A FRAMEWORK FOR DEPLOYMENT OF MOBILE AGENTS AS WINDOWS OPERATING SYSTEM SERVICE FOR INFORMATION RETRIEVAL IN DISTRIBUTED ENVIRONMENTS

BY

BOSEDE OYENIKE OYATOKUN
113798

B.Tech. (Hons), Computer Engineering (LAUTECH), M. Sc. Computer Science (Ibadan).

A Thesis in the Department of Computer Science

Submitted to the Faculty of Science
In Partial Fulfilment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

of the

UNIVERSITY OF IBADAN

DECEMBER, 2013

CERTIFICATION

This is to certify that this res earch work was carried out by BOSEDE OYENIKE OYATOKUN with matriculation number 113798 in the Department of Computer Science, Faculty of Science, University of Ibadan, Ibadan.

Prof. Adenike O. Osofisan
B. Sc (Ife), M. Sc (Georgia Tech.), Ph D (Ife) Computer Science
Professor, Department of Computer Science,
University of Ibadan, Ibadan

Prof. G. A. Aderounmu
B. Sc (Hons), M Sc, Ph D Computer Science (OAU, Ife)
Professor, Department of Computer Science and Engineering
Obafemi Awolowo University, Ile-Ife

DEDICATION

I dedicate this thesis to GOD ALMIGHTY, my source and inspiration

The memory of my parents Pa Joseph Ogunjumo Oyatokun and Madam Nihinlola Amope Oyatokun

And

My 'boys', My Prince, Oluwatimilehin, MoyosoreOluwa

ACKNOWLEDGEMENTS

All glory, honour, praise and majesty to the Lord Almighty through Jesus Christ, for His love, mercies and favour that I receive each day. Father, you are wonderful, without you there is no me, I am exceedingly grateful.

My profound gratitude goes to my supervisor, Prof. Adenike O. Osofisan for her guidance, support and motherly counsel. She made me see possibilities where others have seen impossibility; thank you so much ma, the good Lord will do you good. I am indebted to Prof. G.A Aderounmu (OAU), for his persistent zeal to see that the programme is completed. He devoted his time to attend to me even when not notified before hand, his comments, suggestions and corrections all added to enrich the value of this work, thank you so much sir, God will perfect all concerning you.

I sincerely appreciate the Head of Department of Computer Science, University of Ibadan, Dr. B. A. Akinkunmi for his counsel and advice. I also appreciate the contributions of Dr. A.B.C. Robert for his comments and provision of literature materials and Dr. S.O. Akinola for his untiring support at the verge of completion of the work. I express my gratitude also to Dr. O.F.W Onifade for his support; he read the first draft of the thesis and made necessary comments. I am sincerely grateful to the entire members of staff of the department, you have contributed in no small measure to the success of this work, and God bless you.

I appreciate the Deans of Science and Heads of Department, staff and student of Mathematical Sciences, Redeemer's University and Olabisi Onabanjo University. I sincerely appreciate Prof. T. Ogunsanwo, who believed in me and spoke in my favour before I was given the admission, thanks a lot sir, God bless.

My profound gratitude goes to the entire members of Oyatokun's family, who have been there for me and have been supportive spiritually, financially and morally, God bless you all. I also wish to express my appreciation to my friends and colleagues, OreOluwayinka, Dr T.O. Olatayo, Mrs M.D Okewole, Mr. M.O. Odim, Dr S.A. Arekete and many others I could not mention, thanks a million.

My special thanks to my husband, '*my Prince*', you are indeed inestimable and my lovely children, Oluwatimilehin and MoyosoreOluwa, for their love, support, understanding, cooperation and encouragement, which have contributed to making this

work a success, God bless you, make you exceedingly great and perfect all that concern you.

TABLE OF CONTENTS

IMPLEMENTATION AND SIMULATION OF EMBEDDED MOBILE AGENT

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Mobile Agent Technology (MAT), remote method invocation and remote procedure calling are the three most widely used techniques for information storage and retrieval in network environments. Previous studies have shown that MAT provides a more efficient and dynamic approach to information storage and retrieval than others. However, for mobile agents to effectively perform their various tasks, a static agent platform must be installed on the computers. These platforms consume more memory, increase access time and prevent other tasks from running on the computer. Therefore, an alternative framework that will eliminate the problems associated with agent platform is imperative. Consequently, this work was aimed at developing a more efficient framework for mobile agent system deployment as an operating system service.

Two classes of existing information retrieval agents were adapted to develop Embedded Mobile Agent (EMA) system. The EMA was embedded into the Windows Operating System (OS) kernel, so that it could run as a service for information retrieval. This was done to eliminate the overheads associated with the middleware provided by agent platforms. The targeted OS were Windows XP, Windows Vista and Windows7. Mathematical models were simulated to assess the performance of EMA by measuring service delay, memory utilisation, fault tolerance, turn around time at fixed bandwidth with varying number of network nodes, and percentage denial of service. Denied services were generated by a random number generator modelled after the Bernoulli Random Variable with 0.1 probability of failure. The model's performance was then compared with Java Agent DEvelopment framework (JADE), a widely used open-source existing mobile agent system running on platforms. The implementation was done using four computer systems running the targeted Windows on an existing local area network. Analysis of data was done using descriptive statistics and independent t-test at $p = 0.01$.

The EMA model effectively retrieved information from the network without the agent platform, thereby reducing access times and saving memory, regardless of the version of the Windows OS. The mean service delay for EMA (15067.5 ± 8489.6 ms) was lower than that of JADE (15697.0 ± 8844.5 ms). The embedded agent requires 3 KB of

memory to run compared to JADE platform requiring 2.83× 10^3 KB. The mean fault tolerance in terms of fault recovery time for EMA was approximately 50% that of JADE (327.8 ± 193.1 ms). The mean turn around time for EMA was 499.7 ± 173.0 ms and JADE was 843.3 ± 321.6 ms consequential to the time JADE spent activating platforms. The mean percentage denial of service for EMA was 14.3 ± 9.8 while JADE was 24.7 ± 18.5. Memory requirements and service delay increased with increasing number of nodes while others show no systematic change. For all the parameters tested, there were significant differences between the two schemes.

The embedded mobile agent provided more efficient, dynamic and flexible solution compared to Java Agent DEvelopment framework for distributed information retrieval applications. It could be incorporated into new versions of operating systems as operating system service for universal distributed information retrieval.

Keywords: Mobile agent technology, Embedded mobile agent, Operating system service, Java agent development framework.

Word count: 497

CHAPTER ONE

INTRODUCTION

1.0 Background of the Study

Information is an essential component of any establishment; their survival depends largely on information production, dissemination, consumption and sharing. Information technology has made it possible to access and use information from different sources regardless of their physical location and this is achieved by the increased penetration and use of the Internet (Dale and DeRoure, 1997). Technological advancements bring with it a technological change in businesses and system operations, and information access requirements have given rise to an environment where computers work together to form a network (Seng, 1999).

Computer network is a set of interconnected autonomous computer systems that allows computing resources to be shared (Ferouzan and Fegan, 2007). Shared resources could be data, files, and services like database systems, e-mail services or hardware like printer, modem and so on. The use of computers to form networks results into distributed systems. A distributed system is a computing facility built with many computers that operate concurrently, are physically distributed, have their own failure nodes, have independent clock and are linked by a network (Ashvin, 2004). Adewunmi (2002) described distributed system as a generalised transparent system consisting of

collection of sites connected by a communication network, over which processing logic and/or data is shared based on usage. In distributed environments, each computer is capable of communicating with the others, programs running on each computer can share information and request task to be executed. Some of the objectives of distributed systems are to connect users in open and scalable manner, to make resources easily accessible and reasonably hide the fact that resources are distributed across network (Tanenbaum and Steen, 2007). Distributed systems increase reliability and availability of services, resources can be shared by all computers, increase the speed of processing compared to the old centralised system, reduce communication costs, and have high processing capacity and are expandable (Tanenbaum and Steen, 2007). Distributed systems assume a static configuration of execution environment, and various distributed applications running on a node are bound to such node. This assumption was confronted by developments that introduced mobility in the distributed systems. This form of mobility dynamically changes the locations of the components of an application; this is often called code mobility (Picco *et al.*, 2001). Code mobility is a concept that involves communicating entities in a mobile code systems exchange programs instead of simply data (Fuggetta *et al.*, 1998). The essence of code mobility is to transport some resource-accessing mobile code units from one host to another, and then execute these code units on the resource-bearing host (Fong, 2003). Distributed operating systems employ process migration mechanisms, allowing an operating system process to move from one machine to another and resume execution, the migration is transparent. Code mobility offers a lot of advantages to distributed systems, among the numerous advantages are: support for deployment and upgrade of distributed applications, services on the systems can be customized, it improves the robustness of the system to failure and provides support for disconnected operations (Fuggetta *et al.*, 1998). In distributed systems, services can be executed when all the elements required are located at the same host, i.e. resources, know-how or procedure and computational capability are on the same host. The most widely used paradigms for distributed applications are client/server, remote evaluation and mobile agent paradigms (Vitek, 1997; Lange, 1998; Neeran and Anand 1998; Bellavista *et al.;* 2000 and Bellavista *et al.*, 2001).

The client/server paradigm dominated the scene for a long time (Seng, 1999; Braun and Rossak, 2005). In this arrangement, there is a server that contains resources and it is willing to share the resources with others that may need them and a client that requests

resources or services (Lange, 1998). Each request/response has to be a complete round trip on the network. Several technologies supported the client server paradigm, such as the Object Request Broker's CORBA, Remote Procedure Call (RPC) and Remote Method Invocation (RMI) (Lange, 1998). According to Aderounmu *et al*. (2006), the last two technologies have particularly improved this paradigm. In RPC, the client calls a remote procedure over a socket connection using protocol known to both parties, as if it were a local procedure. RMI is an object oriented version of RPC, in this case, clients invoke remote objects of the server as if it were a local object running in the same virtual machine (Oyatokun, 2004).

Remote Evaluation Paradigm (REV) was proposed as an alternative to client-server and was first introduced as a concept by Stamos and Gifford in 1990 (Neeran and Anand, 1998; Bohoris, 2003). The client in REV has the necessary procedure but lacks the processing capability and resources. The remote evaluation involves transmission of the code containing the required logic with initial parameters from the client computer to the server for execution and the result of the execution is returned to the client computer (Bohoris, 2003).

Another model is the Code on Demand paradigm (CoD), in which the client has the processing capability and local resources but does not have the procedure to access and process the resource (Lange, 1998). The procedure is obtained from the server when needed and execution begins on the client as soon as it receives the procedure. CoD has not received so much attention in recent times because of the non-availability of code-servers.

Further steps from the client-server introduced mobile agent technology from the software agents in the field of artificial intelligence. Mobile agent paradigm consists of a named object that migrates through the network with its code, data and authority of its owner (Wenjuan *et al*., 2009). Mobile agents once dispatched from their origin, are detached from the origin and can make multiple hops before returning to the origin with the results of the computation (Lange, 1998).

The client/server paradigm relies so much on the network connections between the server and client, in which case the network connection has to be maintained throughout the period of communications. Whereas, technology has evolved over the years to favour connectionless systems, the introduction of wireless systems and mobile computing e.g. the mobile phones, PDA and other handheld devices do not favour connection oriented system. In this part of the world, network connection is

unreliable and inconsistent, these and the fact that distributed systems have increased in complexity and size do not favour client/server paradigm. Mobile agent paradigm was proposed as an alternative to client server paradigm for distributed applications (Aderounmu, 2001; Seng, 1999; Dale and DeRoure, 1997). It offers flexibility on the reliance on network connection (Aderounmu, 2003), once launched, can be disconnected, it keeps performing its tasks and can be reconnected to receive the result at a later time (Dale and DeRoure, 1997). Mobile agent is an object that migrates through many nodes of a heterogeneous network of computers under its own control in order to perform tasks using resources of these nodes (Roberto, 2001). Biermann (2004) defines mobile agents as autonomous software capable of performing computational tasks on behalf of another software or human user. In the distributed system research community, it is widely accepted that, mobile agent is a named object that has code, persistent state, data and a set of attributes (e.g movement history, authentication keys) and can move or transport itself from one host to another as needed for accomplishing its tasks (Lange, 1998; Roberto, 2001). Outtagarts (2009) defines mobile agent as a computer entity capable of reasoning, use the network infrastructure to run in another remote site, search and gather the results, cooperate with other sites and return to its home site after completing the assigned tasks. Mobile agents paradigm provides infrastructure for executing automous agents and also migrate them between computers connected by a network. Mobile agents have been defined differently by different researchers, but all have agreed that mobile agents have code, certain data and can move from node to node using an existing network infrastructure.

Mobile agents offer an improved performance in managing distributed systems, facilitate access to multiple information sources and facilitate scalability (Seng, 1999; Finin and Nicholas, 2000; Aderounmu, 2001; and Roberto, 2001). Mobile agents have the potentials to improve the speed and efficiency of computation by moving computation to data, thus eliminating unnecessary and massive data transfer over the network. According to Sridhar and Vikram (2001), they are viable tools when information needed is vast and widely distributed, and in applications or services that need to learn and improve over time.

Mobile agent paradigm has been recognized as a viable tool and a promising approach for building distributed applications (Aderounmu, 2001; Bellavista *et al* 2001; Aderounmu *et al*, 2006; Stoian and Popirlan, 2010). Agents solve complex software

problems in distributed environments where protocols, operating systems, hardware and runtime environments are heterogeneous. Mobile agent has been successfully applied to many distributed applications such as information retrieval and management, electronic commerce, network management, supply chain management and a lot more (Outtagarts, 2009). Meanwhile, there are some major setbacks affecting mobile agent that have prevented it from being widely employed. These include; security, complexity and lack of standard (Fortino and Russo, 2003, Tudor *et al*., 2004). The complexity and sophistication naturally led to several attempts to simplify and extend agents' functionality, thus attention shifted to enhancing specific aspects of agents and providing necessary security for mobile agents, agent platforms and hosts on which they execute. The versatility of mobile agent paradigm also increased research interest in enhancing mobile agents in the area of agent communication and agents' structure so as to extend their functionalities. However, mobile agents operate only on computers with the agent platform previously installed, which consumes more memory, increases access time and prevents other tasks from using the visited computer. This study focuses more on developing an approach that eliminates the use of agent platforms and make agents interact directly with the operating system on the host computer, to eliminate the overheads associated with agent platforms.

1.1 Rationale for the Research

The explosion in the information available in widely dispersed locations through the use of Internet and the exponential increase in the number of users require an efficient and rapid way to store and retrieve the information (Dale and DeRoure, 1997). Mobile agent technology has been adopted to store and retrieve information efficiently and quickly from widely dispersed users (Clark and Lazarou, 1997; Htoon and Thwin 2008). The existing mobile agents require that an agent platform be installed on the computer on which they are expected to run. These platforms consume memory, increase access times, prevent user from performing other tasks on the computer and the mobile agents are limited in operations to the platforms on which they are bound to operate. There are a number of these platforms; unfortunately, these platforms are not usually interoperable with one another, in the sense that, mobile agent built on one platform cannot operate on another platform (Pinsdorf and Roth, 2002; Zeghache *et al*,

2002; Grimstrup *et al*, 2002). Thus, the flexibility, scalability and interoperability expected of mobile agents in distributed environment are challenged. This necessitates a new way of deploying mobile agents that could perform their tasks without going through agent platform. To reduce the heavy reliance of mobile agents on agent platforms installed on the computers, mobile agents can be made to interact directly with the Operating System, since all computers run an Operating System.

The direct interaction of mobile agents with the Operating System has several advantages compared to the existing ones interacting with agent platforms. These include:

- Reducing the memory requirements for the entire system by eliminating installation of mobile agent platforms which resides in the memory
- Reducing the access time for mobile agents to perform their tasks via direct access to the Operating Systems and eliminating the time required to activate agent platform.
- Reducing the number of denied services in the overall system, since Operating System runs continuously, operating system services are privileged programs that could run in real time.
- Increasing the fault tolerance capability and robustness of the system, taking advantage of the auto-safe and auto-recovery facilities provided by the Windows operating System.
- However, the computer system through the Operating System is vulnerable to malicious agents' attack, thus requiring additional security component at an additional cost.

Based on the above advantages that outweigh the disadvantages, this work proposed an embedded mobile agent for information retrieval.

1.2 Statement of the Problem

Major challenges in information retrieval are efficiency and effectiveness of retrieval, in other words, retrieving relevant documents in real time and with minimum cost possible. Development of a distributed information retrieval system requires more efficient technologies to ensure real time retrieval of relevant information. Features

such as efficiency, effectiveness, precision and recall should be related to all retrieval with distributed information retrieval systems.

To deal with the ever increasing amount of information available on the Internet, mobile agent technology has proved to be effective (Clark and Lazarou, 1997). However the complexity of mobile agent technology raises certain issues on scalability, reusability, interoperability and fault tolerance. The restriction placed on mobile agent by the agent platform that must be previously installed on the computer before mobile agent could execute is a great challenge.

To deal with the problem of lack of interoperability, the embedded mobile agent offers the possibility of mobile agents interacting directly with the Operating Systems on the computers they are to run. The existing mobile agents require agent platforms to be previously installed in the computers on which they are to run; this agent platform needs to be explicitly initiated before receiving and providing runtime execution for incoming mobile agents and could executes only mobile agents designed specifically for it.

Embedded mobile agent takes advantage of the fact that all computers run an operating system and attempts to make agents part of the operating system. The kernel mode of the operating system (specifically Windows Operating System) is extended. The focus of this research is to eliminate the static agent previously installed in memory and make mobile agents interact directly with the operating system. To achieve this, a lightweight static agent is made available to run in the kernel mode as part of Windows O/S in form of operating system service.

1.3 Research Aim and Objectives

The aim of this research is to formulate a robust framework for mobile agent system deployment as an operating system service for information storage and retrieval capable of accessing heterogeneous information in a distributed environment.

The specific objectives are to:

a) formulate an enhanced architectural model for mobile agents that can be embedded into the kernel mode of windows operating system
b) deploy the mobile agent designed in (a) through the operating system without passing through the agent platform

c) develop a model for information retrieval with a view to specifying performance parameters
d) develop an information retrieval performance model and measurement factors based on (c) above.
e) simulate the performance measurement factors based on (c) and (d) above.

1.4 Research Methodology

This study was executed in phases and the objectives are achieved through stages. Several techniques were combined to achieve the set objectives, among which are literature review, design, implementation, experimentation and evaluation of the proposed embedded mobile agent. They include:

a) An extensive review of existing models for information storage and retrieval and mobile agent technology.
b) Developing an enhanced mobile agent model for information storage and retrieval.
c) Specification of the various performance parameters such as:
    - Memory utilization
    - Service delay
    - Fault tolerance
    - Denial of service
    - Turn around time.
d) Simulation and program development to implement (b) and (c) above using object oriented programming language specifically Java.
e) Performance evaluation of the developed system with an existing system was carried out.

1.5 Scope and Limitation

This work covers the design and implementation of embedded mobile agent for distributed information retrieval. A static agent was provided and embedded in the kernel mode of the operating system, as an operating system service, the targeted Operating Systems was Windows XP and was extended to Windows Vista and

Windows 7. The popularity of Windows Operating system in this part of the world and the opportunity Windows provide for programmer to make software available in its kernel mode motivate the choice of Windows. An attempt is made to extend the services of Operating System at the kernel mode and not to program the operating system. This work however does not cover security of operating system, and the implementation is limited to a local area network.

1.6 Arrangement of the Thesis

The rest of the thesis is arranged as follows: Chapter two contains an extensive and state-of-the-art review of relevant literature which includes the theory of mobile agent, existing systems of mobile agents for information retrieval, existing agent platforms. The theoretical aspect of information retrieval and different methods of retreival as well as work related to this topic are reviewed. The design of the proposed system is presented in chapter three, which include the architecture of the proposed system, the structure of the enhanced agent as well components interactions. The implementation of proposed system is presented in chapter four, a demonstration of the functionality of the proposed system is also presented. Chapter four also presents the performance evaluation of the proposed system against that of an existing system, the results of the performance are presented in graphical form for the simulated criteria, which are service delay, memory utilisation, denial of service, fault tolerance, turn around time and is followed by discussion of results. Chapter five concludes the work, summarizes the main discussions and contribution of the work and makes suggestions for further research.

CHAPTER TWO

LITERATURE REVIEW

2.0 Introduction

This chapter presents the background concepts of the work and state of the art review of literature on information retrieval, software agents and theoretical aspects of mobile agents. These include a survey of the existing agent systems for information retrieval, existing agent platforms, and some deficiencies in the existing systems that motivated the need for this study. This chapter is aimed at describing information retrieval systems using the mobile agent paradigm and introduce the central focus of this research.

2.1 Overview of Information Technology

Information technology (IT) has become an important and famous aspect of study in today's world and is increasingly moving to the core of national competitiveness strategies around the world. IT is the field of study that deals with the use of computers

and telecommunications equipments to store, retrieve, transmit and manipulate data (Daintith, 2009). The Information Technology Association of America has defined information technology as the study, design, development, application, implementation, support or management of computer-based information systems, particularly software applications and computer hardware (Proctor, 2011). According to Margaret (2005), IT is a term that encompasses all forms of technology used to create, store, exchange and use information in its various forms (business data, voice conversations, still images, motion pictures, multimedia presentations, telephony and computer technology). Meanwhile, information is any meaningful data or message content (Sullins, 2012) and it forms the fulcrum of any establishments. The survival of any establishments depends largely on information production, dissemination, consumption and sharing. IT comprises of anything related to computing technology, such as networking, hardware, software, the Internet or people that work with these technologies. Technology is dynamic, it advances with time, the technological advancements bring with it a technological change in businesses and system operations, and information access requirements have given rise to an environment where computers work together to form a network (Seng, 1999). In the same vein, information technology provides access to information from different sources regardless of their physical location and this is achieved by the increased penetration and use of the Internet (Seng, 1999). As a consequence, information sharing and dissemination are now on a global scale, made possible by the interconnection of several computers and computer networks.

2.2 Computer Network

Computer network is the interconnection of two or more autonomous computers that allows sharing of computing resources. The shared resources could be data, printer, modem or services such as e-mail, database systems (Ferouzan and Fegan, 2007). Networking essentials defines network as a group of computers that are wired together in some fashion which enables sharing of information and services (Shinder, 2001). Networking is the concept of connecting group of autonomous computers together with the aim of sharing resources and services and interacting through a shared communications link (Peter, 2012). The major goals of computer network are:

i. Resource Sharing: resources such as programs, data and hardware peripherals are made available to users on the network regardless of the location of the resources and the user.

ii. High Reliability: computer network provides high reliability by providing alternative sources of supply, there could be multiple copies of resources, both hardware and software resources, such that if one is not available, the other copies could be available.

iii. Cost Reduction: small computers have a better price/performance ratio compared to larger ones. Personal computers cost significantly lower than large mainframes, and a connection of few of them will achieve and even exceed the capability of large and expensive mainframe computers. Moreover, several resources on the network are shared, limited number of these resources is needed and cost of procuring more is thus saved.

iv. Communication Medium: computer network provides a powerful communication medium. An update on a file or database on the network can be seen by other users on the network immediately.

v. Improved Performance: the performance of a computer network can be improved as work load increases by adding one or more processors at a relatively low cost.

2.2.1 Classification of Computer Networks

Computer networks can be classified according to range or distance covered, transmission technology, functional relationship and network topology. This section examines the classes of computer networks based on range and topology.

Range of networks classifies network based on distances covered and the network physical sizes. These are:

i. Local Area Network (LAN): consists of a small group of computers and communication devices interconnected within limited geographical area such as a building or a campus not more than one kilometre (Tanenbaum, 2003). LAN uses digital transmission and transmits bits serially rather than in parallel. LAN is owned, controlled and managed by the organisation running it. LAN differs from other classes of network in their size which is restricted, transmission

technology that consists of a cable to which all the machines are connected, mainly Ethernet and token ring (Tanenbaum, 2003). A LAN is a switching system that

a. employs digital rather than analogue transmission
b. transmits bits serially rather than in parallel
c. employs typical clock rates of one to twenty billion per seconds
d. is relatively noise free compared to analogue voice communication lines, bit error rates of one in one billion are typical; and
e. switches packets or frames of bits, rather than holding transmission bandwidth for the duration of a communication session (Stuck and Arthurs, 1985)

ii. Metropolitan Area Network (MAN): connects two or more LANs usually within a city, using a high-capacity backbone technology such as fibre optical links. MANs are characterized by very high-speed connections using fibre optic cable or other digital media. Its communication links and equipments are owned by either a consortium of users or a network service provider who sells services to users. A well known example is the cable television (Tanenbaum, 2003).

iii. Wide Area Network (WAN): is a group of interconnected LANs over a large geographical area, often a country and continent (Tanenbaum, 2003). WAN exists in an unlimited geographical area; the machines are connected by a communication subnet. The subnet is made-up of two separate elements: transmission lines that move bits between machines and switching elements which are specialized computers connecting a number of transmission lines. Internet is the largest WAN known, spanning the entire globe. The machines are owned by the customers and the communication subnet is owned and operated by an Internet Service Provider (ISP). It uses technologies like Asynchronous Transfer Mode (ATM), frame relay for connectivity over long distances and router to connect LANs to WAN. Other technologies include SONET (Synchronous Optical NETwork) and SDH (Synchronous Digiatal Hierarchy) that provide high speed communication over fibre-optic networks and leased lines for point to point link from the communication subnet to the customers.

2.2.3 Computer Network Topology

Computer network topology is the logical arrangement of nodes, cables and connectivity devices that make up the network. The most common network topologies are bus, ring, star, mesh and tree topologies, this section briefly discuss these topologies.

(i) Bus Topology: all nodes and devices are connected to a common shared cable (called the backbone) as shown in Figure 2.1. Bus network broadcasts signals in both directions on the backbone cable, enabling all devices to directly receive the signal. Bus is cheap, easy to handle and implement, requires less cable, and avoids data collision since one computer transmits at a time and suitable for small networks. It uses Carrier Sense Multiple Access with Collision Detection (CSMA/CD) technology to avoid collision which also limits the size of the network to 2,500 meters (Shinder, 2001). However, the cable length is limited which limits the number of nodes that can be connected, it only performs well with a small number of nodes and fault diagnosis and fault isolation are difficult. A bus is a passive topology because the computers do not regenerate the signal and pass it on as it is done in ring, thus weakening the signal strength over distance and this is called attenuation. Furthermore, if there is a break in the cable, the computers on the opposite sides of the break cannot communicate and two new ends are not terminated and the resulting signal bounce can bring the entire network down (Shinder, 2001).

