();
```

```
        pc.clientPackage.booleanseqsHolder is_file2 = new pc.clientPackage.booleanseqsHolder();
                                                              pc.clientPackage.flagsHolder
 modified2 = new pc.clientPackage.flagsHolder();
                                                              pc.clientPackage.argumentsHolder
 ls2 = new pc.clientPackage.argumentsHolder();
                                                              remoteclient.synchronize(criteria,
 hostA, canP, hostB, canonical.value[i]);
                                                         }
                                                  }
                                           }
                                           else{
                                                  //by size
                                                  if(size.value[i] >sizeA[k]){
                                                         //download if file or Mkdir() and
 synchronize iteratively;
                                                         if(is_file.value[i]){
                                                              //---------download
                                                              File par = new File(dirA);
                                                              String destn = par.getParent();

        remoteclient.download(canonical.value[i], hostB,hostA,destn);
                                                         }
                                                         else{
                                                              boolean created = new
File(dirA,ls.value[i]).mkdirs();
                                                              String canP = dirA + ls.value[i];
                                                              // call synchronize
                                                              pc.clientPackage.flagsHolder size2
= new pc.clientPackage.flagsHolder();

        pc.clientPackage.booleanseqsHolder is_file2 = new pc.clientPackage.booleanseqsHolder();
                                                              pc.clientPackage.flagsHolder
modified2 = new pc.clientPackage.flagsHolder();
                                                              pc.clientPackage.argumentsHolder
ls2 = new pc.clientPackage.argumentsHolder();
                                                              remoteclient.synchronize(criteria,
hostA, canP, hostB, canonical.value[i]);
                                                         }
                                                  }
                                           }
                                    }
                                    else{
                                           //-------download
                                           if(is_file.value[i]){
                                                  //---------download
                                                  System.out.println("no match found");
                                                  File par = new File(dirA);
                                                  String destn = par.getParent();
                                                  remoteclient.download(canonical.value[i],
hostB,hostA,destn);
                                           }
                                           else{
                                                  boolean created = new File(dirA,ls.value[i]).mkdirs();
                                                  String canP = dirA + ls.value[i];
                                                  // call synchronize
```

```
                    pc.clientPackage.flagsHolder size2 = new
pc.clientPackage.flagsHolder();
                    pc.clientPackage.booleanseqsHolder is_file2 = new
pc.clientPackage.booleanseqsHolder();
                    pc.clientPackage.flagsHolder modified2 = new
pc.clientPackage.flagsHolder();
                    pc.clientPackage.argumentsHolder ls2 = new
pc.clientPackage.argumentsHolder();
                    remoteclient.synchronize(criteria, hostA, canP
hostB, canonical.value[i]);
                }
            }
        }
    }
```

How the system implements the bonet's queryFolder:

The method is used by the bonet when performing synchronization, the bonets uses the method to query the properties of a remote file needed to perform the synchronization operation.

Implementation:

```
public void queryFolder (String queryHost, String dirName, pc.clientPackage.flagsHolder size,
pc.clientPackage.booleanseqsHolder isFile, pc.clientPackage.flagsHolder modified,
pc.clientPackage.argumentsHolder ls, pc.clientPackage.argumentsHolder canonical){
        String lst[] = null;
        File file = new File(dirName);

            int[] sizeA = null;
            boolean[] is_fileA = null ;
            int[] modifiedA = null;
            String[] canP = null;
            if(file.isDirectory()){
                String[] tmplst = file.list();
                sizeA = new int[tmplst.length];
                is_fileA = new boolean[tmplst.length];
                modifiedA = new int[tmplst.length];
                lst = new String[tmplst.length];
                try{
                    for(int i=0;i<tmplst.length;i++){
                        File dir_list = new File(file,tmplst[i]);
                        canP[i] = dir_list.getCanonicalPath();
                        lst[i] = dir_list.getName();
                        if(dir_list.isDirectory()){
                            is_fileA[i] = false;
                        }
                        else{
                            is_fileA[i] = true;
                        }
                        sizeA[i] = (int)dir_list.length();
                        modifiedA[i] = (int)dir_list.lastModified(),
                        is_fileA[i] = dir_list.isFile();
                    }
                }
```

```
			catch(Exception e){
				System.out.println(e);
			}
		}
		else{
			try{
				int s = (int)file.length();
				sizeA = new int[1];sizeA[0] = s;
				boolean b = file.isFile();
				is_fileA = new boolean[1]; is_fileA[0] = b;
				int m = (int)file.lastModified();
				modifiedA = new int[1]; modifiedA[0] = m;//modifiedA =
{m};

				String str = file.getName();
				lst = new String[1] ; lst[0] = str;
				String can = file.getCanonicalPath();
				canP = new String[1] ; canP[0] = can;
			}
			catch(Exception e){
				System.out.println(e);
			}
		}
			size.value = sizeA;
			isFile.value = is_fileA;
			modified.value = modifiedA;
			ls.value = lst;
			canonical.value = canP;
			//size.value(sizeA)}
```