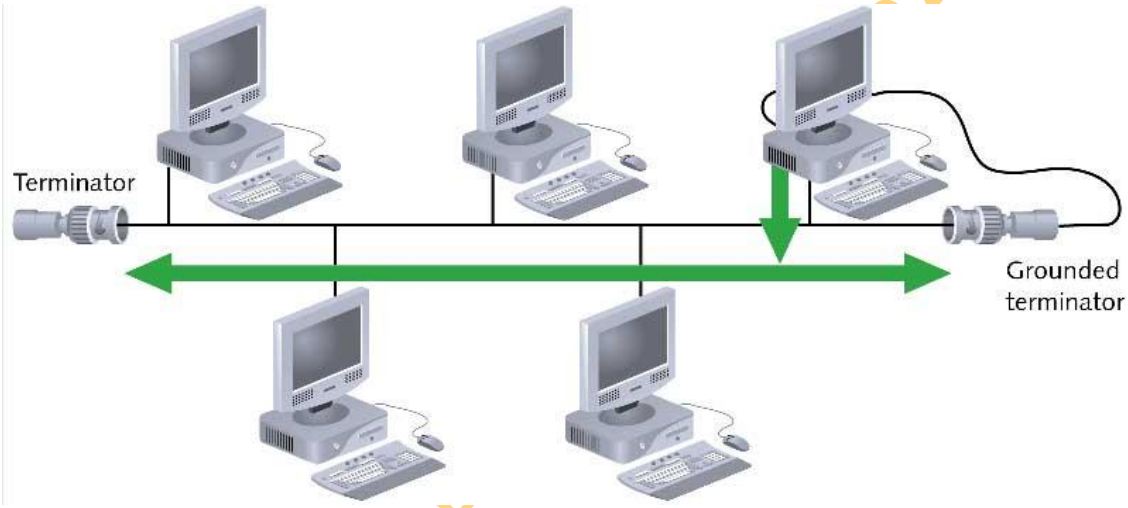


Figure 2.1: Bus Topology (adapted from Feyadat, 2008)

(ii) Ring Topology: each node is connected to its neighbour on both sides as shown in Figure 2.2. Ring network is implemented using token ring technology, in which a small data packet (token) is passed around the network continuously and any device willing to transmit reserves the token for the next trip, then attaches its data packet to the token. Ring topology presents an orderly network, only the node holding the token can transmit data, and it is easier to manage than bus network. Ring network however has a single point of failure, and any change made to the network nodes affects the performance of the entire network. Fault detection and fault isolation are difficult in ring network. In addition, it is difficult to add more computers to a ring network, because the cabling runs in a closed circle, it is necessary to break the ring at some point to insert the new computers. This means the network is out of commission while the addition is made (Shinder, 2001).

(iii) Star Topology: all devices are connected to a central hub which acts as a router to transmit messages. The star network is easily extendable, fault diagnosis and fault isolation is easy, failure of one station does not affect any other or the performance of the network and it is fast. Star topology is more fault tolerant than bus and ring, if one computer is disconnected, only that computer is affected, and the rest of the network can communicate normally. Star topology also offers ease of reconfiguration, adding more computers to the network or removing computers is as simple as plugging in and unplugging a cable (Shinder, 2001). However, star network as depicted in Figure 2.3 requires more cable and additional cost of the central hub and if the hub fails, the entire network fails.

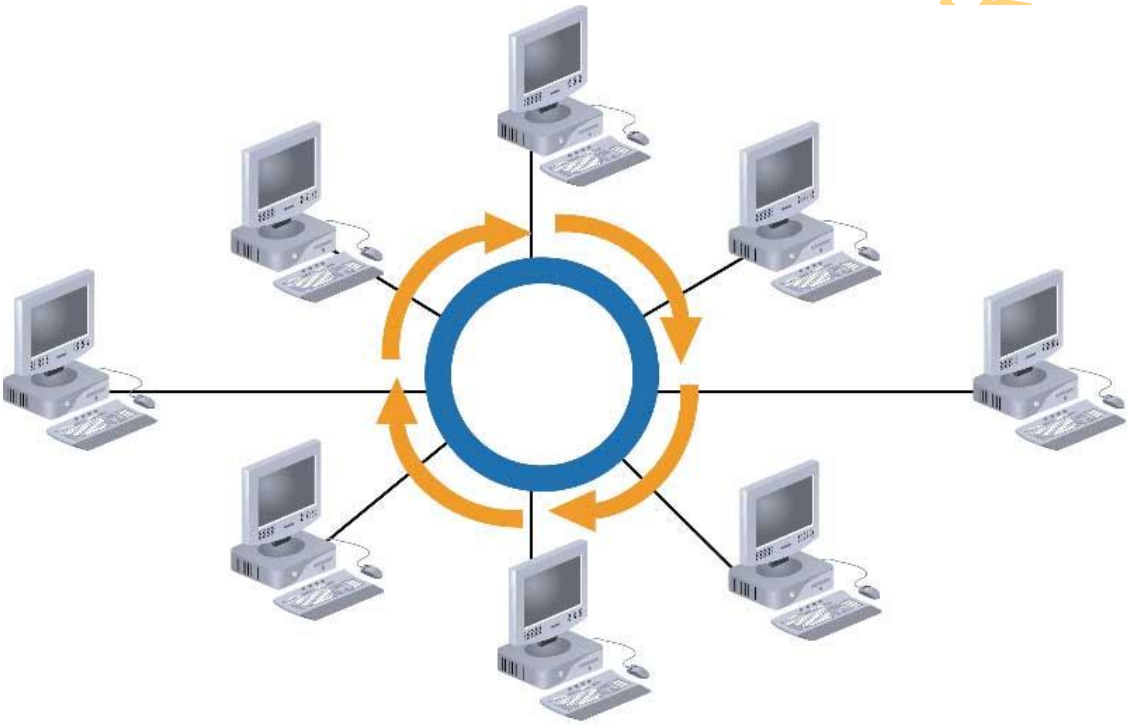


Figure 2.2: Ring Topology (adapted from Feyadat, 2008)

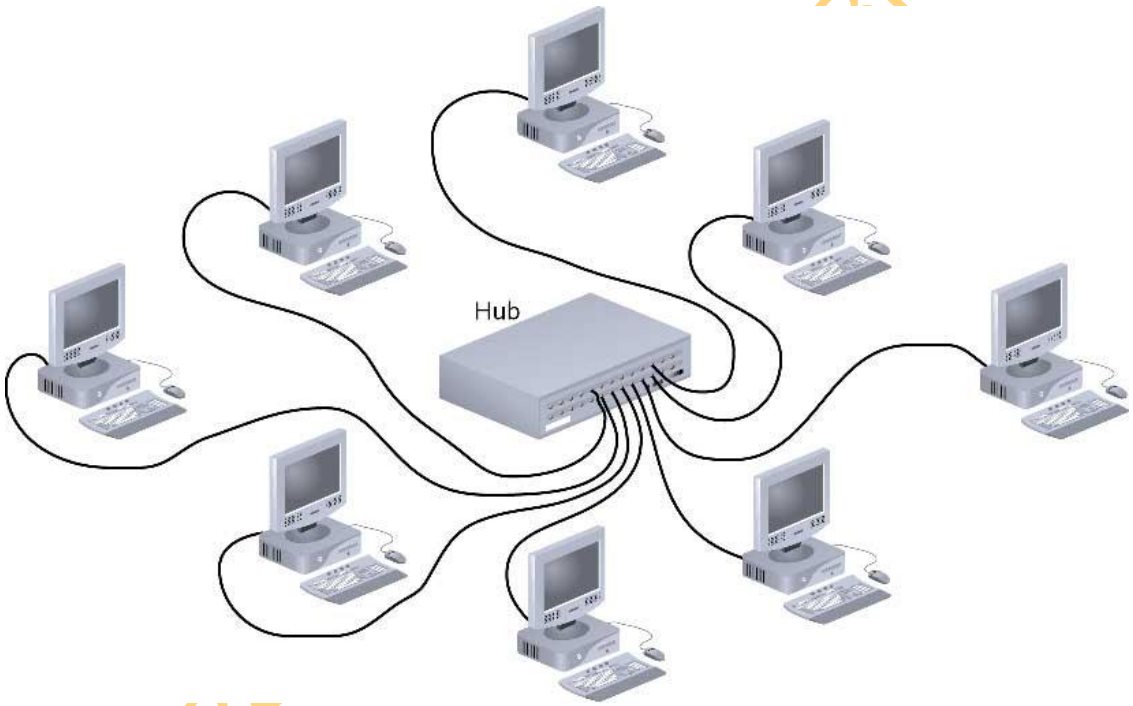


Figure 2.3: Star Topology (adapted from Feyadat, 2008)

(iv) Mesh Topology: is an all-channel topology in which every node is connected to every other node, illustrated by Figure 2.4. Mesh can be either a full mesh in which each node is connected directly to every other node or partial mesh in which some nodes are connected to all other nodes, and some are connected to only those with which they exchange the most data (Rouse, 2010). Message sent on a mesh network can take any of several possible routes to its destination, thus no traffic problems. Mesh network is fault tolerant, point to point links make fault identification easy, however, installation is complex as each node is connected to every node, and this makes managing the network difficult in addition to high cabling cost (Maninda, 2012).

(v) Tree Topology: is a hierarchical topology that can be seen as a group of star topologies arranged in hierarchy as illustrated by Figure 2.5. The tree topology arranges links and nodes into distinct hierarchies in order to allow greater control and easier troubleshooting. Tree is particularly useful in schools where each department can be connected using star topology and they can be connected to the big network in some way (Maninda, 2012). Tree topology provides point-to-point wiring for individual segments and all computers in the network have access to their immediate and larger networks, on the other hand, tree topology is difficult to configure and wire than other topologies and if the backbone breaks, the entire network goes down.

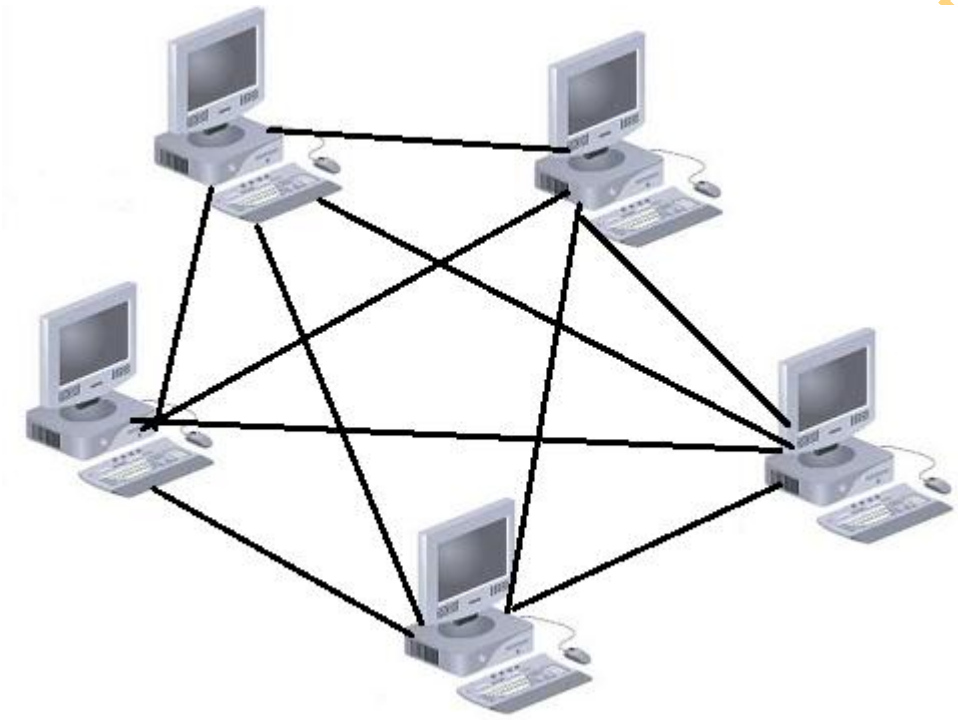


Figure 2.4: Mesh Topology

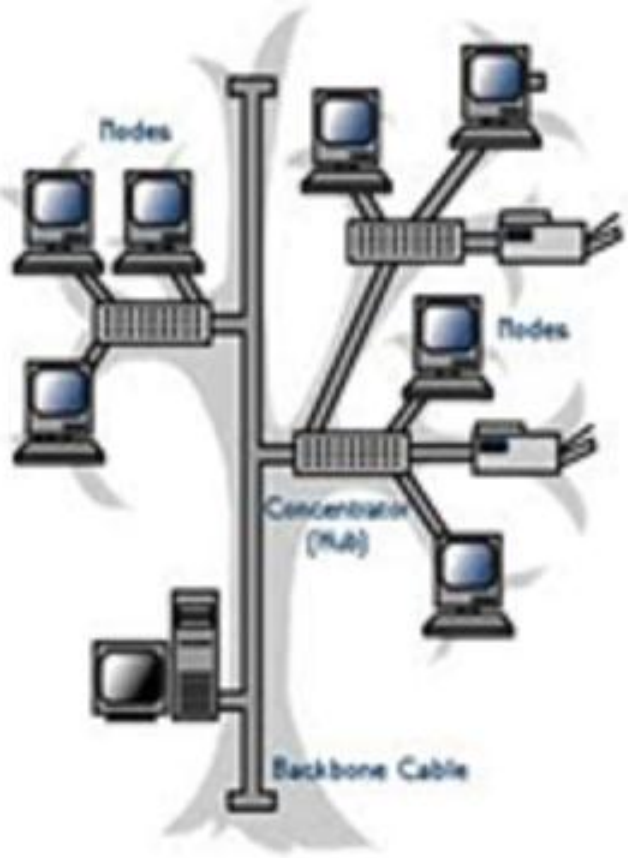


Figure 2.5: Tree Topology (Adapted from Winkelman,1997)

(vi) Hybrid Topology: combines two or more basic network topologies in a way that the resulting network does not exhibit one of the standard topologies. According to Oak (2011), hybrid topology results from the combination of two or more different basic network topologies. Common examples are star-wired ring network and star-wired bus network as described by Gilani (2012).

Star-wired ring consists of two or more star topologies connected using a multistation access unit (MAU) as a central hub, illustrated in Figure 2.6(a). Star-wired bus combines two or more star topologies connected using a bus trunk which serves as the network's backbone, as shown in Figure 2.6(b). Hybrid topology is extremely flexible and very reliable and is able to utilize the strongest aspects of other networks e.g. signal strength (Lister, 2012). Hybrid provides multiple pathways for data transmission between network nodes and the failure of one node does not affect network performance. Hybrid networks are versatile and can be adapted to a variety of network requirements and sizes

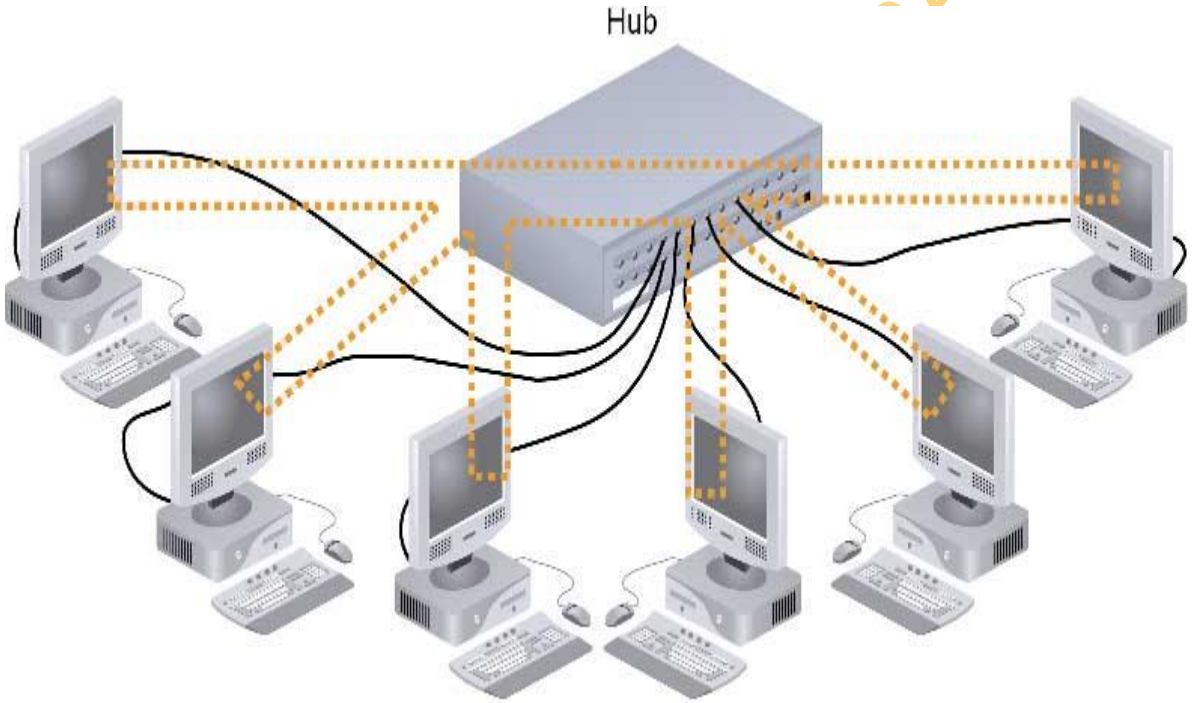


Figure 2.6(a): Star Wired Ring Hybrid Topology (adapted from Feyadat, 2008)

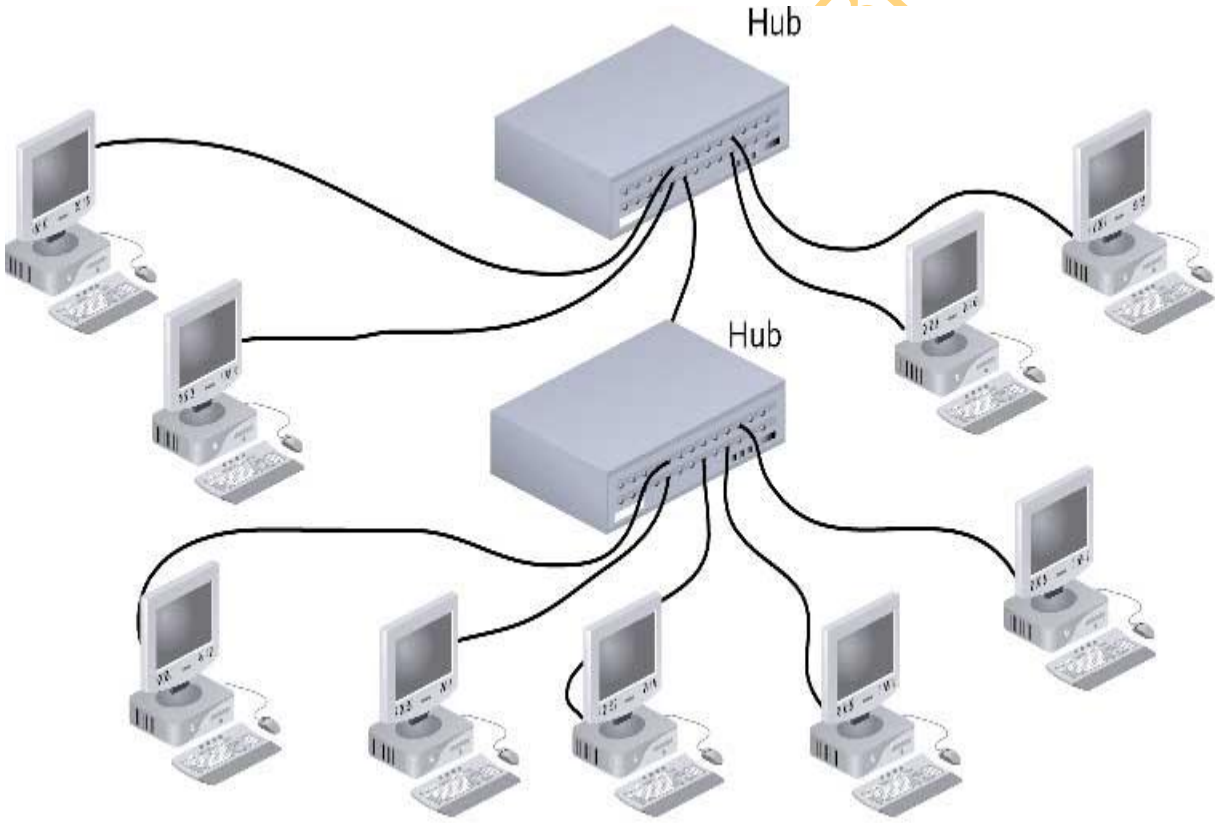


Figure 2.6(b): Star Wired Bus Hybrid Topology (adapted from Feyadat, 2008)

The use of computer networks resulted into distributed systems. Ashvin (2004) defined distributed system as a computing facility built with many computers that operate concurrently, are physically distributed, have their own failure nodes, have independent clock and are linked by a network. According to Adewunmi (2002) distributed system is a generalized transparent system consisting of collection of sites connected by a communication network, over which processing logic and/or data is shared based on usage. In distributed environments, computers are connected through a communication network (serial lines, Ethernet, ATM e.t.c), processing and/or data are spread geographically. Each computer is capable of communicating with the others, programs running on each computer can share information and request tasks to be executed. Some of the objectives of distributed system are to connect users in open and scalable manner, to make resources easily accessible and reasonably hide the fact that resources are distributed across network (Tanenbaum and Steen, 2007). Openness means each component can interact with other components and scalablity implies ability of the system to be easily altered to accommodate changes in the number of users, resources and computing entities. Distributed systems increase reliability and availability of services, resources can be shared by all computers, increase the speed of processing compared to the old centralised system, reduce communication costs, and have high processing capacity and are expandable (Tanenbaum and Steen, 2007). Managing distributed systems involves a lot of movement of codes or data within the network, this led to the introduction of code mobility concept (Fuggetta *et al*., 1999). Code mobility is a concept that involves communicating processes in mobile code systems exchanging programs instead of simply data. Code mobility is aimed at transporting some resource-accessing mobile code units from one host to another, and executes these code units on the resource-bearing host (Fong, 2003).

2.3 Software Agents

Software agents have been defined severally by different researchers and groups, especially in the artificial intelligence community where it originated from. Franklin and Graesser, (1996) defined an autonomous agent as a software entity situated within

and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own goals and so as to effect what it senses in the future.

Software agent has been widely accepted to reflect the above definition and also incorporate two common types of agents, the Mobile Agents and the static Intelligent Agents (Franklin and Graesser, 1996). Mobile Agents can migrate from one computer to another through the network to perform tasks on behalf of its owner, they provide opportunities to access remote computers on a network. Mobile agents provides for easy programmability of remote nodes by migrating and transfering functionalities required. The intelligent agents executes from their origin and depend on cooperation with other agents often in a generic manner through an agent communication language. Static intelligent agents can collaborate to devise or negotiate solutions for complex application scenarios (Bohoris, 2003). From the definition above it can be inferred that software agents execute in an environment or place, are proactive and reactive based on the situation in the environment, possess goal directed behaviour and are persistent. Agents are different from conventional software in several ways, researchers in the area such as Jennings and Wooldridge (1998) and many others agreed on three major attributes that are used to distinguish agents from conventional software, these are autonomy, proactiveness and reactiveness.

Autonomy is usually cited as the significant property that distinguishes agents, i.e ability of the agent to act independently without human intervention, which implies that the agent can make its own decision. There are different definitions for autonomy, according to Shoham (1993), the sense of autonomy is not precise, but the term is taken to mean that the agents' activities do not require human guidance or intervention. Huhns and Singh (1997) introduced five degrees of autonomy, absolute, social, interface, execution, and design. Conventional software does not have this attribute, its behaviour is imperatively declared. Proactiveness is the ability of agents to perform their tasks in a goal oriented manner. Proactiveness enables agents to take the initiative rather than acting simply in response to their environment. Agents act to satisfy a set of goals, this implies that the agent is aware of its goals, conventional software also have goals but the goals are implicit. Reactiveness enables the agent to perceive its environment and respond to changes that may occur in a real time.

2.3.1 Distributed System Paradigms

Movement of software did not start with mobile agents, there was an incremental evolution of mobile abstraction from client/server paradigm and remote evaluation paradigm. Distributed operating systems employ process migration mechanisms, allowing an operating system process to move from one machine to another and resume execution. Mobile code is a technique by which code, not just data, is transferred from source to destination where it is executed; in this case migration is transparent. According to Halls (1997) mobile code are data that can be executed as a program, and opined that the code can be pre-compiled for immediate execution on the recipient's processor, compiled upon receipt for subsequent execution or interpreted by the recipient. Fuggetta *et al*. (1998) defined code mobility as the capability to dynamically change the bindings between the code fragments and the location where they are executed. Code mobility offers a lot of advantages to distributed systems, among the numerous advantages are: supports for deployment and upgrade of distributed applications, services on the systems can be customized, it improves the robustness of the system to failure and provides support for disconnected operations in addition to load balancing (Fuggetta, *et al*., 1998; Picco, 2005). The rationales for mobile code as identified by Picco (2005) are: to move the knowledge close to resources and to enable client customization of the access to remote resources. In distributed systems, services can be executed when all the elements required are located at the same host, i.e. resources, know-how and computational capability are on the same host (Braun and Rossak, 2005; Fuggeta *et al*, 1998). Mobile code system is therefore classified with respect to which element is relocated, this is the basis of the classes of distributed application techniques. The following section examines the classes of distributed application techniques.

I. Client/Server Paradigm

In the client/server paradigm, the server contains the procedure (know-how, i.e, code), processing capability and advertises the set of services it is willing to provide to the client, the client on the other hand needs certain services but does not have the

resources nor the procedure to process them, it requests for these services from the server (Lange, 1998). The client and the server processes communicate either by message passing or remote procedure calls (RPC). In this paradigm, the code is stationary with respect to execution; the request is packaged with the service name and some additional parameters (Braun and Rossak, 2005). Client/server has received support from a number of technologies such as, the Remote Procedure Call (RPC), Remote Method Invocation (RMI) and the Object Request Broker's CORBA (Lange, 1998).

Remote Procedure Call: the client communiacates with the server over a socket connection using a protocol known to both parties such as HTTP (Hyper Text Transfer Protocol), TCP (Transmission Control Protocol) or (User Datagram Protocol) UDP. Both client and server have to be aware of the socket level details, the details are abstracted and the request is made to look like a local procedure call from the viewpoint of the client. Most RPC implementations use stub procedures, a client making a remote procedure call is actually calling a local stub. The client stub parcels up the procedure name and parameters, converts them to form suitable for transmission (marshaling), build a network message and sends the message to the remote machine through a protocol that have been agreed upon previously. When the server stub on the remote machine receives the message, it demarshals the message, extracts the procedure name and parameters and invoke appropriate procedure. The server stub waits for the procedure to finish and then sends a message with the result to the client stub, which later returns the result to the client (Oyatokun, 2004). RPC supports modular and hierarchical design of distributed systems, i.e., server and client are separate entities. However, it is difficult to send increamental results from server to client, RPC favours short results rather than bulk data transfer, and there is no way to pass pointers or procedure references to the server. In addition, client is limited to the operations provided by the server, thus client need to make several remote procedure calls as illustrated by figure 2.7, and bring intermediate data across the network on every call.

The remote method invocation (RMI) is the object equivalent of RPC. In RMI, the server defines objects that clients can use remotely. Clients can invoke methods of remote objects as if it were a local object running in the same virtual machine as the client, Figure 2.7 shows an illustration of this technique. RMI allows a java method to

obtain a reference to a remote object and invoke methods of the remote object as easily as if the remote object existed locally (Aderounmu *et al*, 2006).

II Remote Evaluation Paradigm

Remote Evaluation (REV) was proposed as an alternative paradigm to client/server approach and was first introduced as a concept by Stamos and Gifforf in 1990 (Neeran and Anand, 1998; Bohoris, 2003). REV uses client and server stubs just like the RPC and can be used with any language. In REV, the client has the procedure necessary for the service but lacks the processing capability and resources, the server on the other hand, does not provide a suitable application-specific service that the client can use. Therefore, the client sends fragments of its procedure to the server to execute; resources on the server are used as illustrated by Figure 2.8 and the result is returned to the client. REV can be incorporated into any programming language so the programmer can use any language most appropriate for the application. Gray (1997) argues that all functions and variables referenced in the procedure must be provided at the server or included in the procedure, making the semantics of a passed procedure different from that of a local procedure. This is because the passed procedure cannot access functions and global variables defined in the caller. In REV, the code fragment is mobile and it is sent from the client to the server for execution. Example of REV is the servlet that is uploaded from client to server (Pleisch, 1999).

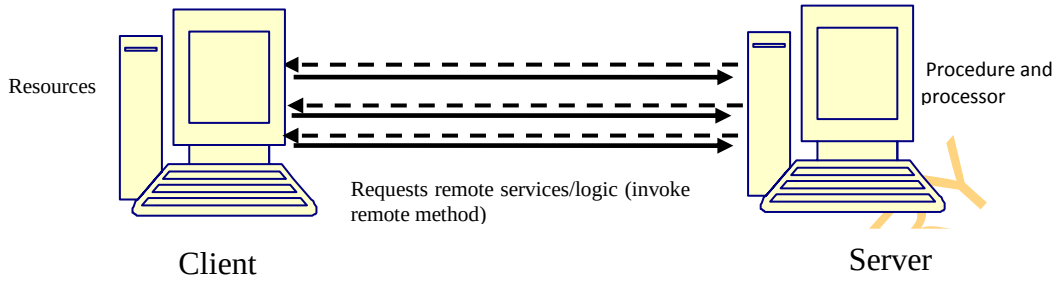


Figure 2.7: Client/server paradigm

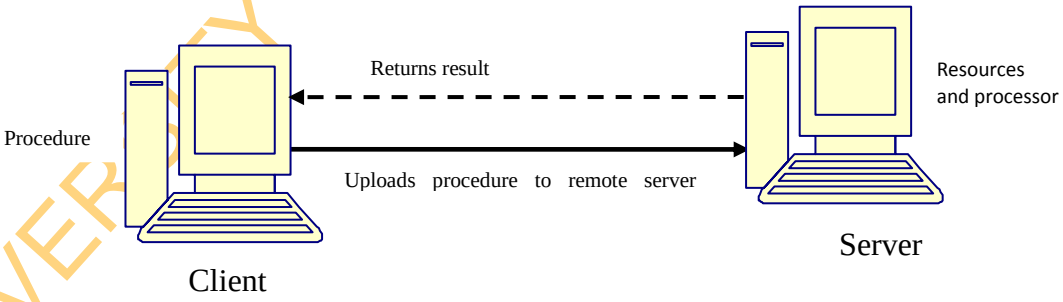


Figure 2.8: Remote Evaluation Paradigm (REV)

III Code on Demand paradigm (CoD)

The client has the processing capability and local resources in its execution environment but lacks the procedure to access and process the resources (Lange, 1998). Meanwhile, there is a server that has the needed procedure and is willing to share same. The procedure is obtained from the server when needed and execution begins on the client as soon as it gets the neccessary code (Figure 2.9). The client does not need to install the code since all the necessary code can be downloaded on request. In CoD, code fragment is mobile and it is sent from the server to the client. Examples of a code-on-demand application is java applets (Fischmeister, 2004; Braun and Rossak, 2005) and it can be downloaded from a Web server to a browser to be executed as part of an HTML page (Pleisch, 1999).

IV Mobile Agent Paradigm

Mobile agent is a special type of mobile code, an entire computational entity migrates through the network with its code, data and authority of its owner (Wenjuan *et al*., 2009). Mobile agents once dispatched from their origin are detached from the origin and can make multiple hops before returning to the origin with the results of the computation. Mobile agent is different from the others, (Client-server, Code on demend and remote evaluation) in the sense that the associated interactions involve the mobility of an entire computational component (Pleisch, 1999). In REV and CoD, the focus is on the transfer of code between components, whereas, in mobile agent, (Figure 2.10), a whole computational component, with its state, needed code and resources required to perform the tasks are moved to a remote site (Carzaniga *et al*, 1997). Mobile agent initiates its migration whereas, other software components e.g the browser, initiate the migration of CoD. In addition, code migration in CoD is unidirectional, from the server to the client and is bound to that client and dies when the client terminates, whereas mobile agents can make multiple hops and return to the origin before it is disposed of.

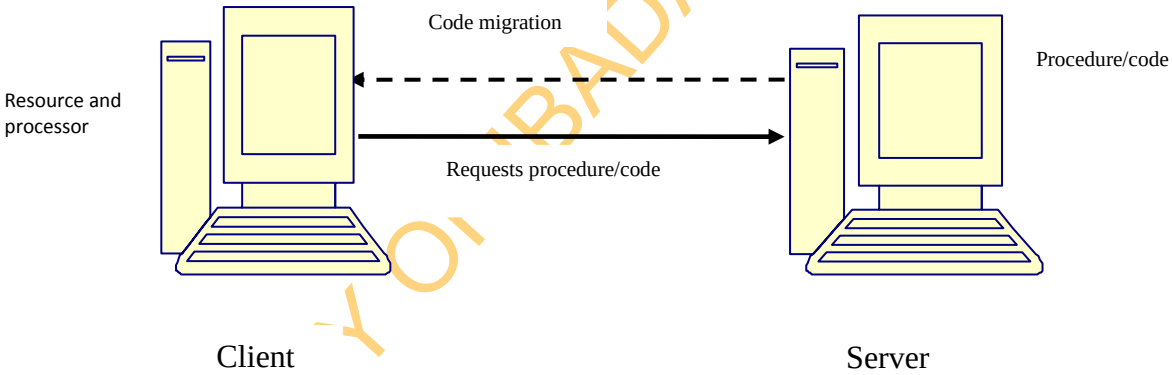


Figure 2.9: Code on Demand (CoD)

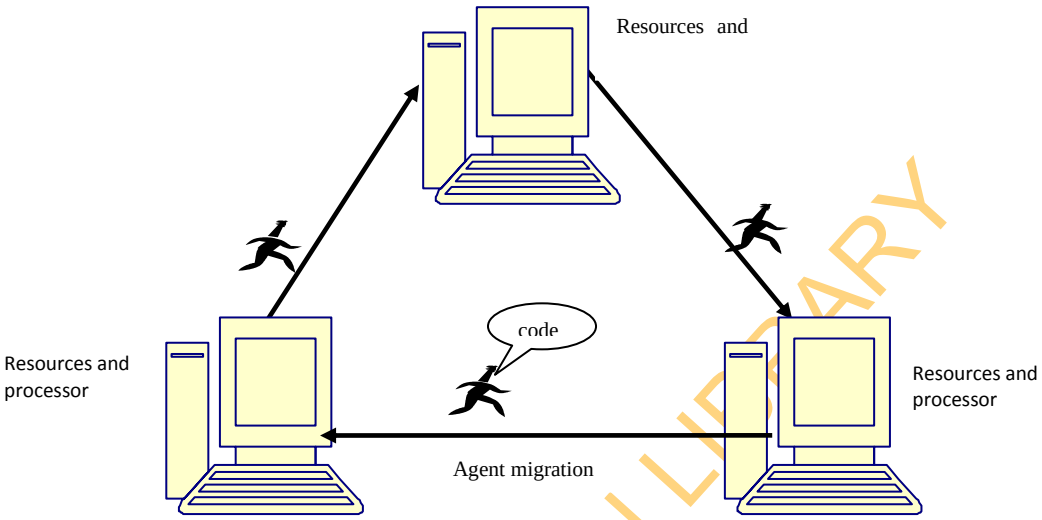


Figure 2.10: Mobile Agent paradigm

2.3.2 Characteristics of Agents

(i) Agents exhibit certain features that are suitable for a wide area mobile networking environment. Agent research community, (Wooldridge and Jennings, 1995; Franklin and Graesser, 1996; Biermann, 2004) agree that an agent can exhibit several but not all of the characteristics listed below:

(ii) Mobility: ability to move under its own control from one machine to another in a network. The agents' ability to carry code and data across network nodes underlies their suitability for a transient environment (Franklin and Graesser, 1996; Bellifemine *et al*, 2007).

(iii) Migration: it can suspend its execution at the hosting machine, transfer its current state (data) and code to a different host and resume its execution there, using its own resources.

(iv) Flexibility: agent algorithms may be dynamically modified according to pertinent environmental conditions, such as network congestion, node failures, and user environment (Franklin and Graesser, 1996).

(v) Concurrent problem solving: agents provide a clear, natural paradigm for performing tasks that may have several concurrent aspects.

(vi) Autonomy: agents must be able to decide where and when to migrate during the accomplishment of its mission without user's intervention (Dale and DeRoure, 1997; Jennings and Wooldridge 1998; Bohoris, 2003; Bellifemine *et al*., 2007).

(vii) Goal oriented: agents exhibit goal-oriented behaviour such that their actions can cause positive changes to the environment (Franklin and Graesser, 1996; Bohoris, 2003).

(viii) Navigability: to support the decision-making process of the agent (where and when to migrate), the objects must have the knowledge of its objectives and plan as well as parameters related to its environment.

(ix) Security: agents must be protected from malicious hosts and hosts must be protected from malicious agents or viruses (Biermann, 2004).

(x) Fault tolerance: Mobile Agent System (MAS) must provide resources to the agent programmers in order to help them in the detection of hardware and software errors, once an error is detected, the agent can perform the necessary procedures to overcome these errors e.g. notify the other agents about the

failures, move to an alternative resources, and wait until a resource becomes active again (Dale and DeRoure, 1997).

(xi) Performance: the moving process of an agent must be efficient, in a way to compensate its use, when compared to other paradigms. The agent needs to be small. According to the requirement of the application being developed, allowing its fast transfer between nodes of a network. The agent also may have to be able to execute in machine with possible memory and processing restriction, e.g. mobile computers and handheld computers.

(xii) Communication/collaboration: a mobile agent can communicate with other agents on the local host or remote machine to achieve its goals (Bohoris, 2003), Bellifemine *et al*. (2007), as well as Dale and DeRoure, (1997) call this feature social ability. Agent constantly migrate and do not have a fixed address on the network; such agents usually need location and tracking mechanisms e.g. of forwarding or actualizing the same service. Communication can be done in asynchronous way (based on datagram) and synchronously using RPC or shared files or resources.

(xiii) Adaptability: agent must be sensitive to diverse traffic conditions, connections and topologies of a computer network as well as to the diversity of resources available in each node (Braun and Rossak, 2005). Mobile agent platform can provide this information to the agent, the information is processed and used in the decision making process related to the migration, fault tolerance, operation mode, (connected / disconnected) of the agent (Bohoris, 2003).

(xiv) Multi platform support: distributed systems are sometimes heterogeneous set of hardware and software. This feature supports the agents' ability to execute in different Operating Systems and computer architecture.

(xv) Reactivity: often agents are required to anticipate future situations and respond to changes in their environment (Bohoris, 2003).

(xvi) Pro-activeness: agents do not only act in response to its environment but is able to exhibit goal-directed behavior by taking the initiative (Wooldridge and Jennings, 1995; Biermann, 2004)

2.3.3 Types of Agent

Software agents can be placed into different classes, there are several dimensions to classify existing software agents. Franklin and Graesser (1996) suggested classifying according to agent properties (mobile, learning etc), the tasks agents perform (information gathering, email filtering), their control architecture (planning, adaptive) and type of control mechanisms (algorithmic, rule-based or fuzzy). Nwana (1996) classified agents according to their properties; this section examines various classes of agents based on Nwana (1996) typology as illustrated in Figure 2.11.

(a) Mobility

Agents are classified into static and mobile agents based on their ability to move around the network (Nwana, 1996).

(i) Static agents do not move around a network. They execute on the host system where they reside, hence are not equipped with a mobility mechanism (Lange, 2004).

(ii) Mobile agents: their predominant feature is the ability to transport between nodes on a network or between nodes across networks (Biermann, 2004). They are not bound to the host that initiates them (Lange, 1998), consist of mobility mechanism that enable them roam the network, interact with foreign hosts, perform the required tasks on behalf of its user and return to the origin host with the result of its computation.

(b) Responsiveness

Agents are classified into deliberate and reactive agents based on their response to the environment in which they are embedded, this classifies agents (Nwana, 1996).

(i) Deliberate agents: originate from the deliberative thinking paradigm. The agents have an internal symbolic reasoning model and they plan and negotiate to achieve coordination with other agents (Nwana, 1996).

(ii) Reactive Agents: do not contain internal symbolic models of their environments, instead, they respond in a stimulus-response manner to the present state of the environment in which they are embedded (Nwana, 1996).

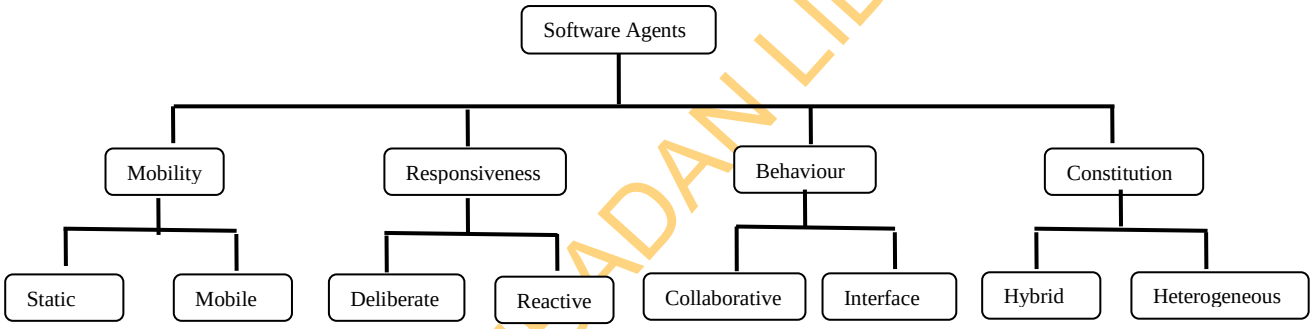


Figure 2.11: Agents Classification (Adapted from Nwana, 1996)

(c) Constitution

Agent system may be distinguished by its structure and composition. Using these criteria, agents are classifed as hybrid or heterogeneous.

(i) Hybrid agents combine two or more agent philosophies or types into a single agent so that the benefit of each agent type is maximized and the weaknesses are minimized (Nwana, 1996).

(ii) Heterogeneous agents system consists of integrated set-up of two or more agent types including hybrid agents, incorporated into a single system (Nwana, 1996). As a result, the agents can interoperate, thus the cost of maintaining and rewriting legacy systems is reduced.

(d) Behaviour

Agents may be classified with several attributes, it is generally accepted that software agents should exhibit the following (Nwana, 1996; Franklin and Graesser, 1996):

(i) Autonomy: agents act autonomously to pursue their given goal; they take initiatives to pursue their goal without human intervention (Biermann, 2004; Nwana, 1996).

(ii) Intelligence or adaptation: agents are capable of adapting to the environment they find themselves and learn from experience and the user models. They learn as they interact with the environment (Nwana, 1996; Wooldridge and Jennings, 1995)

(iii)Cooperation: software agents use standard languages and protocols to cooperate with human agents and other software agents to achieve their goal. This is supported by agent communication languages and protocol, and is the main function of FIPA (Foundation for Intelligent and Physical Agent) according to Bellifemine *et al*., (2007).

Based on these three components of behaviour, software agents are categorized according to Nwana (1996) and as depicted in Figure 2.12 as:

(i) Collaborative agents: emphasize both autonomy and cooperation with other agents to perform their tasks. They may also need to have "social" skills in order to communicate and negotiate with other agent. Collaborative agent system believes that a group agents function beyond the capabilities of the individual members. Collaborative agents are suitable for problems that are too large for a single system to handle and problems that are distributed in nature. It is believed that they can overcome resource limitation and system failure

(Nwana, 2006). However, coordination among multiple agents, particularly when they are autonomous and heterogeneous is a challenge.

(ii) Collaborative learning: emphasize intelligence (adaptive) and cooperation more than autonomy (Nwana, 1996).

(iii) Interface agents: these emphasize autonomy and intelligence in order to perform useful tasks for their owner. They learn by observing and imitating the user, through feedback from the user or by interacting with other agents. The main challenge is how to assist users without bothering the user and how to learn effectively (Nwana, 2006).

(iv) Smart or intelligent agent: are cooperative, autonomous and intelligent.

These distinctions however are not definitive, its only that each type places more emphasy on the intesecting features than the other feature. For example, collaborative learning agents place more emphasy on cooperation and learning than on autonomy, this does not imply they are not autonomous.

2.4 Mobile Agent Technology

Mobile agent technology facilitated by recent advances in computers, communications and artificial intelligence, provides an attractive framework for the design and implementation of communicating applications in general and distributed knowledge networks in particular. The computational model based on mobility, can be seen as a replacement, refinement or extension of the traditional client/server paradigm. The mobile agent framework emerged in the pursuit of open and decentralized models relevant to the dynamic and distributed nature of computations on the Internet.

A mobile agent can be defined as a named object that contains code, persistent state, data and a set of attributes (e.g movement history, authentication keys) and can move about or transport itself from one host to another as needed to accomplish its tasks (Wenjuan *et al*., 2009). It is commonly agreed that mobile agent is an object that can migrate through many nodes of a heterogeneous network of computers under its own control, with its code, data and execution state in order to perform tasks using resources of these nodes (Roberto, 2001). Mobile agent can suspend its execution on an arbitrary point and transport itself, during migration the agent is transmitted completely, (code,

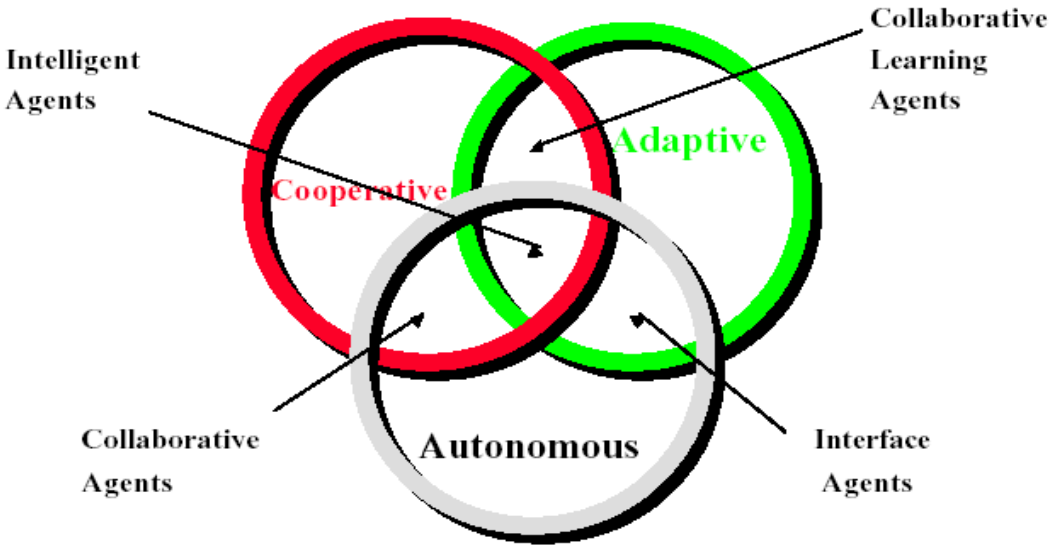


Figure 2.12. Types of agents (adapted from Nwana, 1996)

data and execution state) and at destination, execution is resumed either at exactly the point it was suspended (strong migration) or at some predetermined point (weak migration).

Mobile Agent System (MAS) is the computational framework that implements the mobile agent paradigm, and it provides services and primitives that help in the use, implementation and execution of system development using the mobile agent paradigm (Roberto, 2001). Dilyana and Petya,(2002) defined Mobile Agent System (MAS) as a distributed execution environment for mobile agents. MAS provides two types of services: the basic services are services that are vital for the survival of mobile agent (white page services) and the enhancing services that provide utilities that make mobile agent more efficient in different ways but are not vital in the same way as the basic services (yellow page services).

The mobile agent technology research receives so much attention due to its several advantages it provides over the client server technology for distributed systems (Stoian and Popirlan, 2010; Singh *et al*., 2012). Mobile agent expends low bandwidth since it only moves when neccessary can continue execution even when disconnected from the network (Aderounmu, 2001), it has ability to clone itself to perform parallel execution, implementation and deployment is easy, and it is reliable (Lange, 1998). The basic motivation behind agent's migration is to perform access and information processing locally to a resource or the user, instead of remotely. On the other hand, mobility raised security and efficiency issues.

2.4.1 Strength of Mobile Agent Paradigm

A significant number of benefits of mobile agents have been identified by various researchers such as Chess *et al*., (1994); Lange, (1998); Lange and Oshima, (1999), from the studies of mobile agents technology against alternative paradigms for distributed system such as the Client-server paradigm. Lange (1998) and Lange and Oshima (1999) identified seven good reasons for using mobile agents and these are supported by a good number of other researchers in the area, these include:.

(i) Reduce network load: distributed system often require multiple interactions to complete a task. Using mobile agent allows users to send conversation to a

destination host, so that all interaction can take place locally, hereby, reducing network traffic (Chess *et al.* 1994; Lange, 1998; Bohoris, 2003). Instead of moving large amount of data from remote host and then processing it at the receiving host, an agent sent to the remote host can process the data in its locality and return with only the results.

(ii) Overcome network latency: in critical real time systems, immediate response to important event is required. Controlling the decision of such systems remotely through the network can involve significant latencies (delay) which is unacceptable for some real-time systems. Mobile agent can be dispatched from a central controller to act locally in critical real-time system and thus respond immediately to real time systems that require immediate response (Lange, 1998; Pleisch, 1999, Bohoris, 2003).

(iii) Encapsulate protocols: when data are exchanged in a distributed system, each host has the code that implements the protocols needed to properly code outgoing data and interpret incoming data (Bohoris, 2003). The continuous evolution of existing protocols (to accommodate new requirements of efficiency and security) in distributed systems however, makes it very cumbersome to upgrade protocol code properly in each host. Mobile agents are able to move to remote hosts in order to establish 'channels' based on proprietary protocols (Lange and Oshima, 1999; Pleisch, 1999).

(iv) Execute asynchronously and autonomously: these attributes favour mobile agent in wireless networks. A continuous connection may not always be feasible with wireless connection due to its fragile expensive nature. The mobile device user can insert the task into a mobile agent, which can be dispatched into fixed network and operate asynchronously and autonomously to accomplish its task (Biermann, 2004, Braun and Rossak, 2007). The mobile user can reconnect and collect the agent with the result later (Lange and Oshima, 1999). This makes mobile agent paradigm especially desirable in Nigeria where the network connections are unreliable and often insufficient.

(v) Adapt dynamically: mobile agents have the ability to sense their execution environment and take decisions based on that dynamically (Lange, 1998). In case of a mobile agent moving to a number of host nodes, it can adapt its future behaviour according to information that it has already collected and stored in its state (Braun and Rossak, 2007).

(vi) Naturally heterogeneous: mobile agents are generally independent of the computer configuration and transport. Because mobile agents are generally computer and transport layer independent (dependent only on their execution environments), they provide optimal conditions for seamless system integration (Lange, 1998; Lange and Oshima, 1999).

(vii) Robust and fault tolerant: the ability of mobile agent to adapt dynamically to adverse situations or events makes it easier to build robust and fault tolerant systems (Lange, 1998). In a fixed network, by dispatching mobile agent to execute locally on a remote host, its operation continues even in the face of network failure that makes remote communication unavailable (Bohoris, 2003).

2.4.2 Issues Associated with Mobile Agents

Mobile agent technology has received a lot of attention in the past especially in the academic community, but it is yet to be adopted in the commercial and industrial communities (Pinsdorf and Roth, 2002; Bohoris, 2003). This is attributed to a number of issues associated with mobile agent technology introduced by its complexities, autonomy and mobility (Kotz and Gray, 1999; Bohoris, 2003; Tudor, *et al*., 2004, Singh *et al*., 2012). Some of these issues are discussed in this section:

(i) Lack of standard and interoperability has greatly hindered the global acceptance of mobile agent technology (Bohoris, 2003; Tudor *et al*., 2004, Singh *et al*., 2012). Efforts made by the FIPA (Foundation for Intelligent Physical Agent) and OMG MASIF (Mobile Agent System Interoperability Facility) to provide standards for agents and agent platforms (Fortino and Russo, 2003) are yet to be generally adopted. MASIF's effort is directed at making mobile agent platforms interoperate, while FIPA has come up with certain specifications for agents' communication (Bellifemine *et al*, 2007).

(ii) Lack of killer application. All the applications the mobile agent technology has been applied to can also be solved by other technologies like the client/server technology. Presently, there is no application that can only be achieved through the use of mobile agent paradigm (Bohoris, 2003). Mobile agent paradigm has only been proved to have superior performance at handling such applications,

such as reduced network load, reduced latency, increased fault tolerance and better adaptation to failure.

(iii) Limited practical experience: There has been a good theoretical base for mobile agent paradigm but real world application and practical assessment of mobile agent technology has been limited (Jansen *et al*., 1999). Most of the mobile agent applications that have been built are in laboratories and are for academic research purposes. One of the factors responsible for this is its complexity.

(iv) Complexity: mobile agents coding and deployment are complex processes, some inherent capabilities, such as moving and cloning, also add to the complexity of the design and development process (Jansen *et al*., 1999).

(v) Performance overhead: mobile agents reduce bandwidth utilization and network latency but often at the expense of increased utilization of resources at network nodes (Bohoris, 2003).

(vi) Getting ahead of evolutionary path: Bohoris (2003) is of the opinion that the direct shift from client/server to mobile agent paradigm for distributed applications is too sudden and its unlikely and that the evolutionary path takes time and will most likely move gracefully from centralized protocols, to distributed object frameworks, followed by mobile code solutions and later by mobile agents.

(vii) Security is an important issue in mobile agent paradigm, the agents need to be protected from other malicious agents, hosts need to be protected from malicious agents and agents also need protection from malicious host (Fortino and Russo, 2003). Researches have shown that the first two can be achieved through cryptography and authentication, while the last is difficult to achieve because the agent must be interpreted on the host before it can execute. There is little that can be done to secure the agent form the host other than using trusted hosts.

In addition, Giovanni (2004) identified ten reasons why agents fail; these include difficulties of design, inability to outperform other mechanisms such as remote evaluation, difficulties of development, testing and debugging, difficulty involved in authenticating and controlling agents. Others are lack of ubiquitous infrastructure; mobile agents can be brainwashed, they cannot keep secrets and are suspiciously similar to worms.

2.4.3 Security Issues with Mobile Agent Technology (MAT)

As good and effective as the MAT is, with all its benefits, it introduces new security threats from malicious agents and hosts. It also introduces an additional complication in the sense that, as an agent traverses multiple machines that are trusted to different degrees, its state can change in ways that adversely impact its functionality (Farmer *et al.*, 1996). Security threats of mobile agent technology have been generally classified into three categories; these are disclosure of information, denial of service and unauthorized access (Admassu, 2008; Nitin *et al.*, 2011). Wayne (2000) introduced an interference or nuisance as the fourth class. Security concern of mobile agent technology is in various dimensions, between the agent and the agent platform, between various agents and the network. The agent platform can be compromised by visiting agent; an agent may be compromised by the platform or other agents and the network may be compromised by incoming agent, common security threats to mobile agents are presented under these scenarios.

2.4.3.1 Agent to agent platform

A malicious agent has two major lines of attack on the platform, to gain unauthorized access into the platform and to use this unauthorized access in an unexpected and destructive way (Jansen *et al.*, 1999; Wayne, 2000). Possible threats of this category include:

Masquerading: the agent poses as another agent to gain access to services or data at a host it is not authorized to and to shift blame for actions (Biermann, 2004).

Denial of service: agents may attempt to consume or corrupt a host's resources to prevent other agents from accessing the host services (Wayne, 2000). Host can ignore an agent's request for services or access to resources, i.e., block agent execution.

Unauthorized access: agents can gain access to sensitive data by exploiting security weaknesses or interfering with another agent to gain access to data, information residing at the platform can be disclosed or altered. Depending on the level of access, agent may be able to completely shut down or terminate the agent platform.

2.4.3.2 Agent Platform to Agent

The security threats in this category are difficult to detect and prevent because the platform has full access to both agent data and code (Admassu, 2008). The platform is responsible for accepting and executing the mobile agent, this makes visiting agent open to attacks from the platform (Biermann, 2004). A receiving platform can easily isolate and capture an agent, extract information, corrupt or modify its state or code. Threats in this category are:

Masquerading: Host could assume false identity in order to lure agents, and extract sensitive information from the agent. When an agent arrives at a host, it expresses its code, state and data, malicious host can modify agent's code, state or data without being detected (Nitin *et al* 2011). According to Admassu (2008), this attack has more to do with the capability of a visiting agent to correctly identify and authenticate its executing host, while it is on it.

Denial of Service: a host may ignore service requests, introduce unacceptable delays for critical tasks, refuse to execute agent's code, or terminate an agent without notification (Admassu, 2008). The host may also deadlock other agents while competing for same resources or livelock by generating more work continuously .

Eavesdropping: with agents that are interpreted, the host can inspect their internal algorithms and data. This is serious in mobile agent systems as agent platform has access to the agent's code, state and data (Admassu,2008). The platform can monitor communications, read instructions executed by agent, read all unencrypted data; the visiting agent may be at the risk of exposing proprietary algorithms, trade secrets and transmit privilege information (Wayne, 2000). The platform can also infer secrete information from service requests and identity of the agents it communicates with (Wayne, 2000; Admassu, 2008).

Alteration: hosts can change an agent's internal data, state, code or results from previous processing to influence the agent. According to Wayne (2000) alteration can only be detected but cannot be prevented, since the agent's code and data can be interpreted by the host.

2.4.3.3 Agent to agent

Agent can exploit the security weakness of other agents through any of the following

Masquerading: agent assumes the identity of another agent; this harms both the attacked agent and the agent that is impersonated (Wayne, 2000).

Repudiation: after agreeing to some contract, an agent can subsequently deny that any agreement ever existed or any event or action ever happened or a legitimate transaction ever occurred (Wayne, 2000), it can also modify the conditions of the contract.

Denial of Service: agent can debar other agent from executing its task by sending repeated messages to another agent (Wayne, 2000). This can cause undue burden on message-handling techniques of the system and monetary loss if agent is being charged for resource-utilization.

Unauthorized Access: agent may get hold of and modify another agent's data or code it has no right to (Wayne, 2000). An agent may directly interfere with another agent by accessing and modifying agent's data or code which in turn changes the agent's behaviour.

Eavesdropping: a malicious agent may use platform services to eavesdrop on intra-platform messages of an unsuspecting agent (Wayne, 2000).

2.4.3.4 Other Entities against Agent System

Some other entities both outside and inside the agent framework may attempt to attack the agent system, and carry out actions that could harm or disrupt the agent system (Wayne, 2000).

Masquerading: an entity may disguise itself to intercept or gain access to inter-agent and inter-platform communication for example through replay or forgery (Wayne, 2000).

Eavesdropping: an entity may eavesdrop on messages in transit to and from a target agent or platform to gain information (Wayne, 2000).

Alteration: an entity may intercept agents or messages in transit and modify their contents, substitute other contents or simply replay the transmission dialogue at a later time in an attempt to disrupt the synchronization or integrity of the agent framework (Wayne, 2000).

Denial of service: service can be denied the entire agent system through available network interfaces (Wayne, 2000).

Biermann (2004) categorized these threats using their modes of attack which are fundamental requirements of computer network users. These threats are directed at attacking the following:

- Integrity: attacks against integrity interfere with the agents' execution. Sub-classes in this class are integrity interference and information modification
- Availability: an authorized agent is prevented from accessing objects or resources to which it should have legitimate access. In this class, denials of service, delay of service and transmission refusal are typical sub-classes.
- Confidentiality: a host illegally accesses the resources of mobile agent. The three sub-classes that are identified are eavesdropping, theft and reverse engineering
- Authentication: agent is unable to correctly identify and authenticate its executing host. two subclasses identified here are masquerading and cloning

2.4.4 Protection Methods Against Security Threats

To deal with security, different protection measures have been proposed, these measures provide either detection or prevention of the threats named in the previous section. Mobile agent system (MAS) needs to provide the following security mechanisms:

Privacy and Integrity: agents carry their state and data; these data can have sensitive information. Agents must be programmed in order to apply different levels of access to the information they convey, according to the level of confidentiality of the host. Techniques applicable include contractual agreement, trusted hardware and trusted nodes (Borselius, 2002). Mobile agents system must also provide support for the detection of attacks.

Authentication of agents and servers: MASs have to prevent malicious agents from being confounded as authorized application agents. It must also avoid malicious hosts, receive authorized agents from the system. Mechanisms that allow identification (e.g digital signature) and certification of the servers as well as the agent or the user that the agent represents, have to be supported.

Authorization and access control: the access for some resources of the system must be restricted in MAS. Agents can be configured to respect policies of quota of occupation in the disk and can have the limited access of write to disk or create connections in the network. Execution tracing has also been proposed (Borselius, 2002) to detect unauthorized modifications of agents.

Auditing and metrics: agents consume resource such as network bandwidth, disk space and CPU during its life. These resources have to be monitored in a way to provide information to the agents and to administrators of the distributed system. Agents'activities can be monitored by setting limits for each of the resource and auditing the activities of agents.

Having said so much about mobile agent, it should however be noted that mobile agent cannot operate without an underlying system which provides services for running the agent code, migrating the code and accessing management information in network nodes. Such a system is called Mobile Agent System (MAS). The functionalities of MAS can also be achieved by a static agent previously installed for the same purpose.

2.5 Mobile Agent Platform

The agent platform is the execution environment for agents, it is the underlying system which provides services for running the agent code, migrating the code and accessing management information in network nodes (Pears, 2005). According to previous works (Tudor et al., 2004; Pears, 2005), it has been established that agent platform provides common functionalities that support the migration of agents, the communication between agents, various programming/interpreted languages and various forms of security. A good number of mobile agent systems such as JADE, Agent TCL, Grasshopper, Aglet e.t.c, are in existence nowadays, but each operates independent of the other, this hinders the interoperability of mobile agents. Most agent platforms offer enoumous flexibility at the cost of usabilty, where the user gets tanggled in aspects that are completely irrelevant to his application, thereby losing focus. To support this Tudor *et al.*, (2004) says some mobile agent platforms offer extended built-in functionality at the price of interoperabilty in which case the user has several predefined agent services to chose from and plug in his application but he is confined to that specific agent platform. Often times, a lot of adjustments are made to cutomize either the platform or the agent to make them compatible one with another, which leads to loss of functionalities thus reducing the efficiency and effectiveness of the whole system.

2.5.1 Mobile Agent Platform Architecture

The architecture of agent platform consists of an agent server which runs within an interpreter and is capable of hosting mobile agents implemented in the interpreted language. Pears (2005) defines agent server as a server process that runs at hosts willing to accept mobile agent. The agent server provides a mobile agent interface through which the mobile agent requests for communication and migration from the agent server, and the transport interface, from where the agent server sends and receives mobile agent over a secure communication channel. Figure 2.13 describes the architecture of typical Mobile Agent System, consisting of agent server, mobile agent management, mobile agent interface and transport interface. The transport interface uses TCP, a reliable connection oriented protocol. Agent management manages mobile agent initiation, suspension, resumption and termination, as well as routing requests to the transport and communication manager. Transport manager captures and restores mobile agent state (data variables, code and execution states). Communication manager is resposible for mobile agent communication with local agents. Communication between agents is made possible using method calls or shared memory.

2.5.2 Mobile Agent Systems Interoperability

Agents need to communicate with one another in the process of working together to achieve a common goal, agent paradigm of software development believes that communities of agents are much more powerful than any single agent, which necessitates interoperation of agent systems. Interoperability in mobile agent community focuses on the execution environment and standardization of certain aspects and features of agents while in the non-mobile agent contex the focus is on communication, that is effective exchange of information and knowledge content of agents. Interoperability has been defined by Pinsdorf and Roth (2002:1) as follows:

> *Two mobile agent systems are interoperable if a mobile agent of one system can migrate to the second system, the agent can interact and communicate with other agents (local or even remote agents), the agent can leave this system, and it can resume its execution on the next interoperable system.*

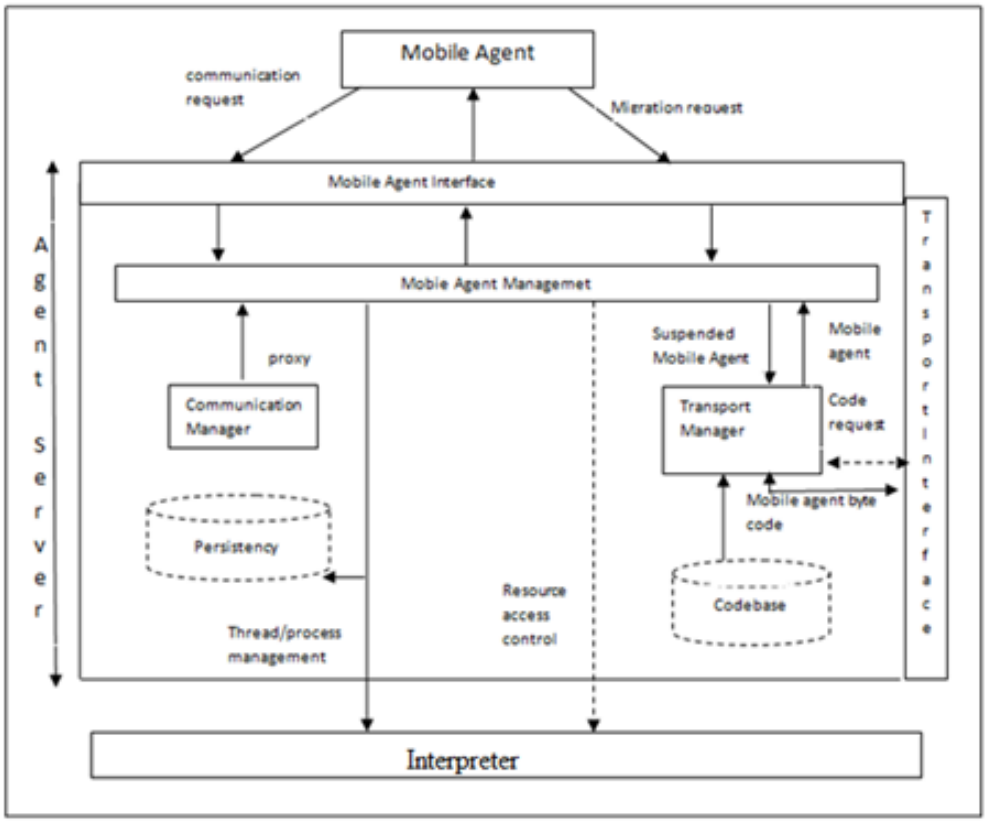


Figure 2.13: Mobile Agent System Architecture. (Adapted from Pears, 2005)

Research works are currently going on in the area of mobile agents' interoperability, several solutions have been proposed but they lack the necessary flexibibility to provide adequate degree of interoperability among the available MASs (Fortino and Russo, 2003). Interoperability is paramount to the global acceptance of MAS in heterogeneous and open distributed environments where agents must interact with other agents to fulfil their tasks and visit different agent platforms to access remote resources.When mobile agents migrate to a new host, the platform on the host provides execution environment, the mobile agent might execute code, make remote procedure calls to access resources on the host, collect data or initiate another migration process (Zenghache, 2002). Problems arise from the fact that not all platforms for mobile agents are the same and thus, cannot provide necessary services for non-compliant mobile agents. Mobile agents differ in architecture and implementation; this impedes interoperability, rapid proliferation of agent technology and growth of the industry (Milojicic *et al*, 1998). Interoperability is directed at making an agent system accept and support the running of agents from another agent system and vendor, support the transfer of agent to other agent systems and find other agents and agent systems. To achieve these, mobile agent paradigm has to clearly define the following: agent management, agent transfer, agent and agent system name, agent system types, authority and location syntax. MASIF (Mobile Agent System Interoperability Facility) attempts to standardize some aspects of this execution environment to provide for mobile agents to interoperate (Zeghache, 2002; Schoeman and Cloete, 2003), while FIPA (Foundation for Intelligent Physical Agent) attempts to standardize certain aspect of mobile agent (Bellifemine *et al*., 2007).

Grimstrup *et al.* (2002) proposed an Interoperability Application Programming Interface (IAPI) that supports registration, lookup, messaging, launching and migration of agent across different patforms. The system was implemented on Grid Mobile Agent System (GMAS). The design added three layers to the MAS layers: Foreign2GMAS (foreign MAS API to GMAS API translation), GMAS2Native (GMAS API to Native API translation), and common communication and discovery service, another two components are added to the participating MAS, Agent Luncher and Gateway. Any MAS willing to host foreign agents implements GMAS2Native translator, agent launcher and gateway while a MAS that wishes to send its agent to operate on another MAS implements Foreign2GMAS translator and gateway. The solution was tested with Java based MASs D'Agent, EMAA(Extensible Mobile Agent Architecture) and

NOMADS (Network Object Mobile Agent System). The system only enabled agent migration among diverse agent platforms but the agents may fail to execute due to difference in the level of the java API. D'Agent uses JDK 1.0.2 while EMAA and NOMADS support Java 2. The performance of the system was also slow, the additional layers on the platforms being the major factor.

Secure and Open Mobile Agent (SOMA) (Bellavista *et al.*, 2001) is another attempt at achieving interoperability, it was developed in compliance with both CORBA (Common Object Request Broker Architecture) and MASIF. SOMA uses a CORBABridge which consists of CORBA client/server which simplifies the design of SOMA entities as CORBA client /server and MASIFBridge which implements the MASIF functionality. The MASIFBridge is limited to only one place per domain in order to reduce the overhead it would have introduced, CORBA client is lightweight and it is extended to many places in the same domain. SOMA achieves interoperability with java-based mobile agents, agents can move, execute and interact, meanwhile the nodes in the system are grouped into domains, each domain has a default place that houses the MASIFBridge which controls the interdomain routing. This implies the success of the interoperability in the system lies with the default place, the system has a single point of failure. The security and fault tolerance of the system is important for interoperability to be fully attained, SOMA achieves security but it is not fault tolerant. Moreover, the MASIFBridge introduced a considerable overhead.

Fortino and Russo (2003) proposed a java-based framework for interoperability among java-based mobile agent systems. The framework consists of three software layers, the Interoperable Mobile Agent Layer (IMAL), the Adaptation Layer (AL) and the Platform-dependent Mobile Agent Layer (PMAL) which constitute a considerable overhead. At the same time, a Mobile Agent Bridge must be developed for each agent platform to be able to migrate; this constitutes an additional overhead on the system. The framework permits interoperability of execution, migration and interaction of java-based mobile agent systems.

Based on the shortcomings of the above interoperability models, there is a need to find a common platform on which agents from different platforms and vendors with different design and architecture can communicate, execute and interact without fear of risk or vulnerability to failure and other attacks.

2.5.3 Standardization of Mobile Agent Systems

There are several well known mobile agents' architectures, all built to serve the same purpose but they have many differences in term of technology, interpreted languages, concepts, architecture and implementation (Mitrovic *et al*., 2011). Some of these systems were developed in academic environments others were developed in industry. According to Nikos *et al*, (2003), some of these systems have disappeared such as Telescript, Odyssey and others will yet disappear while others will emerge in the future. Nevertheless, there are several issues that need to be sorted to make mobile agent technology widely accepted as highlighted by Stoian and Popirlan (2010), they include, supports for secure and efficient execution, standardization, appropriate programming languages and coordination models. Agent research community is aware of the fact that various languages with different programming paradigm such as procedural, object oriented, functional languages, will be used to implement agents, harware configurations and Operating Systems will also differ for different vendors, and agents are expected to be autonomous. The major challenge is to make agents socialize regardless of their language, architecture, concepts and implementation differences. There is a need to set up stardards that these agents and agent platform willing to interoperate must conform to. Standardization aims at finding a uniform platform for agents and agent systems to cooperate to achieve a common goal irrespective of their vendors, design, architecture and implementaion. The standardization issue of mobile agent is tackled by FIPA (Bellifemine *et al*., 2007) while the standardization of MAS is handled by the Object Management Group MASIF (Zeghache *et al*., 2002).

2.5.3.1 Foundation for Intelligent Physical Agent (FIPA)

FIPA is an international association established in 1996 with the aim of developing and setting computer software standards for heterogeneous and interacting agents and agent-based systems. FIPA evolved in stages and has since inception produced a set of specifications that went through three stages of review, FIPA'97, FIPA'98 and FIPA2000 (Bellifemine *et al*, 2007). FIPA defines a set of full standards for both implementing systems within which agents could execute, that is agent platform architecture and specify supports for agent communication, agent management, and agent message transport.

FIPA'97 was identified as a collection of seven parts; the first three specify the core middleware technologies of agent management, agent communication and agent/software interaction. The remaining four are personal assistant, personal travel assistance, audio-visual entertainment and broadcasting network provisioning and management.

FIPA'98 consists of many improvements on FIPA'97, the basic agent management mechanisms were extended, new interaction protocols and human-agent interaction were introduced and work started on agent security management and ontology service to host.

FIPA2000 added a new dimension to agent technology; it was directed at promoting technologies and interoperability specifications that facilitate internetworking of intelligent systems in commercial and industrial enterprises. FIPA2000 focused on high-level semantic-based communications, interoperability between agents rather than platforms and agent agreements and interactions over an extended period of time.

FIPA2000 remains the current specification though a lot of improvement has been made to it since it was adopted by IEEE in 2005, and named FIPA-IEEE. The focus presently is on agents and web service interoperability, human-agent communication, mobile agents and peer-to-peer nomadic agents. Some of the important achievements of FIPA in agent technology since inception are listed below according to Bellifemine *et al* (2007):

i. A set of standard specifications supporting inter-agent communication and key middleware services.
ii. An abstract architecture providing an encompassing view across the entire FIPA2000 standards. This architecture underwent an incomplete reification as a Java Community Project known as the Java Agent Services (JAS) Java Specification Requests (JSR82).
iii. A well-specified and much-used agent communication language (FIPA-ACL), accompanied by a selection of content languages (e.g. FIPA-SL) and a set of key interaction protocols ranging from single message exchange to complex transactions.
iv. Several open source and commercial agent tool-kits with JADE generally considered as the leading FIPA-compliant open source technology available today.

v. Several projects outside FIPA such as the completed Agentcities project that created a global network of FIPA-compliant platforms and agent application services.

vi. An agent-specific extension of Unified Modelling Language (UML), known as Agent Unified Modelling Language or Agent (AUML).

2.4.3.2 The Central Concepts of FIPA

Several agent-related ideas were proposed during the evolution of FIPA, but the standardization specification task was mostly concerned with agent communication, agent migration and agent architecture.

A. Communication

Agents are distributed code processes that are made up of two parts, components and connectors, the components that produce and consume messages which are exchanged through the connectors. FIPA specifies languages, message sequencing, ontology, protocols, it specifies communication between agents and has an associated formal semantics in what is known as FIPA-Agent Communication Language (FIPA-ACL). FIPA-ACL is grounded in speech act theory which states that messages represent actions or communicative acts also known as speech acts or performative (Bellifemine *et al*, 2007). For an agent to be FIPA compliant it must be able to receive any FIPA-ACL communicative act message and at least respond with 'not understood' if it cannot process the message. FIPA communication can be separated into seven sub-layers as identified by Bellifemine *et al*., (2007), although O'Brien and Nicol (1998) identified five sub layers, within the OSI or TCP/IP application layer:

1. Transport: the lowest application sub-layer protocol is the transport protocol. FIPA defined message transport protocol (MTP) for IIOP (Internet Inter Object Request Broker Protocol) Inter and Intra-Operability Platform (IIOP, 1999), WAP and HTTP.
2. Encoding: FIPA defined several message representations for using higher-level data structures including XML, string and Bit-Efficient, instead of simple byte-encoded messages. Bit-Efficient is intended for use when communicating over low-bandwidth connections.
3. Messaging: in FIPA, message structure is specified independent of particular encoding to support flexibility. This is important because it is necessary to

specify certain key parameters in addition to the content to be exchanged such as the sender or receiver, message type, time-out for replies.

4. Ontology: defines the vocabulary and meaning of the terms and concepts used in the content expression. Individual term and concept contained in the content of a FIPA message can be explicitly referenced to an application-specific conceptual model or ontology. FIPA allows the use of ontology when expressing message content but does not specify any particular representation for ontology nor provide any domain-specific ontology. It is possible to reference Web-based ontology if required.
5. Content Expression: defines the grammar and associated semantics for expressing the content of a message. The content of FIPA messages may take any form, but FIPA has defined guidelines for using general logical formulae and predicates and algebraic operations for combining and selecting concepts. FIPA-SL (FIPA-Semantic Language) is mostly used for expressing contents with examples of logic formulae: not, or, implies, equiv etc and example of algebraic operators: any and all.
6. Communicative act (CA): defines the type of communication being performed, FIPA classifies messages in terms of its action or performative, implies, request, agree and refuse are examples of communicative act.
7. Interaction protocol: defines the social rules for structuring the dialogue between agents. FIPA defines several interaction protocols specifying typical message exchange sequences such as request, which describes one party making a request of another which in turn should agree or refuse to comply. Interaction protocols provide structure for the dialogue between agents while others are contained in each FIPA ACL message (O'Brien and Nicol, 1998).

B. Agent management

Agent management is a framework within which FIPA compliant agents can exist, operate and be managed. It establishes the logical reference model for the creation, registration, location, communication, migration and operation of agents (Bellifemine *et al*, 2007). The agent management reference model consists of agent that resides in an agent platform, with a number of services it can render; each agent has an identity AID, by which it is uniquely identified. The agent platform has an agent management system

that contains the descriptions of agents and one or more directory facilitators that contain the description of agent services. These components are described below:

Agent Platform (AP): provides the physical infrastructure in which agents are deployed. It consists of FIPA agent management components, the agents and other support software. One agent platform may be spread across multiple computers.

Agent: resides in an agent platform and offers computational services that can be published as a service description. Agent has one owner and an identity which can be described by FIPA agent Identifier (AID) that labels an agent so it is distinguished from the others.

Directory Facilitator (DF): is an optional component of an agent platform that provides yellow page services to other agents. DF maintains an accurate, complete and updated list of agents and must be able to provide current information about agents in its directory without bias to all authorized agents. An agent that wishes to publicize its services finds a suitable DF and requests registration of its description. The DF is committed to the act of registering and not the agent, the agent can request deregistration of a description should the DF fail to broker information relating to the agent. The agent may request the DF to modify its agent description at any time, or issues a search request to a DF to discover descriptions matching supplied search criteria.

Agent Management System (AMS): is responsible for managing the operation of agent platform such as creation and deletion of agents, controlling the migration of agents to and from the platform. All agents register with the AMS in order to get an AID, the AMS keeps these AIDs as directory of all agents present in the platform and their current state. The AMS can request any agent to perform specific management function and could enforce the operation if the request is ignored. Only one AMS can exist on a platform and if the platform spans multiple machines, the AMS is the authority across all the machines.

Message Transport Service (MTS): is a service the agent platform provides to transport FIPA-ACL messages between agents on any given platform and between agents on different platforms. FIPA compliant message is presented in a transport envelope and comprises of the encoded message (payload) that consists of the message content and message parameter such as the sender and recipient of the message.

C. Agent Abstract Architecture

An abstract agent architecture was created and standardized as a means to avoid the impact on platform implementations of incremental revision to main specifications. The most important mechanisms such as message transport and directory services are abstracted into a unified specification. The goal of this approach is to permit the creation of systems that seamlessly integrate within their specific computing environment while interoperating with agent systems residing in separate environments (Bellifemine *et al*., 2007). The architecture defines how two agents can locate and communicate with each other by registering themselves and exchanging messages. Agents communicate by exchanging messages which represent speech acts and which are encoded in an agent communication language. Services provide support services for agents and include the standard services of agent directory services, message transport services and service directory services. FIPA Abstract Architecture specified agent message structure, message transport service, agent directory services and service directory services.

The first official specification proposal of Process Documentation Template was released in 2010, revised in 2011 and final version was released the same year which was later approved in the year and became the IEEE FIPA Experimental specification document XC00097A (Cossentino, 2011).

2.5.3.3 Mobile Agent Systems Interoperability Facility (MASIF)

MASIF is the first standard for mobile agent systems which was adopted by the Object Management Group (OMG) in 1988 (Braun and Rossak, 2005). MASIF addresses the interface between agent systems, it defines agent system as a platform that can create, execute, transfer and terminate agents. MASIF is a collection of definitions and interfaces that provides an interoperable interface for mobile agent systems (Milojicic *et al*, 1998). MASIF aims at achieving interoperability between agent platforms written in the same language but by different vendors. According Braun and Rossak (2005) MASIF focuses on standardizing three things and Milojicic *et al*, (1998) believe they are four:

i. Agent management: manages different systems through the included standard operations such as creating agent, suspending, resuming and terminating agents
ii. Agent transfer: provides a common infrastructure for agent's applications to freely move mobile agents among different but compatible agents systems.

iii. Names for agents and agent systems: mobile agents cooperate with other agents and agent systems, there is therefore a need for standardized syntax and semantics to identify one another and communicate.

iv. Agent system type and location syntax: the agent transfer cannot happen unless the agent system type can support the agent. The location syntax is standardised so that agent system can locate each other (Milojicic *et al*, 1998).

MASIF however, does not cater for agent communication because it is addressed by CORBA (Common Object Requst Broker Architecture) extensively (Braun and Rossak, 2005). Very few platforms are MASIF compliant because of its closeness to CORBA.

MASIF has three components, CORBA services, Mobile Agent Facility (MAF) and Object Request Broker (ORB). CORBA provides the common services, ORB provides the communication infrstructure while MAF defines the fundamental functions of a mobile agent system which includes MAF Finder that registers and unregisters agents and agent systems and locate agents and agent systems dynamically and MAF Agent System that requests and provides services (Milojicic *et al*, 1998). Request services could be data, classes, creating agents, moving agents out and services provided could be transfering agents, creating agents, assigning authority, authentication, terminating or resuming agent running, transfering data, classes forwarding data or classes to agents, processing received data or classes.

2.6 Multi-Agent Systems

The explosion of available information in widely dispersed location, and distribution of enterprise operations have given rise to distributed problems that require distributed computations. Distributed computations are often easier to understand and develop than complex centralized computation that is adapted to solve distributed problems, at other times; centralized computation may not even be possible. Especially, when the necessary information is stored in large and complex systems that are geographically dispersed, manipulating such complex systems will require intelligent and distributed techniques. Mobile agents have been adopted as one such technique. Advance in the complexity of information systems increase the problem solving scope which single

agents could no longer cope with, this led to building systems that consist of multiple agents.

Multi-agent systems have been defined by Sycara (1998), as a loosely coupled network of software agents that interact to solve problems that are beyond the individual capabilities or knowledge of each problem solver. These problem solvers are agents that are autonomous and heterogeneous in nature. According to Shoham and Leyton-Brown (2009), multi-agent system consists of multiple autonomous entities with different information and diverging interests. In essence, multi-agent system consists of a number of autonomous agents interacting with one another. It is commonly agreed upon in the agent research community that a community of agents working together performs better than a single intelligent agent. The need to use agent technology to build sophisticated software to solve complex problems of the real world led to the use of multi-agent systems. Multi-agent system emphasizes that the agents concerned are computational information entities. For these agents to successfully interact, they will require the ability to communicate, cooperate, coordinate and negotiate with one another. This is the focus of Distributed Artificial Intelligence (DAI).

2.6.1 Motivation for Multi-Agent System

The motivations for the increase in multi-agent research identified by researchers are numerous and diverse, but most (Sycara, 1998; Shoham and Leyton-Brown, 2009) agree on the ability of multi-agent system to perform the following:

- to solve problems that are too large for a centralized agent to solve, because of resources limitations or the shear risk of having one centralized system that could be performance bottleneck or could fail at critical times.
- to allow for the interconnection and interoperation of multiple existing legacy systems
- to provide solutions to problem that can naturally be regarded as a society of autonomous interacting components –agents.
- to provide solutions in situations where expertise is spatially and temporally distributed
- to provide solutions that efficiently retrieve, filter, use and globally coordinate information from sources that are spatially distributed.

- To enhance overall system performance in terms of: (i) computational efficiency (ii)reliability (iii) extensibility (iv) robustness (v) maintainability (vi) responsiveness (vii) flexibility (viii) reuse.

2.6.1.1 Characteristics of Multi-agent Systems

Sycara (1998) identified four major characteristics of multi-agent system as follows:

- Autonomy: each agent has incomplete information or capabilities for solving the problem and thus has a limited view point.
- There is no system global control
- Multi-agent Systems are generally open and data are decentralized
- Computation is asynchronous

There are two categories of multi-agent systems depending on the mode and level of interactions of the component agents; these are cooperative and competitive multi-agent systems.

Cooperative Multi-agent system: the agents share the same desire and pursue common objectives in an unknown environment.

Competitive Multi-agent system: the agents cannot be assumed to have same desire or objectives, even in situations where agents are best off cooperating, they may not realize it or may not behave as if their interests are aligned.

2.6.1.2 Application of Multi-Agent Systems

Multi-agent systems have been applied in a wide area of human endeavour, some of which identified by researchers such as Sycara (1998) are highlighted below:

- ✓ HealthCare system: multi-agent systems have been used for patients scheduling and management, senior and community care, medical information access and management, and decision support.
- ✓ Telecommunication: multi-agent systems have been used effectively for managing distributed networks and realization of advanced telecommunication services.
- ✓ Traffic and transportation: for distributed vehicle monitoring and air traffic control and monitoring, such as OASIS which is a sophisticated agent air traffic control system based on the Believe-Desire-Intention model.

- ✓ Industrial applications: multi-agent systems have been successfully applied to industrial process control, system diagnosis, manufacturing, transportation logistics and network management.
- ✓ Information management: multi-agents are used for searching and filtering mass information from the Internet, electronic commerce and automated business processes, business process management, supply chain management.
- ✓ Multi-Robotic systems: multi-agent systems are used for distributed planning techniques for coordinating different robot, for example in MISUS multi-agents are used for scientific exploration with cooperating rovers.

2.6.1.3 Advantages of Multi-Agent Systems

Multi-agent systems offer a number of advantages that make them more desirable than the single agent systems, some of these advantages are highlighted below:

- ✓ Distributes computational resources and capabilities across a network of interconnected agents.
- ✓ Allows for the interconnection and interoperation of multiple existing legacy systems.
- ✓ Multi-agent system models problem in terms of autonomous interacting component (agent), which proves to be a more natural way of representing task allocation, team planning, user preferences, open environment and so on.
- ✓ Multi-agent system performs at a relatively high speed
- ✓ It allows reuse of components (agents)
- ✓ It is highly reliable.

2.6.2 The Agent Framework Design

The proposed system is a multi agent system with the agents designed using the itinerary pattern. There are basically eight kinds of agent design patterns: itinerary, branching, star-shaped, master-slave, MoProxy, meeting, Mutual Itinerary recording and facilitator (Emerson *et. al.*, 2003). Aridor and Lange (1998) identified nine patterns that were grouped conceptually into three, and Danny (2008) also agrees with the classification, the classes are travelling, task and interaction. The travelling patterns include Itinerary, Forwarding and Ticket. In the task patterns two specific patterns are

identified, Master-Slave and Plan. A set of five patterns identified within the Interaction patterns are Meeting, Facilitator, Locker, Messanger and Organized group.

The basic idea of design patterns according to Emerson *et. al*., (2003) is to define general solution models for common problems that are found in a given context. Design patterns make the development of applications easier, present a better understanding of the project, increase flexibility and promote reuse of components (Ajay *et al*, 1999: Silva and Delgado, 1998; Emerson *et. al*., 2003).

I. Itinerary pattern

Itenerary pattern provides a way to execute the migration of an agent, which will be responsible for executing a given task in remote hosts (Emerson *et. al*., 2003). The agent will create the itinerary object and initialize it with a list of destinations to visit sequentially and a reference to the agent. The agent will dispatch itself to the next available destination in its itinerary or back to its origin. The agent receives an itinerary on the source agency indicating sequence of agencies to visit. Once in an agency, the agent executes its task locally and continues on its itinerary. After visiting the last agency on its itenerary, the agent goes back to its source agency with the result (Aridor and Lange,1998; Danny, 2008).

The itinerary pattern is used when there is a need to hide the details of an agent's tour from its behavior in order to promote modularity of both parties. It is also used when there is need to provide a uniform interface for sequential traveling of agents to multiple hosts and to define tours that can be shared by agents (Emerson *et. al.*, 2003).

The itinerary pattern was chosen for this study because of certain attributes inherent in it, among these are the fact that it supports variations in routing, it makes it easy to provide such variations by simply replacing one itinerary object with another or by defining itinerary subclasses, the agent class however, is not modified. Itimerary patterns also facilitate sharing of tours by different agents. This is made possible by sharing itinerary objects, although not simultaneously, for example, two agents may use the same tour to multiple users' desktops with different messages. The itinerary pattern also simplifies the implementation of sequential tasks, tasks can be encapsulated in special task objects while an itinerary class is extended with an interface to associate task objects with destinations. The itinerary object keeps track of the current tasks to perform. When the agent is dispatched to a new destination, it simply triggers the execution of the current task saved by its itinerary object.

II. Branching Pattern

The agent receives a list of agencies to visit and clones itself according to the number of agencies in the itenerary. Then each clone will visit an agency of the received list, execute its corresponding task and notify the source agency when the task is completed. This pattern splits the tasks that can be executed in parallel (Emerson *et. al*., 2003).

III. Star-shaped pattern

The agent receives a list of agencies it has to migrate to. It migrates to the first destination agency, executes a task, goes back to the source agency. The agency repeats this cycle until it visits the last agency on its list (Emerson *et. al*., 2003).

IV. Master-Slave

The master slave pattern defines a scheme whereby a master can delegate tasks to a slave agent. A master delegates a task to be performed on a given agency to a slave agent, in order to continue executing other tasks that cannot be interrupted (Emerson *et. al*., 2003). Master-Slave also improves performance, a master agent can continue to perform other tasks in parallel with the slave agent. The master agent creates a slave agent, the slave agent visits the remote host, accomplishes the task and returns to the master with the results of the tasks (Aridor and Lange,1998). The master agent receives the results from the slave agent. The slave agent then destroys itself. The Master-Slave pattern is used when an agent needs to perform a task in parallel with other tasks for which it is responsible and when a stationary agent wants to perform a task at a remote destination. The Master-Slave pattern provides a fundamental way to reuse code among agent classes. However, the behaviour of a slave agent is not dynamic, as it is fixed at design time. An agent cannot be transformed into a slave at runtime nor can a slave agent easily be assigned to perform new tasks.

V. MoProxy pattern

The MoProxy (mobile proxy) pattern is used to control access to a resource (Ajay *et al*., 1999). When an agent needs a resource, it asks the Resource Granter (RG), indicating the desired permissions. Then, the resource granter returns a mobile proxy

for the agent with desired access right on the resource, in order to access the resource with the desired permissions depending on the restrictions of the resources. This prevents the agent from directly accessing the resource. This pattern is unique in that the mobile proxy can move along with the agent. If the resource moves, the mobile proxy keeps track of the resource, it moves along with the agent and provides the agent with controlled access to the resource at any location. Thus mobile proxy provides location transparent controlled access to an agent.

VI. Meeting pattern

The meeting pattern presents a way to promote local interactions between agents distributed on the network(Emerson *et. al.*, 2003). Such interactions allow the execution of given tasks as well as the optimization of results. The meeting pattern uses the notion of meeting to synchronize various agents which were initially at different hosts, so they can visit a virtual place and find one another. The meeting agent, who will meet others, has a meeting object that keeps the place and the identification of the meeting. In this way the meeting agent requests from the meeting object the place of the meeting and then migrates to it. Each agent dispatches itself independently to the meeting place, where it will use the unique identifier to locate a specific local meeting manager object to register itself (i.e add itself to a list of agents that have arrived at that host). A meeting manager that manages the meeting notifies the agents already registered in the meeting place about the arrivals and departures of new ones (Aridor and Lange, 1998). Departing agents unregister themselves before leaving the meeting place. The meeting object intermediates the register of the agent on the manager. Through the use of unique identifiers, multiple meetings can take place simultaneously at a single host. Meeting objects can be distributed by messages or located in central directories. The meeting pattern is used where there is a need for agents on different hosts to interact locally, and the overhead of travelling to a central place and interact locally is less than associated with remote communication. It is also used when agents cannot interact remotely, since they are located behind firewalls or on hosts with unreliable and low-bandwidth network connections, or if their origins are disconnected for the network, e.g. laptops. One solution is for these agents to dispatch themselves to a remote host where they can interact more efficiently. Meeting pattern is also used when agents need to access local resources on a given host, in this case the agents need to interact locally with the service provided by a given host.

VII. Facilitator pattern

Defines an agent that provides a name service and localization of agents with specific abilities thus facilitating the localization of a given agent (Emerson *et. al*., 2003). It is often convinient to assign a symbolic name to an agent in order to locate it on a later occasion.

VIII. Mutual Itinerary Recording

Is a general schema that guarantees the itinerary of a given agent will be registered and tracked by other cooperative agent and vice-versa, in a disposal of mutual support (Emerson *et. al*., 2003). When an agent is moving between platforms, it carries the information from the last platform, the current and the next ones to the cooperative agent through an authenticated channel. The agent keeps the register of the itinerary and it always compares the itinerary that it possesses with the received one. When an inconsistence is detected it should be treated . e.g it would either disallow the agent to visit the platform that caused the inconsistency or suspend the functioning of an agent or send the agent back to the source agency.

The use of mobile agent design patterns presents solutions that can be reused, avoiding loss of time and efforts to investigate problems that have already been solved.

2.6.3 Mobility Strategies

There are two types of agent migration proposed in the existing mobile agent platforms depending on what is transported with the agent code, these are strong and weak mobility. The state of a mobile agent is divided into two parts, the runtime state which contains all information for the control of a mobile agent such as program counter and stack, and the data state which contains information such as the local variables and resources (Genco, 2008). Both strong and weak migration save states and later restore the state previously saved before the agent starts execution.

Strong migration: the agent migrates with its execution state, both runtime state and data state are saved and transported to the destination with the agent's application code . After migration, it can re-obtain the state before migration and resume execution from

the point where is was suspended. D'Agent (Agent TCL) implements strong migration because of the Tool Command Language it uses. It uses agent_jump statement for migration and execution continues with the statement following the agent_jump statement.

Weak migration: the agent migrates with its code and transferable resources, i.e only the data state is saved and transported to the destination. On the destination node, the agent is launched from the main application and it re-executes from the beginning and its execution context is re-initialized. This work like other Java-based agents, implements the weak migration strategy in which the migration is performed by executing the runAndMove() method inherited from the MobileAgent class.

2.6.4 Mobility Patterns

Itinerary is a set of sites that a mobile agent has to visit. Itinerary of an agent can be static i.e fixed at the time of agent initialization or dynamic in which case it is determined by the Mobile agent itself. Order is the order/sequence in which a mobile agent visits sites in its itinerary, order could also be static or dynamic. Based on these two concepts, the mobility patterns of an agent can be categorized into the following:

i. **Static itinerary:** The itinerary of the mobile agent is known a priori and cannot change. Static itinerary can further be classified based on their order as

ii. **Static Itinerary Static Order (SISO)**: the order in which an agent completes its itinerary is static and known a priori, as illustrated in Figure 2.14. Applicable implementation strategies include sequential client server, sequential mobile agent. An example is an auction agent that is required to visit a set of auction sites in a specified order.

iii. iii **Static Itinerary Dynamic Order (SIDO):** the order in which an agent completes its itinerary is dynamically decided by the agent, as shown in Figure 2.15. Available implementation strategies include sequential client server, Sequential mobile agent, parallel client server, parallel mobile agent. Example is a shopping agent which finds the cheapest price of an item from a set of on-line shops.

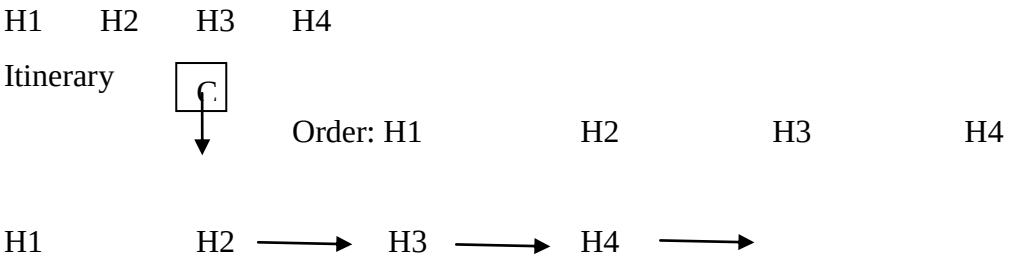


Figure. 2.14: SISO itenerary pattern

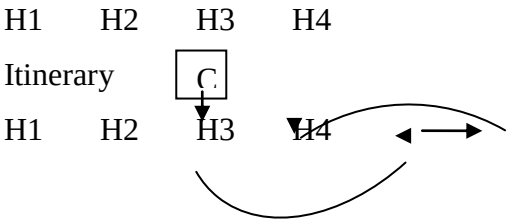


Figure 2.15: SIDO itinerary pattern

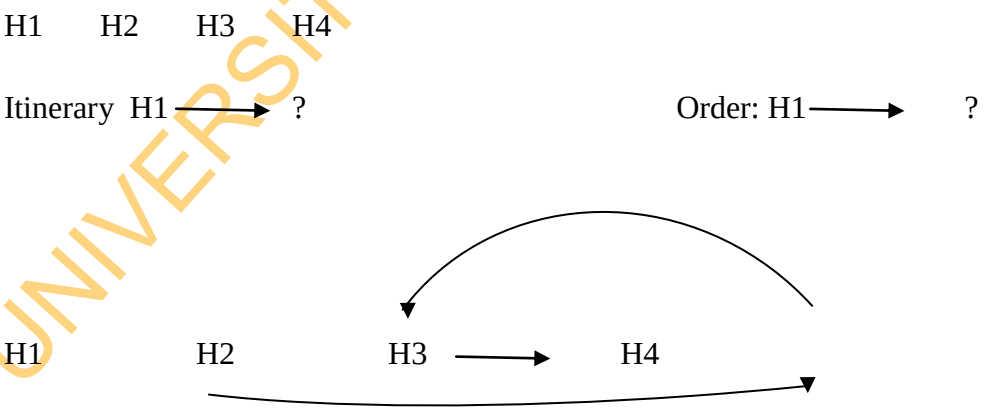


Figure2.16: dynamic itinerary pattern

iv Dynamic Itinerary

The itinerary of an agent is not known a priori but it is dynamically determined by the agent itself, however, the first site should be known a priori, illustrated in Figure 2.16. Available implementation strategies for dynamic itinerary are sequential client server, sequential mobile agent. Dynamic itinerary also implies dynmamic order. An example is a shopping agent that is required to find a particular product, a shop that does not have the product can recommend another shop and the recommended shop is included in the mobile agent's itinerary dynamically.

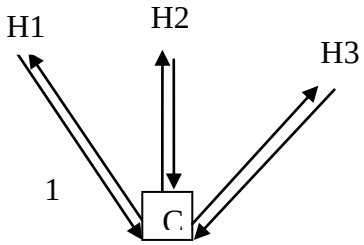
2.6.5 Itinerary Implementation Strategies

Sequencial Client Server: is based on the traditional client server paradigm. The client makes a request to the first server returns with the result of processing to the client, makes a request to the second server and returns to the client and so on until the list of servers in its itenerary is exhausted, as shown in Figure 2.17 (a).

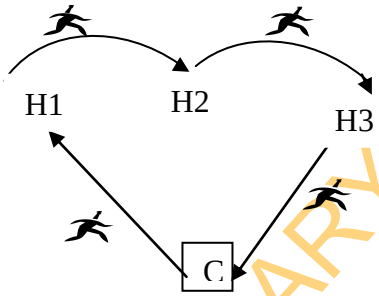
Sequential Mobile Agent: only one mobile agent moves from its sourcce to the first host in its itinerary and then to the second host and so on, as shown in Figure 2.17 (b), until it visits all the hosts in its itinerary and then returns with the result to the source host.

Parallel Client Server: is also based on client server paradigm. The client initiates parallel threads of execution instead of sequnetial requests, where each thread concurrently makes a request to one of the servers and processes the reply as shown in Figure 2.17 (c).

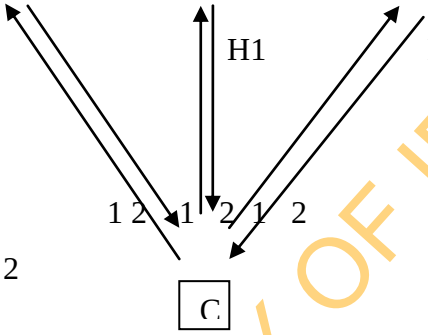
Parallel Mobile Agent: with this approach, the client initiates multiple Mobile Agents, each of them visits a subset of the servers in the itinerary, as shown in Figure 2.17 (d). The Mobile Agents each return to the client and pull their results together to complete the task.



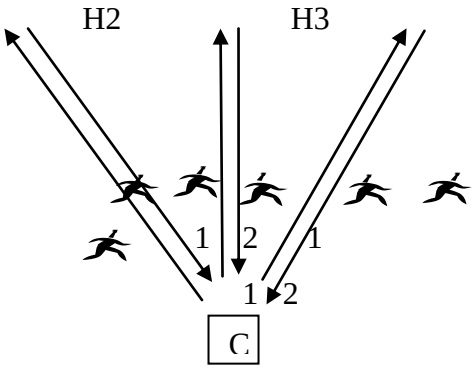
(a) Sequential Client Server



(b) Sequential Mobile Agent



(c) Parallel Client Server



(d) Parallel Mobile Agent

C: Client H: host computer

: Mobile Agent Message Exchange/MA movement

1 2 3..... Sequence of messages / MA movement

Figure 2.17: Itinerary implementation strategy

2.7 A Survey of Mobile Agents Systems

This section examines the most common Mobile Agents Systems currently available and attempt to describe them as brief as possible in detail.

2.7.1 JADE: Java Agent Development Framework

JADE is a software platform that provides basic middleware-layer functionalities, it contains two parts: a platform for agent following FIPA standard, and a software package for java agent development. JADE according to David, 2004: Bellifemine *et al*, 2007: and Giovanni, 2009, is an open source FIPA compliant agent platform that supports implementation and desktop deployment of software agents using an extensive developer's toolkit and compatibility with standard and heavier java environment. The main objective of JADE is to simplify the development of agent applications in compliance with the FIPA specifications for interoperable intelligent

A significant merit of JADE is that it implements this abstraction over a well-known object-oriented language, java, providing a simple and friendly Application Programming Interface (Bellifemine *et al*, 2007). JADE like other agent platforms, provides services necessary for agents execution in the network, allows agents to dynamically discover other agents and communicate with one another. JADE supports mobility of code and execution state, as well as complex conversations by providing a set of skeletons or typical interaction patterns to perform specific tasks, such as negotiations, auctions and task deligation.

JADE platform is composed of agent containers that can be distributed over the network. Agents live in containers which are the Java processes that provide the JADE run-time and all the service needs for hosting and executing agents. The set of active containers is called platform, there is main container which is the first container to be lunched and other containers are launched from the main container (Bellifemine *et al*, 2007: Giovanni, 2009, and Wenjuan *et al*, 2009). Agent Management System (AMS) and Directory Facilitator (DF), are parts of the platform, there is only one AMS on one platform, which offers the white page service and agent lifecycle service, and maintain the directory of agent identifier (AID) and the state of agent.

To process a mobile agent's request to move, JADE executes the following algorithm

(i) Suspend the agent (halt the agent's thread)

(ii) Identify the agent's state

(iii) Serialize the agent class and state

(iv) Encode the serialized agent class and state according to the transport protocol

(v) Provide authentication information to the server

(vi) Transport the agent

Before accepting the agent, the receiving agent system verifies that it can support the agent profile (agent system type, language and serialization method). If it does, the following algorithm is executed

(i) Authenticate the client sending the agent

(ii) Decode the agent

(iii) Deserialise the agent class and state

(iv) Instantiate the agent

(v) Restore the agent state

(vi) Resume agent execution.

For communication and transport JADE implements all standard Message Transport Protocol (MTP) defined by FIPA, HTTP, HTTPS, IIOP, JMS (Java Message Service) and Jabber XMPP (Extensible Message and Presence Protocol) and other proprietary protocols like the IMTP (Internal Message Transport Protocol).

2.7.2 Grasshopper

Grasshopper is the first mobile agent platform that complies with MASIF and FIPA standards developed by IKV Technologies. It is being continuously developed since 1997 and is offered as a free product for educational use. The entire platform is implemented in Java, the environment is built from one region registry and several agencies, which can be compared to agent platforms in agent system (Wallin, 2004). Grasshopper contains several services which are security, registration, persistence, management, transport and communication, it also provides developers with interesting features, including graphical user interface to manage agents, agencies and region. Grasshopper supports a number of transport and communication protocols, with the default proprietary protocol based on TCP/IP, and it also has supports for RMI/JRMP and CORBA/IIOP. Agent mobility in grasshopper is weak code mobility, thus agent state is restarted each time it migrates to a new location. Grasshopper uses Secure Socket Layer (SSL) protocol to protect agents during migration, certificates in

authentication and cryptographic algorithms for encrypting the data packets under transmission (Wallin, 2004). However, a call to an agent which is moving can end up executing on the copy of the agent at the origin, which will be removed once the agent arrives at its destination (Gupta and Kansal, 2011).

2.7.3 Agent Tcl (D'Agents)

Agent Tcl system was developed at Dartmouth College to address the weaknesses of existing mobile agent systems (Syed *et al.*, 2000) and it focuses on five research areas, these are (i) performance, (ii) support for multiple languages (iii) cryptographic authentication and restricted execution environments to protect a machine from malicious agents, (iv) economic-based models to limit an agent's total resource consumption across multiple machines and (v) networking sensing, navigation and planning services (Gray, 1997). Agent Tcl is an agent support environment, the platform is written in C programming language and the agents are written in TCL (Tool Command Language). Agent Tcl is similar to other mobile agent systems but it distinguishes itself with it combination of multiple languages (Tcl, java and scheme), a simple but effective security model and its simple migration mechanism (agent_jump) and both low and high-level communication protocols. Agent TCL supports strong migration with the agent_jump command which captures the current state of the agent and transfers this state to a server on the destination host and execution continues from the command immediately after the agent-jump command on the destination host (Gray *et al*, 1996). Agent Tcl has a simple but effective security model, to protect migrating agents and for authentication, it uses PGP (Pretty Good Privacy) for its digital signatures and encryption (Gray *et al*, 1996), for transport and communication, it uses a proprietary protocol over TCP/IP. Agent TCL has been used in information management and information retrieval applications (Gray, 1995).

2.7.4 Aglet

The Aglet framework, developed by IBM (Tokyo research Laboratory) is a Java based framework for mobile agents (Venners, 1997). An aglet is a combination of the applet model and the agent model, in principle adding mobility to applets. Aglets' transport and communication is based on Agent Transfer protocol, which is modelled over

HTTP protocol. Protocol capabilities are accessed through the Agent Transfer and Communication Interface (ATCI), which allows an abstraction from the underlying transport protocol. With Aglets, Java objects can be constructed that can move from one host on the network to another. That is, an aglet that executes on one host can suddenly halt execution, dispatch to a remote host, and start executing again. When the aglet moves, it takes along its program code as well as the states of all the objects it is carrying. Aglets have methods for creation, running, suspension, waking up, activating, deactivating, cloning, dispatching to other hosts, retraction from other hosts etc. Along with this it has methods where one can specify what the agent should do when it arrives at a particular host.

2.7.5 TACOMA: Tromso And COrnell Moving Agents

TACOMA is a joint project of the University of Tromsø (Norway), and Cornell University (New York) (Johansen *et al*., 1995; Outtagarts, 2009). Agents are written in Tool Command Language (TCL), although they can technically carry scripts written in other languages too, the current TACOMA supports Scheme, Perl, Python, Java and C, TCL, though TCL remains the principal language. The TACOMA project's main objectives were to investigate what services need to be provided in order to support easy building of agents and how agents can be used to solve problems by Operating Systems in the previous TACOMA versions. The basic concepts of TACOMA include the following

- Agent: the computational or execution unit
- Folder: list of element each of which is an uninterrupted sequence of bits.
- Briefcase: collection of named folder (mobile agents carry them as they migrate)
- Cabinet: collection of named folder statically stored in a given machine. Cabinet are used to permanently store information in a given machine, so agent can leave state behind in when it moves around.
- Meet: allows an agent to run another agent passing a briefcase as a parameter.

An agent's state is explicitly stored in folders, which are aggregated into briefcases. An agent is created by packing the program into a distinguished folder called code. Next, its intended host's name is stored in the host folder. Absolute migration to this destination is requested using the *meet* primitive. A briefcase containing the code, host

and other application-defined folders is sent to this agent. Agents can also use the *meet* primitive to communicate by co-locating and exchanging briefcases. Both synchronous and asynchronous communication is supported. An alternative communication mechanism is the use of cabinets, which are immobile repositories for shared state. Agents can store application-specific data in cabinets, which can then be accessed by other agents. Tacoma offers directory services through broker agents. These agents maintain a database that provides services by which each agent can deal with other agents. Tacoma does not provide automatic state migration rather agents need to capture the internal state by themselves.

2.7.6 Telescript/Odyssey

Telescript is a framework and a scripting language for implementing mobile agents developed by General Magic Inc. one of the inventors of the concept of mobile agents. Telescript is an object-oriented programming language in which state-oriented migration is seen as the basic operation which is provided by the *go* instruction and a *ticket* argument that determines the destination site in "varying levels of specification" (General, 1995). A Telescript *engine* exists at each site to accept and authenticate migrating agents and to restart the execution of agents at the statement immediately after the *go* command.

A telescript program consists of a collection of classes. These classes have properties which are either operational or attributes and can be either private or public. The three major concepts in the language are *agents, places* and *go*. *Agent* and *places* are processes; *agents* are mobile while *places* are stationary. *Agent Go* to *Places*, where they use places, services and or interact with other agents that are in the same place. An agent always executes in the context of one or more enclosing place, places provide a service for agent to interact with, places can be nested. Two telescript agents can meet in the same place. Meeting agents can call each other's operations and meetings motivate agents to migrate. They permit all interactions between two agents to be done locally rather than remotely. Security is paramount in telescript, the server needs protection from malicious agents, agents' information must be protected as they travel from host to host. Each place has its own policies and each engine has an overall security policy. The agent transfer is authenticated using RSA and encrypted using RC4.

Two concepts of telescript security are safety and security. Safety refers to features that promote robustness and prevent accidents and security refers to features that provide protection and integrity in the presence of malicious users. These security features protects agents and places from each other. Every agent is uniquely identified by a telename. A telename consists of an authority which identifies the owner of the agent and identity which distinguishes an agent from another agent of the same authority. Authority component is cryptographically generated and cannot be forged.

Telescript shares the concept of remote programming with Java, and when Java became successful Telescript was re-implemented in Java and renamed Odyssey. Odyssey uses the same travel metaphor as Telescript, and the agents themselves are programmed in Java. One notable exception is that Odyssey does not have the *go* instruction, since Java does not provide facilities for capturing an executing program's complete state, making it impossible to implement the *go* instruction without modifying the Java virtual machine. An Odyssey agent carries all of its objects along with it, but it must either restart execution on the destination machine or follow an itinerary in which specific methods are executed at specific destinations.

2.7.7 Voyager

Voyager is a java-based and agent-enhanced Object Request Broker (ORB), developed initially by ObjectSpace Company in 1997 and currently by Recursion Software Incorporated. The goals of voyager include enabling programmer to create state of art distributed programs quickly and easily, while providing a lot of flexibility and extensibility for the products that are being created with the voyager system (Syed *et al*., 2000). Voyager is a commercial product with a free license allowing non-commercial use of its core technology. It is entirely programmed in java and fairly simple to use, it uses java syntax to create remote objects and move them between applications. There are two main types of objects in voyager: applications (which act as hosts) and agents (which can move between them), exchanging messages (Michael and Takanori, 1997). Both applications and agents can instantiate objects or call methods on remote hosts. The transport and communication is based on a proprietary ORB on top of TCP/IP. One great advantage of voyager is its supports for both traditional client server architecture and agent-based architecture.

2.8 Windows XP Operating System

Operating system is a suite of software that controls and manages the hardware resources, all the activities of the computer machine and provides common services for computer programs. The structure of an operating system consists of two major layers, the user mode and the kernel mode as shown in Figure 2.18. This study focuses on Windows XP, which is a 32-bit pre-emptive multitasking operating system for Intel microprocessors (Silberschartz *et al*., 2009). In Windows XP, the user mode is the layer closest to the users, it consists of the applications that users run and support programs for application. The kernel mode is the layer closer to hardware and it consists of programs that help all the software running on the computer system to use the hardware as well as device drivers (WIN133, 2009). Windows XP was chosen for this study because it was created to work on almost all hardware platforms (Silberschartz *et al*., 2009). Windows XP incorporates a Hardware Abstraction Layer (HAL) which sits between the XP and the hardware, XP interacts with the HAL who directly interacts with the hardware to perform any required hardware functions; the XP kernel sits on top of the HAL (WIN133, 2009 and Silberschartz *et al*., 2009). The kernel mode programs are trusted programs that perform privileged activities with the computer's hardware. At the heart of kernel is the Microkernel which provides the building blocks for all the Executive Services running in the Kernel mode. The executive services on the other hand provide services for applications and together with the microkernel make up the Windows XP kernel. Windows XP provides avenue for extending the services for the executive services, in other words, applications can be made available and embedded in the kernel of XP through the executive services that will give an impression of programming the XP directly.

2.8.1 Windows Operating System Service

Operating System Services are long-running executable programs that run in its own Windows session, without users' intervention, this is similar to Deamon in Unix Operating Systems. They give the impression of configuring some part of the PC and are managed by the Service tool in Windows Operating System where they can be enabled and disabled. The service does not support user interface but Windows operating system provides an opportunity for developer to build a user interface in it.

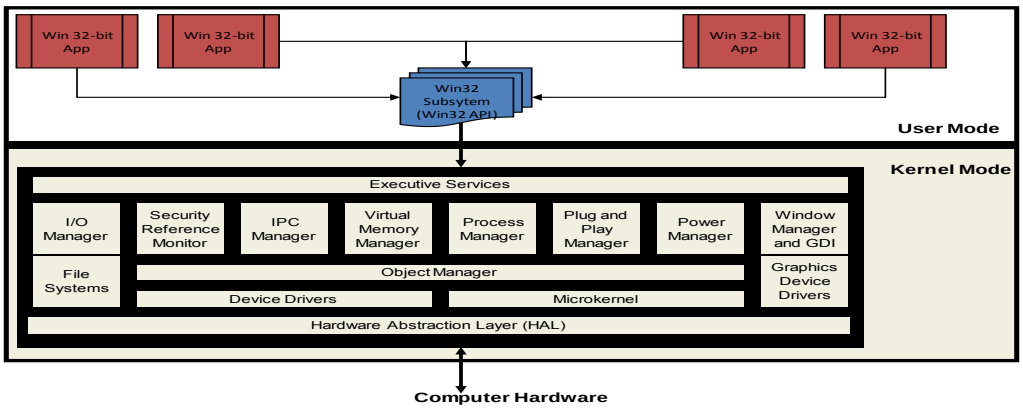


Figure 2.18: The Architecture of Windows XP (adapted from WIN133, 2009)

These services can be automatically started at the boot of the computer, can be stopped, paused or restarted without interfering with other users or applications.

Customised services are provided for Windows Operating System by setting up the inheritance and other infrastructure elements. The service class is set up to inherit from the ServiceBase class and a main method for the service project that defines the services to run and calls the Run() method on them. Service must be created in a Windows service application project that creates a **.exe** file when built and inherits from the ServiceBase class. ServiceBase class provides a base class for a service that will exist as part of a service application. In java programming language this class is referenced by:

Public class Servicebase extends Component

A service can start manually or automatically. Windows provides an interface called the Service Control Manager which manages the starting and ending of services and projects containing Windows service have installation components that can handle messages from the Service control Manager. The operating system service provided in this work starts automatically at the boot of the computer on which it resides, it remains active in the background until it is manually stopped or paused or the system shuts down.

2.8.2 Peculiarity of Service Applicationn

Service application is distinct and it distinguishes itself from other applications running on the computer by the following features:

(i) The compiled executable file that runs as service application project must be installed on the server before the project can function meaningfully.

(ii) Installation components need to be created from service application

(iii) The main method for the service application must issue the **Run** command for the services the project contains. The Run method loads the services into the service control manager on the appropriate server. This method is generated automatically from windows service project template.

(iv) Windows service applications run in a different window station than the interactive station of the logged-on user.

(v) Windows service applications run in their own security context and are started before user logs into the windows computer on which they are installed.

2.8.3 Service Control Manager (SCM)

SCM is a Windows System process that starts, pauses, stops and interacts with Windows service processes. It maintains a database of installed services and driver services, and provides a unified and secure means of controlling them. The database includes information on each service, how each service should be started and thereby controls access to the service. SCM is located in %SystemRoot%\System32\services.exe executable and it starts at system boot. SCM is a remote procedure call server, so that service configuration and service control programs can manipulate services on remote machines. The main class involved in service creation is defined as

System.ServiceProcesss.ServiceBase()

This class overrides methods from the ServiceBase class when creating a service and define the code to determine how the service functions in this inherited class.

System.ServiceProcess.ServiceProcessInstaller () this class install the service and

System.ServiceProcess.ServiceInstaller() unistalls the service

2.8.4 Windows XP Service Lifecycle

As mentioned earlier, Windows Service is a long executable program running in its own Window session without user's intervention. Figure 2.19 describes windows XP lifecycle, the Windows Service is installed onto the system where it will run. This process executes the installer for the service project and loads the service into the Services Control Manager, the service is started. The start method passes processing to the application's OnStart method and processes the defined code. A service can run indefinitely untill explicitly stopped, paused or the computer shuts down.

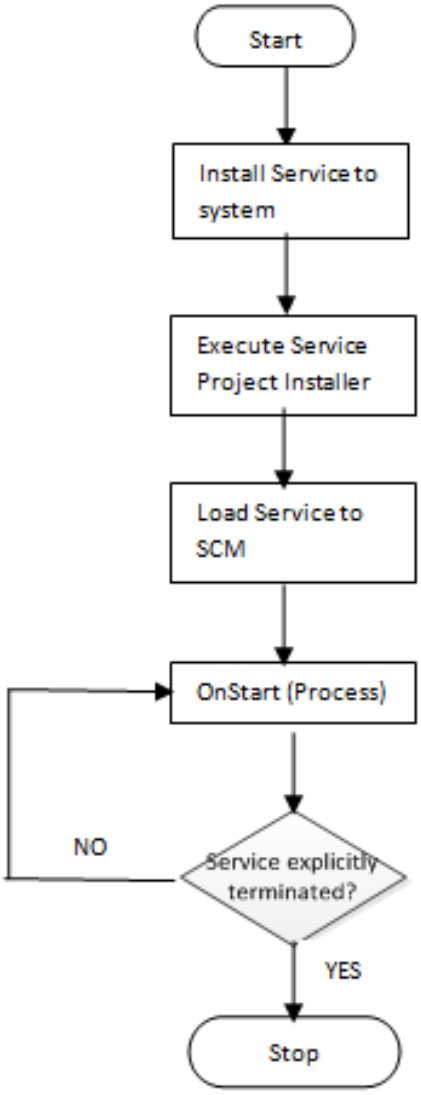


Figure 2.19: Windows Service Life Cycle

2.9 Introduction to Information Retrieval

This section examines information retrieval, its evolution over the years, concepts and principles with other information that is needed to understand the topic. Information retrieval deals with the recovery of documents from a document collection, for a given user information need expressed in form of a query. Information retrieval starts when a user issues a query i.e. a formal statement of his information need; the IR system evaluates the query with reference to information collection and provides the user with a set of data that answers the query.

2.9.1 Evolution of Information Retrieval

To have a good understanding of the evolution of information retrieval, a historical survey of the major technological milestones is necessary. Lesk (1996) used the analogy of Shakespeare's seven ages of man to describe and predict the evolution of information retrieval based on the prediction of Vannevar Bush. Bush in 1945 set a goal for fast access to contents of the old libraries which is expected to be achieved by 2010 (Lesk, 1996).

The childhood stage (1945 -1955), the idea was conceived due to information explosion after World War II. Bush predicted that a time will come when computers will be selecting their own data, perform complex arithmetic operations, record results for future use and so on. The first IR system was built in the 1950s, it used indexes and concordances.

The schoolboy stage occurred in the 1960s, many experiments were performed, the first experiment used mechanical searching of manual indexing, keying in the traditional indexes. The first large scale information system was built and the idea of free text searching arose, i.e. complete retrieval of any document using a particular word and no cost for manual indexing. The mathematics of recall and precision were developed as measures of IR systems. New retrieval techniques such as relevance feedback, multi-lingual retrieval were invented. 1960s also saw the beginning of natural language question answering and AI researches began building systems to retrieve answers instead of just documents. Most works in the 1960s were still research and learning, there was no access to really large amount of machine readable text to build large systems.

The invention of word processing system and time-sharing systems in the 1970s led to a lot of text in machine-readable form that started full text retrieval systems. As research progresses, there arose the probabilistic information retrieval model which involve measuring the frequency of words in relevant and irrelevant documents using term frequency measures to adjust the weight of words. The term weighting techniques improved the performance of IR systems over the simple word matching that was prevalent.

In the 1980s the use of online IR expanded with availability of full text instead of just abstracts and indexing and spread of outline retrieval into use by non-specialists. There was increasing interest in new retrieval methods such as sense disambiguation using machine-readable dictionaries and computational linguistics, and the statistical kind of retrieval.

In 1990s, things seemed to progress well, more text were available online with full text search algorithms for retrieval. The Internet put IR to test; everyone could access the Internet and provide information freely as well as classify their information. Information retrieval received a boom especially in the USA where it was suggested that computer network could bring information close students, and digital libraries were developed, this is the origin of distributed information retrieval.

The fulfilment of Bush's prediction came in 2000s, a lot of books are available online, and some ordinary questions can be answered by referencing online materials instead of printed materials. Research focused on multimedia retrieval i.e. retrieval of images, sound and video which led to more serious image recognition and sound recognition research that have been more promising than computational linguistics. New and improved retrieval systems were developed to multimedia information retrieval.

In the year 2010s, the fulfilment of Bush's prediction is being exploited, a lot of conversion to machine-readable forms have been done but not complete. Multimedia information is available and easy to deal with. Research now focuses on improving the IR systems and learning new ways to use the new IR systems.

2.9.2 Overview of Information Retrieval (IR)

Information retrieval deals with the recovery of documents from a document collection, for a given user information need expressed in form of a query. Information retrieval starts when a user issues a query i.e. a formal statement of his information need; the IR

system evaluates the query with reference to information collection and provides the user with a set of data that answers the query (van Rijsbergen, 1997). According to Hiemstra (2000), information retrieval system is a software program that stores and manages information on documents; the system assists users in finding the needed information and does not explicitly return information or answer questions. However, in IR, query may match several objects with different degrees of relevance, the set of result obtained are therefore, ranked according to the degree of relevance to the query. There is no perfect retrieval system that would retrieve only the relevant documents and no irrelevant documents; therefore measuring the degree of relevance of documents forms a vital part of IR. It is useless to have so much information that is not relevant; it is also not desirable to have unretrieved relevant information. The whole process of information retrieval is summarised by William (2007) as

"A full-text search engine takes a user's query q, consisting of discrete terms $\{t_1,.....t_{|q|}\}$*; evaluates it against a document collection D, consisting of documents* $\{d_1, \ldots . .,d_N\}$*; and answers it with a ranked list of documents* $\{a_1, \ldots . a_r\}$*,* $a_i \in D$*, ordered in decreasing estimated relevance to the query q."*

- Query is the user's request to the computer in an attempt to communicate the information need, and it is made up of distinct terms.
- Document is an item of whatever units we have decided to build a retrieval system over. Documents are data objects, usually textual, though may contain other types of data such as images, sound, video or mixed-media records.
- Document Collection is a group of documents over which retrieval is performed.
- Information need is the topic about which the user desires to know more.
- Ranked list of documents are members of the document collection that are found to be relevant to the user's query when a specific ranking algorithm has been applied.

A document is relevant if it is one that the user perceives as containing information of value with respect to their personal information need in other words, satisfying his information need. Information retrieval systems are evaluated based on two criteria, effectiveness and efficiency. An effective system returns results containing more relevant documents and an efficient system incurs reduced costs while finding result documents. The cost of a search according to Craswell (2000) includes several factors

such as computation or storage resources expended at client or server, network resources such as bandwidth expended in their communication and monetary network usage or per-search charges.

Most computer based retrieval systems store only a representation of the document (or query), this representaion is neccessary because it speeds up the retrieval process. Information retrieval systems must support three basic processes that are of major concern: (i) how to represent each document (indexing), (ii) how to represent the user's information needs in a form suitable for a computer to use (query formulation) and (iii) the matching or comparison of the two representations. The matching process usually results in a ranked list of documents with the most relevant documents towards the top of the list.

2.10 DISTRIBUTED INFORMATION RETRIEVAL (DIR)

In this information age, there is an influx of information available on different sites covering large geographical areas for ease of access, use and consistency. Distributed Information Retrieval includes searching document collections in these various sites for documents relevant to our information needs. Document collection can be a single source, a single location e.g a library or a set of libraries in a local environment or a wide-area environment like a corporate network or even the Internet. According to Craswell (2000), document can be full text, bibliography, sound, image, video or mixed-media records.

Existing information retrieval services centralize the indices in the servers such as Google, collect the content information on the web servers by using web crawlers. Web crawlers are computer programs that automatically browse the World Wide Web in a methodical and orderly fashion, they are also called ants, automatic indexer, worms, bots, web roborts or web spider. The process of scanning the WWW is called web crawling or spidering.

An individual or organisation wishing to publish electronic documents set up a document server, a user views such documents using a document client e.g a web browser. To view a document, the client sends a request containing a document identifier (such as Internet URL, keywords) and the document server returns the document in question if available (Craswell, 2002).

A distributed information retrieval problem arises when a user has access to many documents that are spread across several servers, as it is with the WWW, and requires some systematic organisation or search facility to find relevant information. It is possibile for a single IR system to request all documents from every document server and perform its search task over the combined set or to set up various search servers on the network, each covering documents from one or more document servers (Craswell, 2000). IR system available across the network is called a search server and it is accessed through a search client. Systems which return search results, such as search server, usually return to the user a results list ranked according to the order of their relevance to the user query. If the list contains more relevant documents, the system is more effective, and a system is more efficient if it involves reduced costs in finding the ranked list. Users want a system that is both effective and efficient. Queries are almost always less than perfect, they retrieve some irrelevant documents as part of the result list and don't retrieve all the relevant documents in the collection. It is useless retrieving so many documents that are irrelevant, at the same time relevant documents that are not retrieved are useless. This leads to the measure of the performance of the retrieval systems. Two major parameters are measured:

- Precision is the fraction of the documents retrieved that are relevant to the user's information need, it is called precision at n or P@n. if we are interested in raising precision, we need to narrow the query.

  $$\text{Precision} = \frac{|\{\text{relevant document}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant document}\}|}$$

- Recall is the fraction of the documents that are relevant to the query that are successfully retrieved. It can be seen as the probability that a relevant document is retrieved. In order to raise recall, we broaden the query.

  $$\text{Recall} = \frac{|\{\text{relevant document}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

2.10.1 Information Retrieval Model

IR systems store a representation of documents as well as query, there are several of such representations, each is helpful in developing specific information retrieval tools. The representation takes different forms which are categorized on mathematical basis consisting of set of theoretic models that represent document as sets of words or phrases, and derives similarities from set theoretic operations on those sets and

occurence of terms, the search term either occurs or not. Common models are Standard Boolean Model, Extended Boolean Model and Fuzzy retrieval. The Algebraic Models represent documents and queries as vectors, matrices or tuples. The similarity of the query vector and document vector is represented as a scalar value and measures the frequency of term occurrences. Common models are Vector Space Model, Generalised Vector Space Model, (Enhanced) Topic-Based Vector Space Model, Extended Boolean Model, and Latent Semantic indexing (Latent Semantic Analysis). The Probabilistic Models treat the process of document retrieval as a probabilistic inference, i.e. Probability of occurence of terms. Similarities are computed as probabilities that a document is relevant for a given query using the occurrence or non occurrence of terms. Common probabilistic model are: Binary Independence Model, Probabilistic Relevance Model, Uncertain inference, Language Models and Divergence-from-randomness Model. In the following section we examine some of these information retrieval models.

2.10.2 Boolean Retrieval Model

In Boolean retrieval model a query is posed in the form of a Boolean expression of terms, i.e. terms are combined with logical operators AND, OR, and NOT (Manning *et al.*, 2009). Boolean model views documents as just a set of words and retrieves a document if the expression of the query evaluates to true. Parenthesis can be used to group keywords to specify the order in which the Boolean operators are applied and also allows more complex query. The AND operator returns the intersection of the document and query, OR returns the union of the two and NOT considers all the document not in the specified set. The model keeps a dictionary of terms (vocabulary or lexicon) using the inverted index concept and for each term maintains a list that records which documents the term occurs in. Each item in this list is called a posting, the list is then called a posting list or inverted list and all the posting lists taken together are called postings.

2.10.2.1 Advantages and Disadvantages of Boolean Retrieval

The Boolean model is the first model for information retrieval and it ruled for a long while. It uses the basic Boolean operators that users are familiar and comfortable with,

the associated merits and demerits as identified by Yepes (2009) and Craswell (2000) include the following:

(i) Boolean model gives users a sense of control over the system.

(ii) It possesses a great expressive power and clarity; it is effective if a query requires an exhaustive and unambiguous selection. It is clear why a document has been retrieved given a query, and if the resulting set is too small or too large, it is clear which operator will produce a larger or smaller set.

(iii) Boolean model can also be extended with proximity operators and wildcard operator, which gives it the capability to compete with other models.

(iv) It is easy to implement and it is computationally efficient.

(v) However, Boolean model does not provide a ranking of the retrieved documents according to relevance. A document is either retrieved or not.

(vi) The similarity function is Boolean, it looks for exact matches, and there is no room for partial matches.

(vii) It is often difficult to translate queries into a Boolean expression.

(viii) In addition, the rigid difference between Boolean AND and OR operators does not exist between these words in natural language which makes it difficult for non-expert to use.

(ix) The query language is expressive but complicated, Boolean operators may retrieve too much or too little.

Despite these shortcomings, Boolean model is still being used by popular systems, it is the standard model for the current large-scale, operational retrieval systems and many of the major on-line information services use it, such as PubMed (Yepes, 2009).

2.10.4 Ranked Retrieval Models

Ranked retrieval models take care of the shortcomings of Booloean model, these models take into account the number of occurrences of terms in the documents to compute ranking. The ranked retrieval models return an ordering over the top documents in the collection for a query, and are suitable for non experts who need not use query language of operators and expression but their natural language. The size of the result set is not a problem, because the models just return the top *k* results. One

prominent ranked retrieval model is the vector space model; this is described briefly in the subsection that follows.

2.10.4.1 Vector Space Model

In this model, both the documents and the query are represented in multi-dimensional vector space where each distinct term corresponds to a dimension and the document's coordinate in that dimension is defined by the respective $W_{t,d}$ value (weight of *t* in *d*) (William, 2007). The vector space model procedure can be divided into stage:

- Document indexing which removes all the non-significant word in the document, these are called the stop words.
- Similarity coefficients find the similarities between document and query by using associative coefficients based on inner product of the document vector and query vector, the similarity is the overlap of the two.
- Length normalization transforms the documents' vectors into unit vectors, this compensates for the effect of document length on the vectors.
- Term weighting finds the weight associated with each term with respect to relevance to the user's needs. Term frequency, collection frequency and normalized length factor contribute to the term weighting calculation; product of these three is the resulting term weight.
    - Term frequency *tf* is the number of occurrences of term *t* in document *d.* Term frequency is content descriptive for the document.
    - Collection frequency is the total number of occurrences of term *t* in the collection.
    - Normalized length is the square root of the sum of squares of the document frequencies.

Document frequency *df* is commonly used in place of collection frequency, *df* is the number of documents in the collection that contain term *t.* This factor assumes that the importance of a term is proportional to the number of documents in which the term appears. The idea behind document frequency is that terms that are very frequent in the collection are less specific, this is expressed with the inverse document frequency (*idf*) obtainable by the formular

$$idf_t = \log(N / df_t) \quad \text{.....(2.1)}$$

each term in each document and query is expected to contribute a weight $W_{t,d}$ to the score of the document, and this is estiamted to be the combination of the term frequency and the inverse document frequency.

$$W_{t,d} = tf_{t,d} * idf_t \quad \text{.....2.2}$$

This implies that the term's contribution is positively associated with the term's frequency within the document (*tf*) and negatively correlated with the number of documents it occurs in (*idf*) (Salton and McGill, 1983).

The similarity score between a document and a query is estimated as the closeness of their respective vectors, the closer the two vectors, the more relevant the document is estimated to be to the query. Documents are ranked in this vector space using the cosine of angle between the vectors of the query and the documents in the collection.

$$Cos(q,d) = \frac{q.d}{|q|.|d|} \quad \text{.....2.3}$$

Where *q* is the query vector and *d* is the document vector

$|q|$ *and* $|d|$ are the Euclidean lenghts of the query and the documents respectively.

The combination of term frequency with inverse document frequency and length normalization (*tf-idf*) has proved to be the superior weighting scheme with respect to recall and precision.

2.11 Mobile Agent Architecture for Information Retrieval

Existing IR system based on client server paradigm consists of document servers set up by some organisation or individual wishing to publish a set of electronic documents. Documents in the document servers are viewed using the document client e.g. web browser. To view a document, the client sends a request containing a document identifier, e.g. Internet URL, to the server, which in turn searches its repository and returns the result of the search to the client. An information retrieval system available across the network is called the search server and needs a search client to access it. The existing system uses a search broker, which is a sophisticated search client that acts as an intermediary between a user searching for information and a set of search servers. Given a query and a set of search servers, the broker selects a set of servers at random that is likely to return relevant documents, queries them concurrently and produces a

single ranked results list. The search broker interfaces with various servers to retrieve their results and then apply a result merging method on the returned set.

{S, selection (S',q` Retrieval {(R_1,R_2,.....$R_{|S'|}$)q} Mergin ▸ R_M

Where q is the query S: the search server. S': selected search servers best for answering the query.

During server selection, the search broker selects a set of servers S', deemed best to answer the query. The servers' choice depends on both effectiveness and efficiency. It is usually assumed that all servers have equal search cost.

Using search broker however has certain flaws, these include the fact that at server selection not all servers with relevant documents are selected. There is also the possibility of selecting servers that have no relevant documents. A search server's index can become out of date as documents change.

The rapid growth of the networked environments especially the Internet, enomous available information widely dispersed, as well as quest for information across borders and platforms has increased the complexity of information sources. The multitude, diverse and the dynamic nature of on-line information sources however, make accessing any specific piece of information a difficult task (Brewington *et al.*, 1999). The use of agents for information retrieval provides a viable solution to these issues. Agents facilitates access to multiple information sources and the distributed nature of agents facilitate scalabity in the networked environment (Finin and Nicholas, 2000). Clark and Lazarou (1997); Htoon and Thwin (2008) identified certain functions distributed information retrieval agents are expected to perform, they are as follows:

- Accept requests from human user or other agent client
- Translate these requests to language of the information source or one understood by the information source
- Identify information source that contains information relevant to the request
- Pose the request to the source
- Collect the corresponding results from the sources
- Process the returned results
- Presents the result to the client.

2.11.1 Existing Agent Based Information Retrieval

Knowbot (Knowledge-Based Object Technology) collects information by automatically gathering specified information from web sites. Knowbot provides a single query language to access a variety of information sources and it serves as a representative for the user (Finin and Nicholas, 2000). Knowbot is a combination of data and a thread of control that can move among nodes in a distributed environment. The Knowbot Operating System provides a runtime execution environment which includes security mechanism, support for migration and facilities for communication between Knowbot and other programs. Knowbot is written in an interpreted object-oriented programming language called Python.

Metacrawler is a metasearch engine that queries a variety of search engines and provides a uniform user interface for these search engines. It combines the top web search results from different engines, downloads and scans pages if necessary (Finin and Nicholas, 2000).

Letizia: is a user interface agent that assists users browsing the World Wide Web (Lieberman, 2001). As the user browses, Letizia tracks user behaviour and attempt to anticipate items of interest by doing concurrent, autonomous exploration of links from the user's current position. People usually browse depth-first, Letizia browses breadth-first, and it uses a variety of heuristics to identify interesting pages (Finin and Nicholas, 2000). When an interesting page is identified, it displays it in a separate browser window. Letizia is implemented in Macintosh Common Lisp and it uses Netscape as a web browser and user interface. The agent runs as a separate process, and communication between Lisp and Netscape takes place using AppleEvents and AppleScript interprocess communication.

Retsina is a multi agent system (task, interface, information, negotiator agents) that cooperate with outlook based on Resource Description Framework (RDF) files to check appointments for changes autonomously, contact data accessible quickly and agrees with other Retsina users on appointments. It is a personal assistant agent.

However, the agent in this work is built to retrieve information from distributed databases using certain key to search for the information. The agent is written in Java, an object oriented programming language, it is a light-weight object embeded into the

operating system to run as part of the operating system and not on an existing platform. Our effort is directed at making agents run without passing through an agent platform.

2.12 Related Works

The mobile agent technology is rapidly gaining acceptance as a distributed computing paradigm, due to its ability to cope with the complexity in dynamic distributed systems. Several works have been done in this area of research, a lot of its many capabilities are yet to be explored. In retrospect, mobile agent has been applied in many areas of research such as, information retrieval and management (Clark and Lazarou, 1997; Brewington, *et al*., 1999; Teresa, 2006; Htoon and Thwin, 2008), electronic commerce (Busch *et al*, 2002; Rahul and Scrdhar, 2001; Hartmut *et al*, 1998), grid job scheduling (Iyilade, 2005; Enock and Munehiro, 2005), intrusion detection (Christopher and Thomas, 2001), expert finder (Iyilade *et al* 2005), network management (Aderounmu, 2001, Bohoris 2003), traffic detection and management (Chen *et al*, 2009), examination system (Gawali and Meshram, 2009), supply chain management (Wenjuan *et al*, 2009) and many more. A lot of issues arose with use of mobile agent; security, complexity and lack of standard. The complexity and sophistication naturally led to many attempts to simplify and extend agent functionality, thus attention later shifted to providing necessary security for mobile agents, agent platform and hosts on which they execute (Biermann, 2004; Narjes *et al*, 2009; Ibharalu *et al*, 2011). The versatility of mobile agent paradigm also increased research interest in enhancing mobile agents in the area of agent communication (Priya *et al*, (2009)) and agent structure (Stoian and Popirlan (2010)), in order to extend their functionalities. It is on this note that an attempt was made to enhance the structure of mobile agents in order to extend its functionalities in this research.

Clark and Lazarou (1997) developed a multi agent system to search for and retrieve technical papers stored in databases distributed over the network. The system managed sites as local deductive databases and WWW as distributed deductive databases. Existing systems focused mainly on system architecture and agent responsibility whereas this system combined the architectural, implementation and agent functionality aspects of distributed information retrieval system. In addition, the system also incorporated basic user interface agent class and adopted the browsing paradigm

for information retrieval. The system used prolog-like query formulation which provides accuracy and expressiveness of user query, and the predicate set of prolog query formulation corresponds to the attributes of the document these features combine to enable the system produce precise answers. The system is fully distributed, scalable and modular in nature. However, this system can be used only by professionals with the knowledge of the document type, authors and topics, the query is also entered in a Prolog-like form and these arguments must be supplied, which does not make it user friendly.

Seng (1999) presented mobile agent technology for enterprise distributed applications: an overview and an architectural perspective. The work shows the broad applicability of mobile agent technology in the enterprise, and also explored the implementations of this broad applicability from an architectural perspective.

Aderounmu (2001) developed intelligent mobile agent for computer network performance management. The system consists of a static agent resident on each host in the network and a mobile agent that migrates through the network to check the performance of network resources such as, hard disk, random access memory, printer availability. The mobile agents' codes were written in java and C++, and a mobility infrastructure was developed on top of TCP/ IP to facilitate socket-based connection between the source and the destination machines. The scheme was analysed against client server RPC, an existing paradigm for information retrieval and it proved to be superior with reduced bandwidth usage as the number of requests increases and reduced percentage denial of services in the face of network failure. However, the mobile agent can only execute on a host with its particular type of static agent installed and the agent also gets lost in transit if a node in its itinerary is not available, agent could not dynamically take alternative route. This work forms the background of this thesis.

An intelligent agent architecture for DNA- microarray data integration was proposed by Angeletti *et al* (2001). The agent architecture was proposed as communication and coordination tool among distributed sites. The agent architecture was applied to the case of expression of yeast genome, where several microarray hybridization experimental datasets are available. Mobile agent was deployed to search data and arrange genes according to similarities in their pattern of expression. The mobile agent uses its query language to search and select the available distributed experimental data by using Kohonen algorithm and related map.

Rafael *et al* (2003) presented a multiagent architecture for information retrieval in distributed and heterogeneous data sources. The work focused on the application of multiagent system, it retrieved and classified medical information from two hospitals as a decision support tool for doctors. The agent system was implemented with the master/slave design pattern and the agents were built on top of IBM's aglet framework named Tahiti Server. Aglet Tahiti Server placed a limitation on the system, in the sense that the functionality of the system was limited to those supported by Tahiti Server. The master/slave design pattern used generates so many slave agents that migrate through the network at the same time, causing a significant bandwidth overhead.

Tudor *et al* (2004) proposed a framework of reusable sructures for mobile agent development in an attempt to find ways of unifying mobile agent platforms. The authors defined reusable patterns in the context of location, agent, message, behaviour, agent idetifier e.t.c commomn to agent platforms. The implementation was based on JADE platform, and was applied to assessment service in virtual learning environment (VLE). The patterns separated the behavioural model from the actual skeleton of the JADE platform. The system was made up of a server agent, whose behaviour was distinctly defined, its observer behaviour checks periodically for the time of the assesment, when it's time, it performs its itinerary behaviour and migrate to the next node. On the destination node, it spawns a user agent, set it up with the task behaviour to deliver the test to the user. The server agent detaches and migrate to the next location perform the assignment until the final node before returning home. When all the tasks are accomplished, the user agent on each host sends the user's answer to the server agent that spawned it. The behaviour of the agents are separated from the agents structure, thus any agent visiting a node can use these behaviours. Only one agent migrates through the network on the departure trip, this saves network bandwidth. However, all the spawned user agents have to return to the origination agent at the end of the tasks, sending a lot of them almost at the same time, this puts a lot of load on the network. There is also a problem of unbalanced load on the originating agent and its node.

Mak and Fukuda (2005), developed a system "AgentTeamWork Grid" their system consists of 4 types of agents (commander, resource, sentinel and bookkeeping agents) using the branching pattern. The agents work together to download new resource XML files from FTP servers. This system spawns many agents into the network at the same time thereby putting additional load on the network bandwidth. At the same time,

because there are so many agents roaming the network concurrently, there is a high likelihood of agents collition. Therefore, a collition detection and prevention mechanism must be provided at an additional cost.

Rafal and Janusz (2006) proposed anonymity architecture for mobile agent systems. The architecture consists of infrastructures for concealing the identity of the user and the origin or base station of the mobile agents. The architecture consists of two modules, one is the untraceability protocol infrastructure that is responsible for obscuring agents' base addresses and the second is additional anti-traffic analysis support which aims at protecting agents from traffic analysis attack. The infrastructure was implemented for JADE using e-Health as their case study. The agent can be mistaken for a malicious program and denied access.

Htoon and Thwin (2008) in mobile agent for distributed information retrieval developed mobile agent on top of Aglet workbench to search for technical papers over the network. The system was a multi-agent system consisting of seven kinds of agents that work together to perfom the retrieval task across local and remote data repositories and databases. Results showed that the download speed can be significantly improved by mobile agents and that the download time does not depend on the size of the file or information being downloaded, the turnaround time was significantly reduced. With mobile agent approach, systems can be built in a truly distributed manner without a central data repository or a potential single point of failure. The work focused on controlling and managing distributed information retrieval processes. The system consists of two types of agents roaming the network which puts additional load on the network. Moreover, the agents are bound to Aglet workbench and are limited in functionalities to those functionalities provided only by Aglet, in other words, they are not interoperable with agent from other platforms and vendors.

Chen *et al* (2009) developed mobile agent system for distributed traffic detection and management. It designed agent-based real-time traffic detection and management system (ABRTTDMS). The authors integrated mobile agent technology with multi-agent system to enhance the ability of the traffic management system to deal with the uncertainty in a dynamic environment. The system was simulated through a laser-based vehicle detection system as against the existing magnetic loop detectors and video monitoring systems. The ABRTTDMS can integrate different detectors (such as laser detector, loop detector, video camera detectors and other detectors) into one system by wrapping them into agent-based sub-systems.

Wenjuan *et al* (2009) presented the development of mobile agent system for supply chain management and then presented a mobile agent system for supply chain management in the case of flower trading. The mobile agents make some intelligent activities in supply chain management automatically and intelligently, alleviating the bottlenecks involved in SCM negotiation, decision making and collaboration. The system is suitable for the rapid changing demand in global markets, dynamic and changing business environments, agile capability, and flexibility associated with current supply chain management. The work focused on application of agents to some intelligent activities, mobile agents are built to deal with negotiation, decision and collaboration intelligently and automatically. The agents written in java programming language are built to run and execute on JADE platforms, security and improvement of agent capabilities are intended for future research. The branching design pattern used clones an agent up to the number of agencies required to visit and sends them all on the network putting additional load on the network bandwidth.

Gawali and Meshram (2009) designed an Agent-Based Autonomous Examination System a multi-agent system containing main agent, mobile agent and stationary agent. The system was developed to support the functionality of examination systems in Aglet environment. The stationary agent authenticates candidates for valid user name and password at user authentication and selection of subject, and database with the help of mobile agent. After successful authentication and selection of subjects, mobile agent collects questions, their alternatives and correct answers retrieved by stationary agent from the database. As candidates answer the questions, main agent stores the answers given by candidate in the database and updates the score. When the examination gets over, the main agent processes the results and displays result.

Priya *et al*. (2009) proposed an enhanced communication scheme for mobile agent. The frequent movement of agents poses challenges for the design of an efficient communication protocol for mobile agents. The proposed scheme overcomes the problem of overloading of the agent and provides a mechanism for mobile agent to automatically update itself according to environmental changes in a predictable and visible manner. It also reduce the resources overhead over the network dynamically by temporarily destroying the idle agents and reconstructing them when needed.

Stoian and Popirlan (2010) proposed an enhanced mobile agent architecture which is based on the existing agents but with some additional components. The system focuses on agent architecture; it improves the structure of existing mobile agent adding

Recorded Agent Information which extends its functionality. The architecture proposed is suitable only for hierarchical design patterns of mobile agent, it uses a master/slave approach in which a master agent spawns and dispatches slave agents one each to all other agencies. Results obtained showed that the efficiency of mobile agent can be improved by enhancing its structure. Processing took a shorter period with the use of the Recorded Agent Information provided to enhance the mobile agent. The Recorded Agent Information includes learning and optimizing algorithm that learns from the previous runs and makes use of optimal results for better performance. The work involves sending so many slave agents on the network which consumes a lot of the network bandwith. In addition to this, the agent pattern is not dynamic, each is defined at design time and cannot change, slave agent cannot be made to act as the master neither can master agent acts as slave agent. On the other hand, the work forms one of the motivating factors for this research work, the fact that the functionalities of mobile agents can be improved for greater efficiency by enhancing its architecture is a motivating factor for this research.

2.12 Overview of Proposed Approach

The focus in this work however, is to enhance mobile agent to perform its assigned task without having to go through the agent platform, emphasis is on mode of deployment of mobile agents. The work is inspired by the need to improve the capability of mobile agents, so that more of its numerous potentials can be exploited. This work is directed at making the agent an operating system service. The target operating system is Windows Operating System and Windows XP specifically. Operating Systems' services are long-running executable programs that run in its own Windows session, without users' intervention. The motivations are to save time, reduce memory requirements, enhance the heterogeneous property of the mobile agent since it is platform (MAS) independent, as well as its portability across different organization while conforming to the FIPA standards. The proposed embedded agent is provided with a dynamic migration algorithm that enables it to determine alternative route to its destination in case of failure of a node and report same on its return to the origin. Further more, the system employed the itinerary design pattern, which limits the

number of agents roaming the network at every point in time to one type to conserve the network bandwidth.

CHAPTER THREE

SYSTEM DESIGN AND PERFORMANCE ANALYSIS

3.1 Introduction

This chapter discusses the design strategies for the proposed embedded mobile agent in distributed information retrieval. It contains various stages of design and the associations between different components of the proposed system. The chapter also presents the conceptual design of the performance metrics that are used to evaluate the proposed framework.

3.2 The Agent System Model

The proposed system architecture comprises of a light-weight static agent embedded in the kernel mode of Windows operating system and mobile agent that moves between network environments taking advantage of network resources to fulfil their goals. Mobile agents are expected to have the ability to migrate to the appropriate location when given the constraints of security, cost, distance and other factors and may contain behaviour and some form of knowledge base. The behaviour specifies the agent's

reasoning ability and functionality and may also include its norms and belief. A typical mobile agent contains an agent model, a life cycle model, a computational model, a security model, a configuration model and a navigation model. Software agents for information retrieval use human searching techniques and are able to learn from the environment. They are capable of autonomous organisation of tasks, they can divide a huge task into sub-task and distribute the problem for ease of computation.

The existing mobile agent for information retrieval framework consists of mobile agents that execute on agent platforms previously installed on the computer machine. The agent platform is installed in the memory on top of the operating system running on the host and runs in the user mode; this obviously consumes memory, and increases access time. An attempt is made to provide solution to these problems by designing a light-weight agent as part of the operating systems to run in the kernel of the operating system, to free memory and reduce access time. Figure 3.1(a) shows the block diagram of existing system that uses agent platform while Figure 3.1(b) shows the block diagram of the proposed architecture without the agent platform.

3.3 Proposed Embedded Mobile Agent (EMA)

The embedded mobile agent is proposed for information retrieval in distributed environments. The embedded agent defined as mobile agent class with certain attributes such as execution state, data, and the added enhancement. The added enhancement is in the form of Terminate and Stay Resident (TSR) Program that enables the mobile agent to directly interact with the operating system on any host in the form of Windows Operating System Service. A lightweight static agent was embedded into the kernel mode of the Windows Operating System. The agent is automatically activated at the boot of the machine on which it resides; it runs in the background continuously without interfering with the other user's operations until it is restored. The mobile agent launched from a host moves through the network to the host topmost on the list of its itinerary to retrieve information from distributed databases.

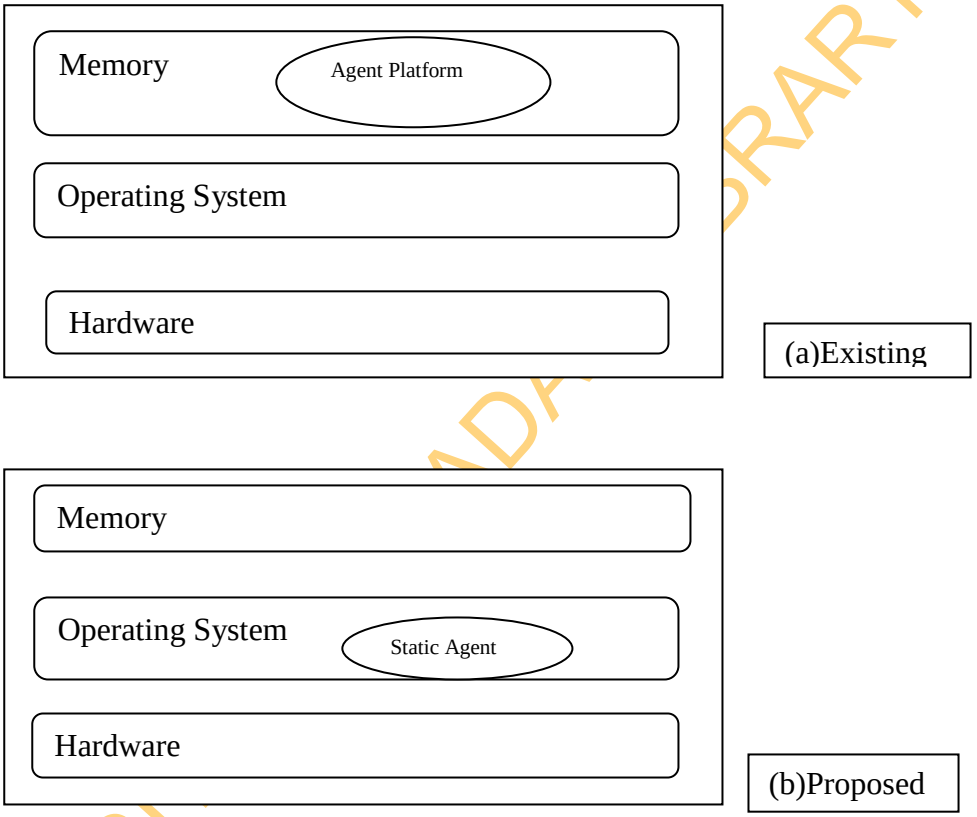


Figure 3.1: Block diagram of the existing and proposed frameworks

3.3.1 The Internal Structure of Agent

The embedded static agent is designed using the layered architecture. Each layer represents a particular function, following the principles of layered architectures such as the three layer architecture employed by (Ahmed, 2007 and Wenjuan *et al.*, 2009). The layered architecture is a programming is programming paradigm that uses different layers to allocate responsibilities of an application. According to Spicer (2000), layered architecture defines hierarchy among components and there are no direct software dependencies up the architecture, that is, lower layers have no dependencies on higher layers, which is important to the reuse of the architecture. This implies that one layer can be modified without affecting other layers. In addition, layered architecture is easy to design and implement once the layers and their interaction are clearly defined. Based on this architecture the agent in this work is built to consist of three m ain subsystems as depicted by Figure 3.2. The layers are knowledge formation process layer, knowledge base layer and inference engine or decision making layer.

1. Knowledge formation process model: is a learning algorithm by which the intelligence of the agent is learnt, it consist of the description of contents of the information sources which includes description of the classes contained in the information source and the relationship between these classes. It is used to determine how to process an information request. Knowledge formation is based on information transmission carried out through signals, in which the information is stored with the help of a code. Knowledge involves three processes, these are
   (i) Encoding: generates knowledge encoded in forms that facilitate its transmission to others.
   (ii) Exploration: captures "search, variation, flexibility, discovery, experimentation, innovation". Exploration generates new, unsettled knowledge with potentially high but certain returns.
   (iii) Exploitation captures "refinement, choice, selection, efficiency, implementation, execution". Exploitation generates incremental knowledge with moderate but certain and immediate returns.

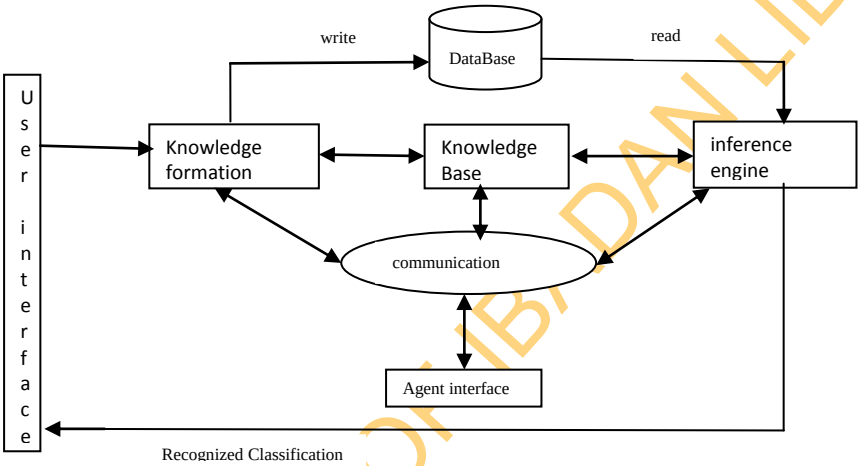


Figure: 3.2: Internal Structure of static Agent

2. Knowledge base: consist of the intelligence of the agent, defines the agent's area of expertise and the terminologies for interacting with that agent. In this research work, the knowledge base consists of terminologies in the information retrieval domain.
3. Inference engine: maps the query with the knowledge in the knowledge base to make decisions; it describes the resources that are available to an agent to answer information requests.
4. Communication: queries to an agent are expressed in the structured query language (SQL). These queries are composed of terms in the general model, therefore, there is no need for users or other agents to know or be aware of the terms used in the underlying information sources. To make the database available to the network of agents, a wrapper is built around the existing system to turn it to an agent with access to that information. The wrapper built is a relational database management system which is created and managed by MySQL supported by XAMPP and communication between agents is via message passing.

The knowledge formation process learns from the knowledge stored in the knowledge base and the inference engine combines the learnt knowledge with the existing knowledge to make decisions. The three layers interact with the agent interface through communication. The database is outside the agent, but the agent can read from the database and write to the database depending on the request. The user interface directly interacts with the agent through the knowledge formation process and the recognised class of relevant information, which is the output from the inference engine is related to the user interface.

Database: is a large repostiory of data, all the data to be used by the system are stored in the database, which maintains the integrity of data stored in it. The database consists of weather data from different locations in Nigeria.

3.3.2 Mobile Agent Components

A mobile agent contains code, state information and attributes. The attributes of mobile agent include its name which is unique for identification, the authority of the owner of the agent and the agent system type. Code contains the logic of the agent, that is code defines the behaviour or the required tasks of the agent, same type of agents use same

code. Code in an object oriented context means the class code necessary for an agent execution (Lange and Oshima, 1999).

Data corresponds to the value of the agent's instance variables and include information about the mobile agent such as its launcher, movement history, resource requirements and authentication keys for use by the infrastructure, these are refered to as the initial data. The data could also include results of the mobile agent's tasks on different nodes visited, refered to as the generated or received data.

The state information defines its changing variables, which enables mobile agent to resume after moving to a new agent host, such as the stack pointer and program pointer. The state means the attribute value of agent that help it determine what to do when it resumes execution at its destination (Lange and Oshima, 1999). According to Aderounmu (2001), state is needed for the agent to resume computation after travelling.

Itinerary corresponds to the path that defines the agent's journey between different hosts. The itinerary can be determined during creation by the agent's creator or at run-time according to specific input variables as received during computation (Biermann, 2004). The agent itinerary in this work is defined at run time, by selecting the list of agent_environments the agent should visit.

Name is a unique identity which depends on an algorithm that will, during the creation of mobile agent, give it a unique identification for address and navigation. Agent needs to be uniquely identified so that its owner can communicate with and control it while in transit. The agent environment also needs to be uniquely named so that an agent can specify its desired destination and its current location. The agent environments in this study are given unique names, which corresponds to the name of the computer on which the agent environment resides.

The agent class has attributes execute state, communicate, record and the list of tasks to perform. The UML diagram (class diagram) of the embedded mobile agent is as shown in Figure 3. 3. The agent map corresponds to the agent itinerary. Audit trail tracks the operation of the agents such as acceptance or rejection on any host, delay and time it starts executing and unavailable host.

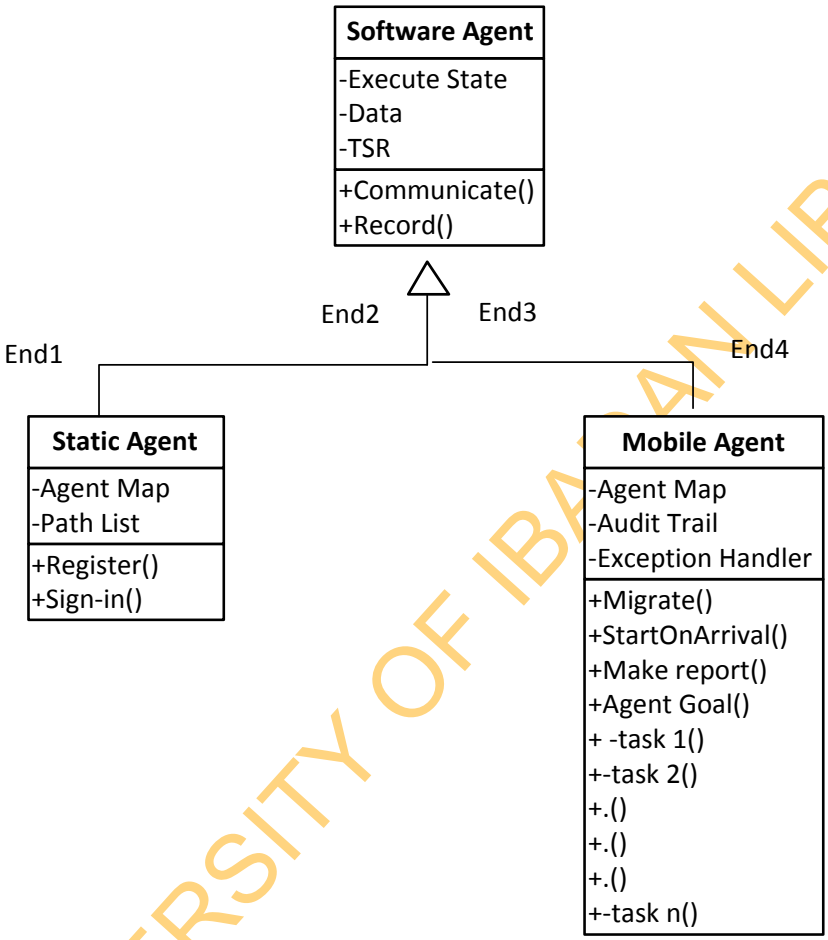


Figure 3.3: agent UML Class diagram

3.5 The architcture of the proposed system

The proposed system consists of a static agent embedded into the kernel of the operating system on each host and the mobile agent that migrate through the network to perform the assigned tasks. Figure 3.4 illustrates the conceptual model of the proposed system; it consists of a lightweight (small size, precisely 1 KB) static agent embedded in the kernel mode of the operating system on each site. The operating system logically sits on top of the hardware that is connected to the network. Each visiting mobile agent migrates through the network to a remote host and interacts with the static agent seeking access and eventually, performing its assigned tasks on the host.

The structure of the proposed enhancement is depicted by Figure 3.5 which presents the architecture of Windows XP operating system with the static agent embedded in the kernel mode as an extension of its executive services.

The overall architecture of the system is illustrated in Figure 3.6; it consists of four hosts connected to an existing local area network. The mobile agent from remote host interacts with the static agent in the kernel mode of the visited host operating system, giving an impression of the mobile agent directly interacting with the operating system. The static agent is a Terminate and Stay Resident (TSR) program embedded in the kernel of the Windows operating system; it is automatically started as soon as the operating system boots. The static agent automatically closes when the operating system shuts down and it can be explicitly shut down. The idea is to eliminate the mobile agent platforms that are installed in the computer memory which consumes memory and increases access time.

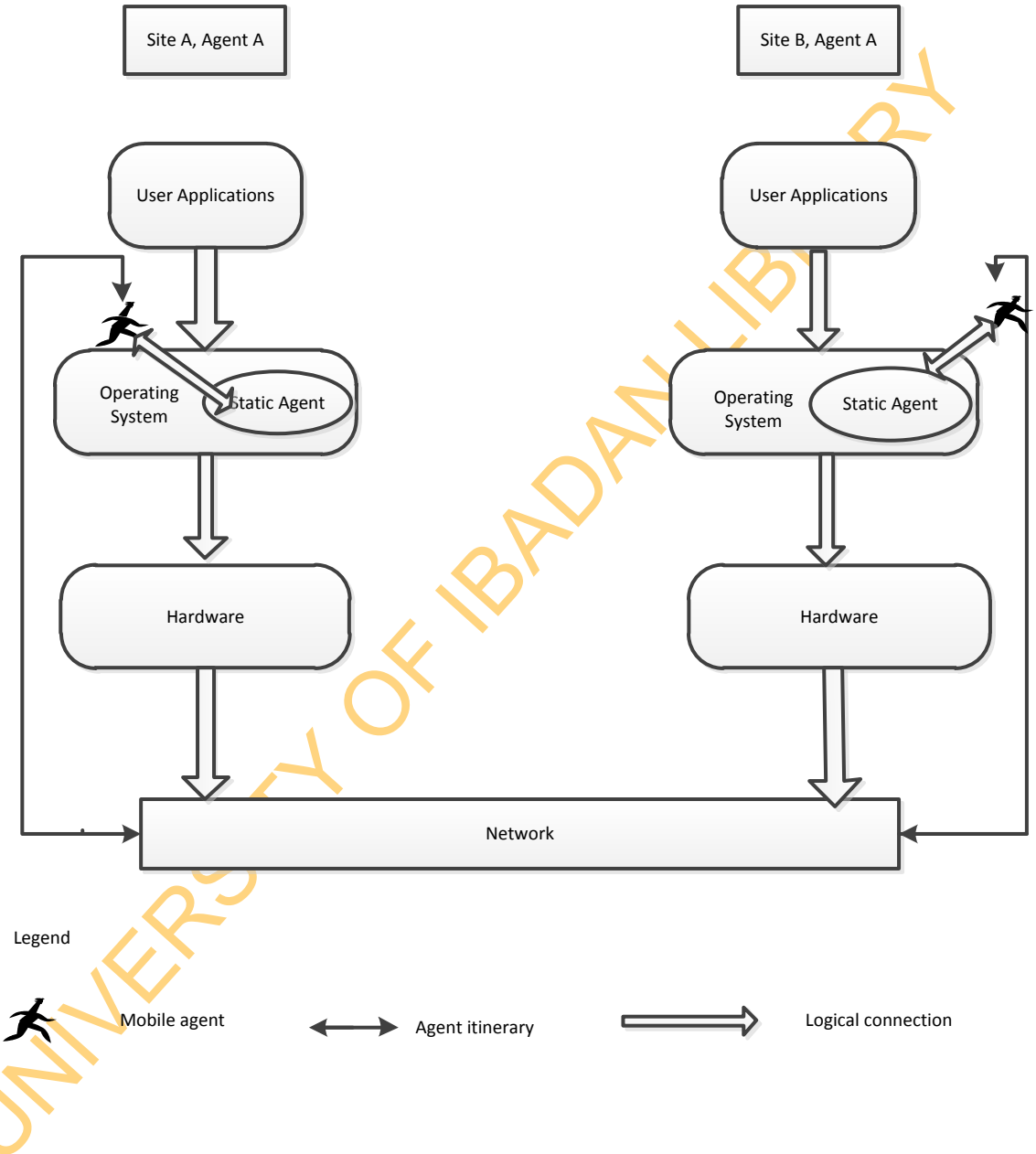


Figure 3.4: The Conceptual model of the proposed system

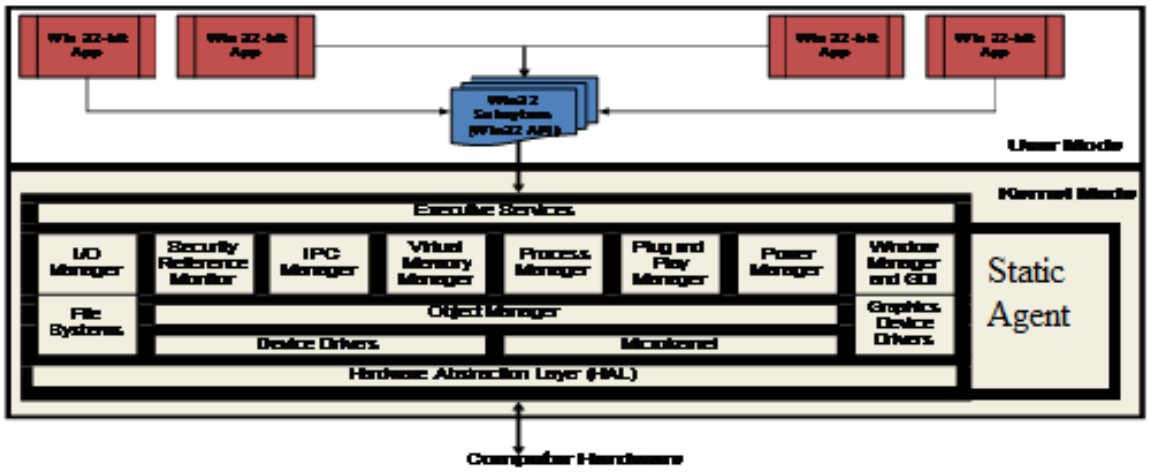


Figure 3.5: Proposed embedded agent as Windows XP operating system service (adapted from WIN133, 2009)

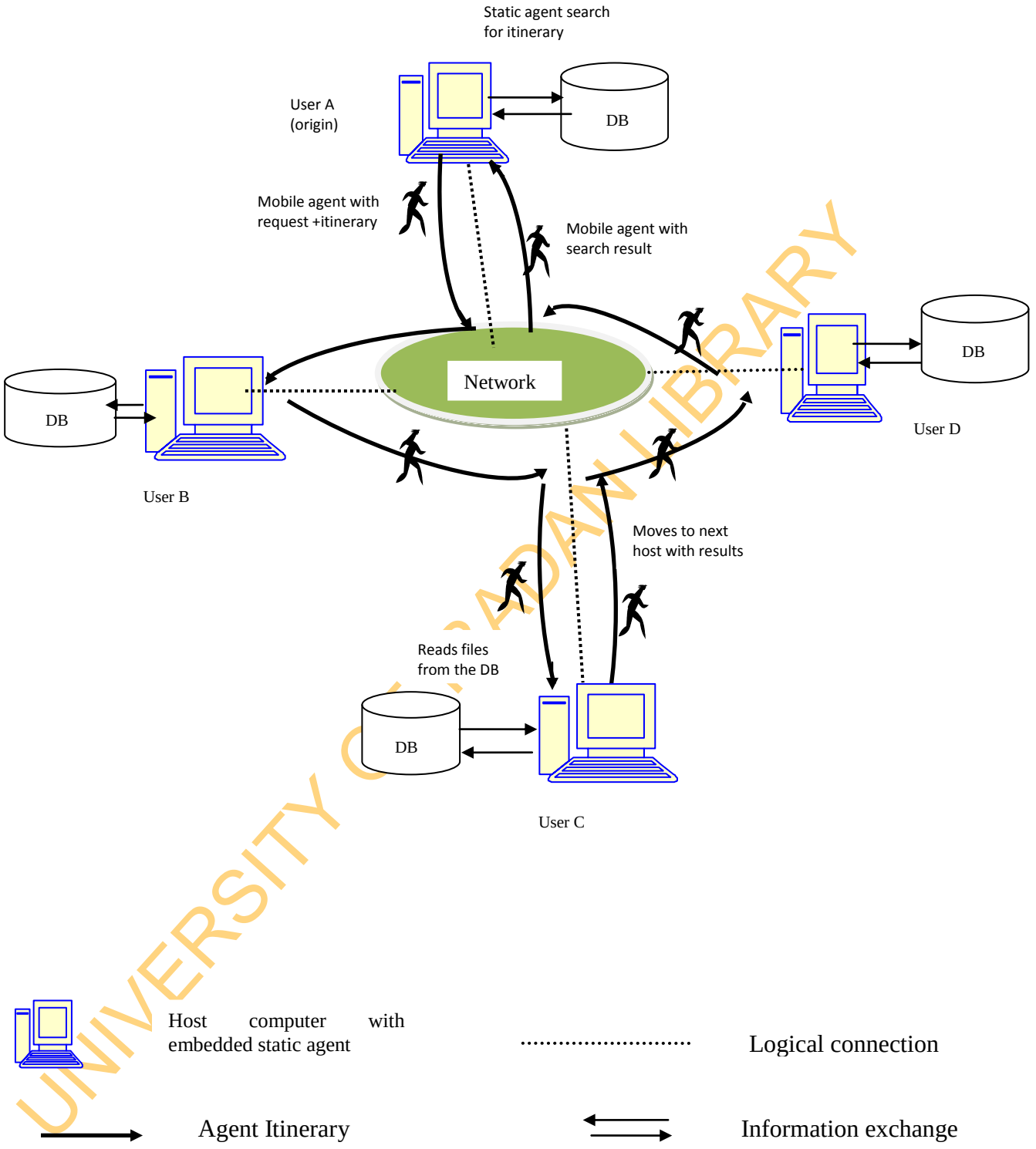


Figure 3.6: Overall System Architecture

In this work, the static agent performs a number of functions related to information retrieval in distributed environment.

(i) It is responsible for listening to the port for incoming agent.

(ii) It negotiates passage to the destination host and ensures that the mobile agent is successfully transferred. If the mobile agent is rejected, the static agent restarts the mobile agent to allow it to choose another destination.

(iii) It validates and authenticates the incoming agent

(iv) It launches received mobile agents and provides runtime execution for the mobile agent according to the level of trust given to the agent. The runtime execution environment will depend on the access level granted to the mobile agent and the functions it wishes to perform.

(v) It provides a registration to register the static and mobile agents and the available resources.

Mobile Agent as the name implies is the computational element of the system that can migrate between nodes on the network that is, it can move from node to node and execute there as required to achieve its goals. The mobile agent is supplied with the list of nodes to visit and the user query. When it arrives at the destination, it is recieved by the static agent who authenticates and validates it. The static agent provides a runtime execution for the mobile agent depending on the level of access granted it. The mobile agent negotiates with the static agent giving the data for the search process, when it gets the result of the process, it adds the result of the process to its bag. The mobile agent initiates a **move()** to the next node in its itinerary. The mobile agent is saddled with the following responsibility:

(i) Migrates from node to node in the network

(ii) Negotiates access with the static agent on the remote host

(iii) Negotiates with the database for the search process

(iv) Receives the required information and adds it to its bag.

(v) It autonomously determines the next node to visit and initiates a move

(vi) It returns to the originating host with the results of the search and disposes itself.

Every remote node has a database, which is a repository of information about weather. The database in each node is constructed with PhpMyAdmin in XAMPP for windows. When the mobile agent states it requirements, the database is accessed and matches for

query terms are searched. Structured Query Languge (SQL) was used to create, examine, manage and manipulate relational databases. SQL is a standardized query language across different database vendors so that a program could communicate with most database systems without having the needs to change commands.

Each node has the capability to accept and dispatch users' jobs or information needs. Users send their requests to the static agent who searches its local files for remote sites that are likely to have information relevant to the user's request. The static agent then initiates a mobile agent, supply it with the user query and its itinerary. The mobile agent initiates a ***move()*** and moves to the first host in its iteneray, performs its task, adds result to its bag and moves to the next node in its itinerary.

3.6 Communication Pattern

The agents in the system communicate with each other using message passing, this is necessary to accomplish their given tasks. Messages are used to transfer data between the agents; messages have sender and receiver's addresses and the message contents as parameter and are sent to a target destination. Message passing is a point-to-point communication model which represents a simple connected undirected graph (Chalopin *et al*., 2006). The vertices represent the processes at the nodes and edges represent the communication link. Two vertices are linked by an edge if corresponding processes have a direct communication link. This is suitable for the work in which mobile agent from the origination node could visit remote nodes and execute there. The following section describe the communication procedure.

3.6.1 Communication at the Initializing Node (Origin)

The interaction between the constituent components at the originating host is discussed in this section and depicted in Figure 3.7. Conceptually, the user enters a query via the user interface, the static agent interpretes and represents the query in a format understandable by the agents and initiates a search of the local database for list of resource servers. The static agent activates the mobile agent, supplies it with the itinerary and request data. The Mobile Agent saves its current state, signs off the host and takes its journey to the first location in its itinerary.

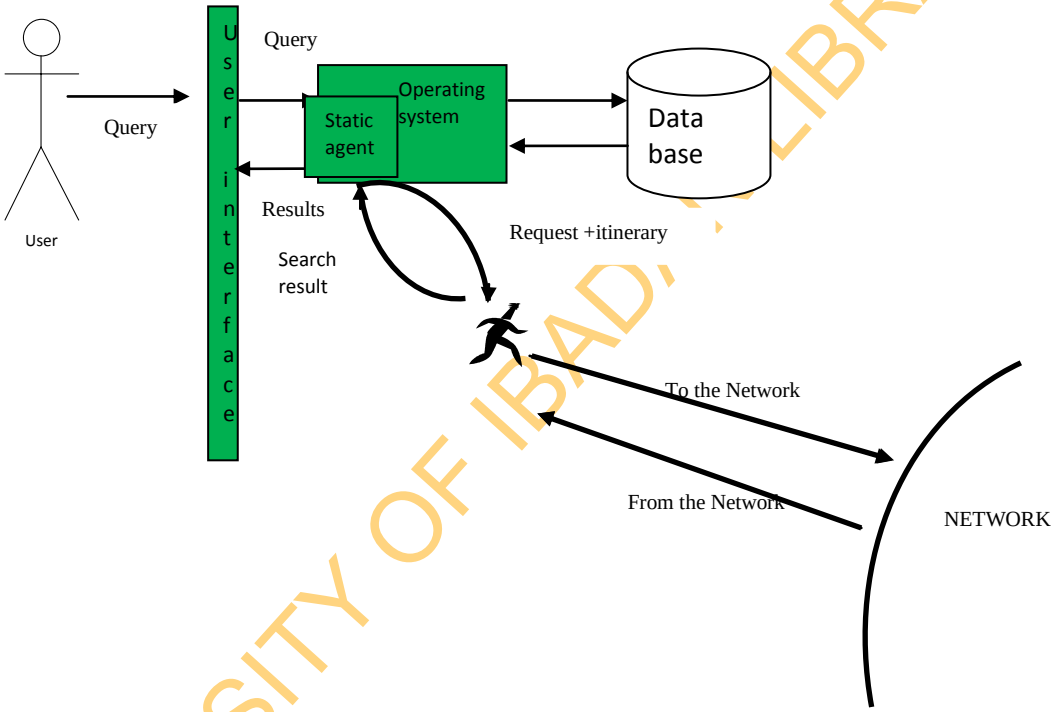


Figure 3.7: Interaction at the initiating Host / origin

The developed system uses a parametric search technique; the user enters the query by selecting the search criteria, the weather condition or the temperature range, and supplies the list of agent environments (nodes to visit i.e. agent itinerary). The static agent interpretes and represents the query in a format understandable by the agents and initiates the migration of the mobile agent. The mobile agent takes its itinerary and negotiates passage into the network to accomplish its tasks.

3.6.2 Communication at Remote Host

On the remote host, the following interactions take place in concept and principle as illustrated in Figure 3.8. The static agent on the remote host authenticates and receives in coming mobile agent, initiates a search of its local files for the relevant information. The mobile agent downloads the information and adds it as part of its bag. The mobile agent moves to the next host in its itinerary. The mobile agent on reaching a new host in its itineray, repeats the same process and moves to the next host until the last node in its itinerary. It then returns home with the results in its bag and forward the result to the static agent at the origin who displays the result to the user. The interaction takes place by executing the following algorithm

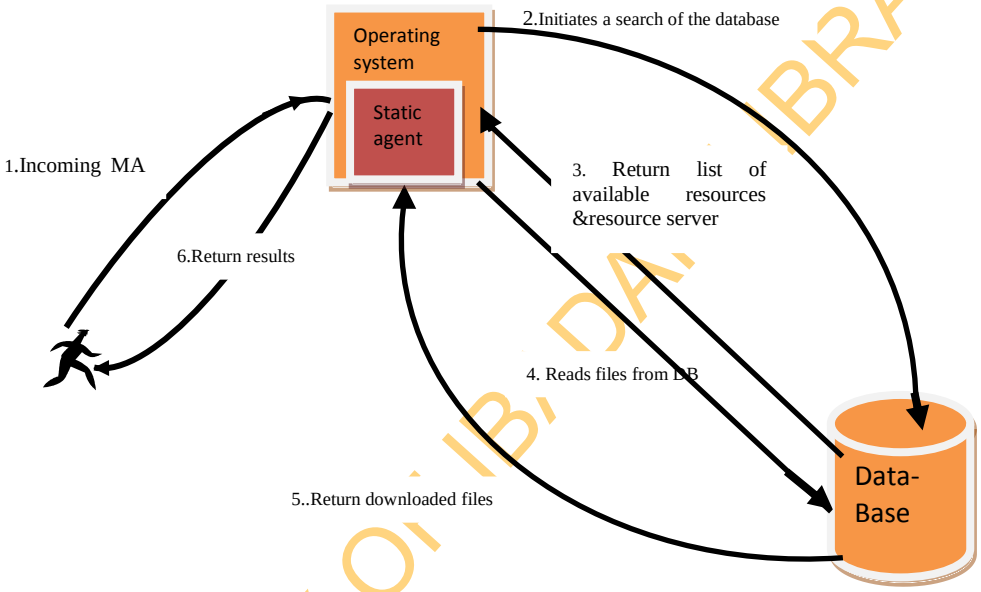


Figure 3.8: Interactions at the receiving Host

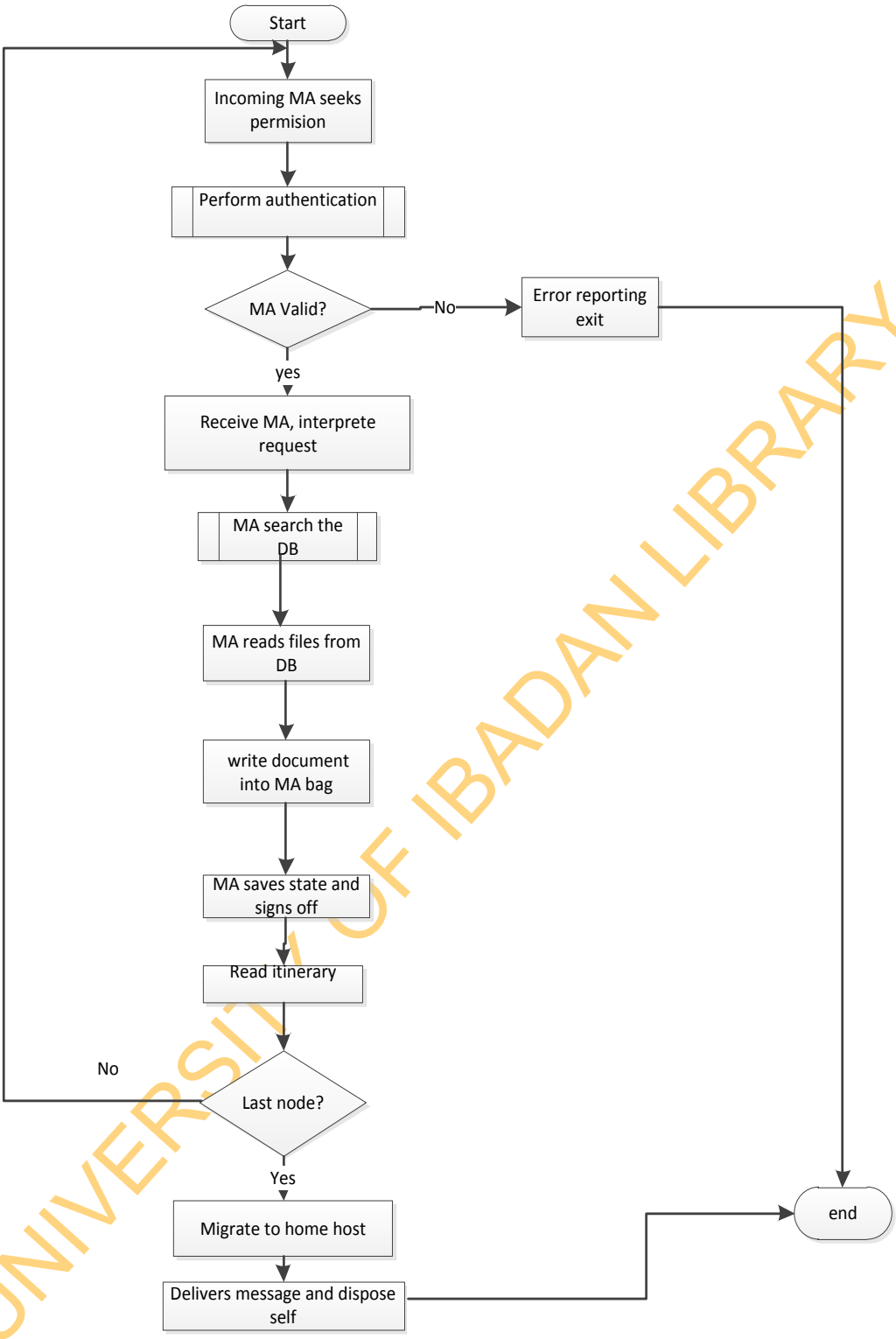


Figure 3.9: Interactions at the receiving host flowchart .

3.7 Agent information

The mobile agent has as part of its bag the address of the host where it is currently located, the address of the host visited before moving to the current location, the host it was able and unable to locate. The agent also has information about its current status (either active or suspended). The algorithm for the agent's operation is as show in Figure 3.10.

The user initiates the creation of a mobile agent at the home host. The mobile agent loads its state and itinerary data. The agent saves its present state and initiate a move to the first destination in its itinerary. On the new host, the mobile agent is authenticated and given access, performs its assigned function and adds to its bag the obtained results. It then checks its itinerary and proceeds to the next destination to perform similar tasks and if all nodes have been visited, the mobile agent returns to source host, and presents the results. The result is then displayed to the user.

Step 1: Incoming Mobile Agent seeks permission to perform its tasks. The static agent receives and authenticate the incoming agent.

Step2: The static agent after recieving the requests interprets the requests and initiates a search of the local database for available relevant documents.

Step3: The mobile agent queries the database using keywords

Step4: The search results are added to the mobile agent as part of its bag. The mobile agent saves its current state, signs off the visited node, exit and continue in its itinerary, and if it's the last node in its list, returns to the origin, delivers the result and disposes itself. The interaction at the receiving node is summarized by the flowchart in Figure 3.9.

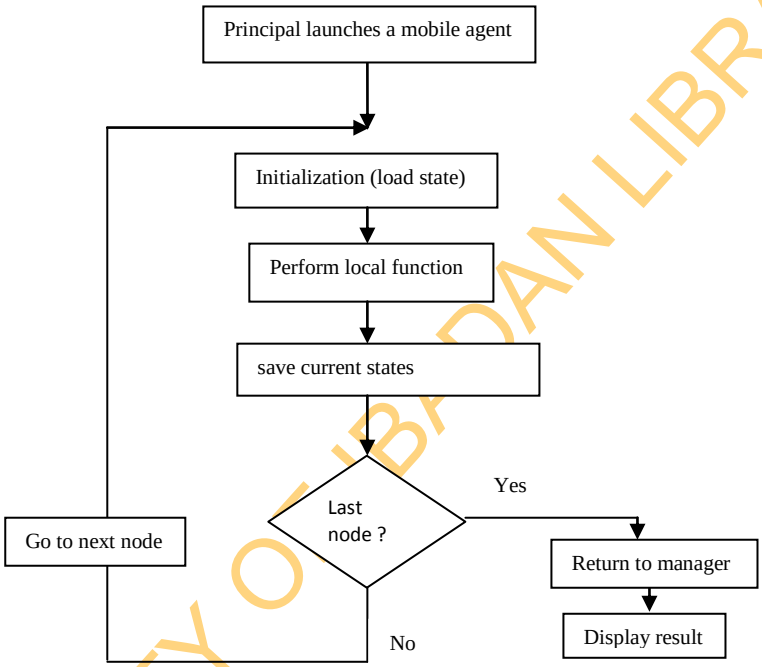


Fig. 3.10 Main loop of mobile

3.9 Mobile Agent Migration

Mobile agent migration desribes the process of packaging the code, data and execution state of a mobile agent into a form that could be transported through the network. The agent transfer process can be initiated by the agent itself, by another agent residing in the same environment or by another agent or management outside the place. The agents in this work were written in Java, which provides a standard feature for object serialization. Java only requires the object to be implemented using the java.i.o.serializable interface, and it automatically handles the serialization internally. Serialization is the process of converting an object state into a form that can be transmitted over a network connection and later restored at the other end by a process called deserialization. An object cannot be instantiated without an associated class file which represents the behaviour i.e methods. Java allows progarmmer to customize class loading. Custom class loaders are used when the default java class loader cannot locate a class in either the local cache or directories specified in the CLASSPATH system variable. Therefore, the programmer can dynamically load the class from a remote location over the network.

When an agent migrate, it takes with it, its code, its state of execution and transferable resources as supported by Gherbi *et al*., (2009) and Mitrovic *et al*., (2011). Agent migration operation is defined in the steps below and depicted in Figure 3.11.

(i) Naming: When the mobile agent on a home host is started, in this work it is started by the static agent on the same host, who defines the agent name and its own identity. This naming is important for identification and authentication of the agent on other hosts to be visited. The agent identity is verified against the registered agents in the agent base, if it is authenticated, its execution can then be suspended.

(ii) Suspend execution: the current execution thread of the agent is suspended and every thread is also suspended. This is necessary, so that the data and state are frozen and cannot be modified.

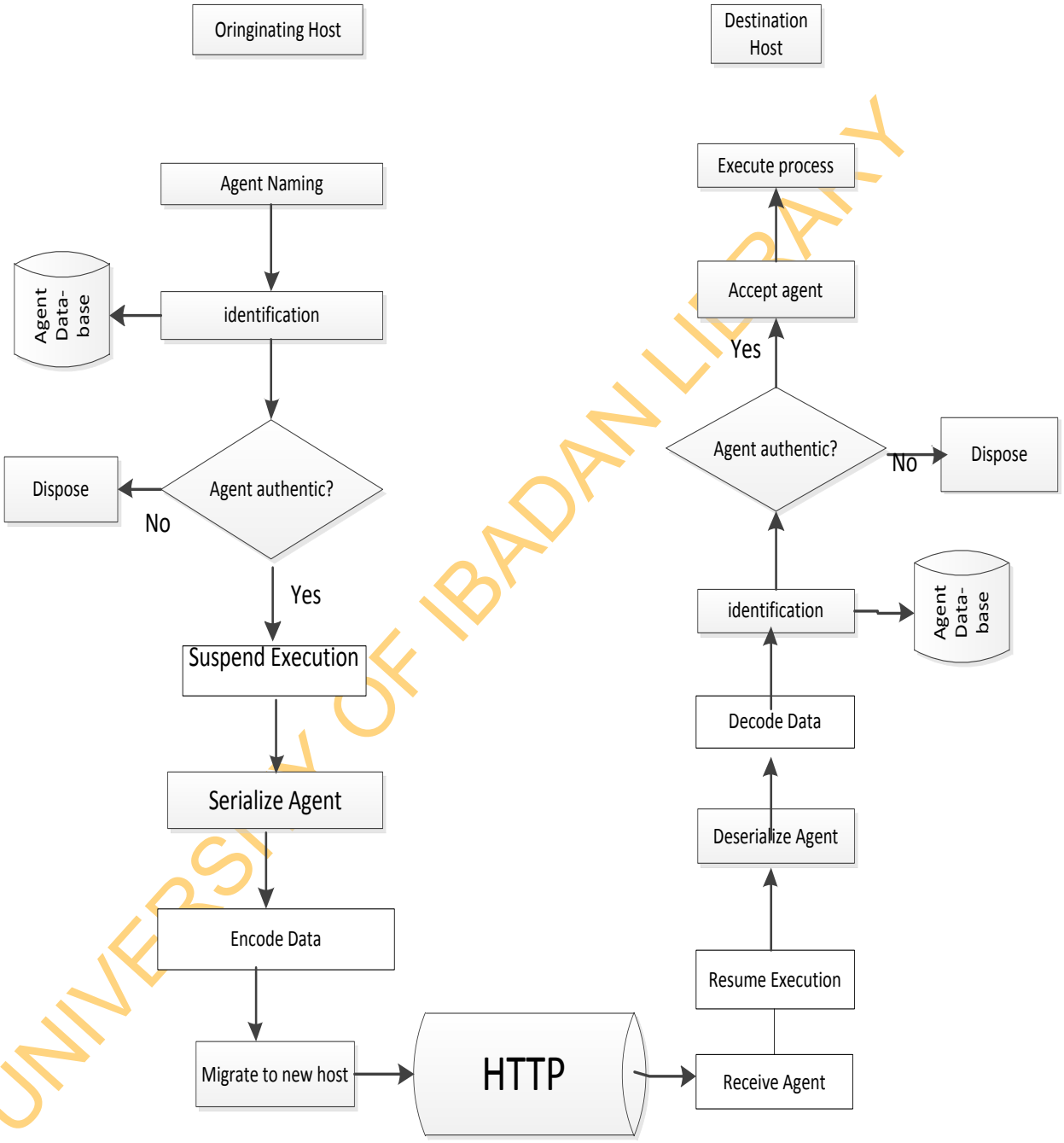


Figure 3.11: Mobile Agent Migration.

(iii) Serialization: the agent is transformed into a stream of flat bytes consisting of the agent's data and state information. It also captures the agent's state so as to know the point of suspension.

(iv) Encode data: the current state of all variables (data) of the agent is serialized, i.e. their current values are written to an external persistent representation. The value of the agent's state is also stored.

(v) Migrate the agent: the serialized agent is transferred to the receiver using a migration protocol, HTTP over TCP/IP in this case.

On the receiving host, the static agent listening to the port for incoming agent is trigered and the following processes take place.

(i) Receive the agent: the serialized agent is received using the migration protocol.

(ii) Resume execution: the receiving host resumes execution on receiving the agent

(iii) Deserialize agent: the serialized agent is deserialized on the destination host, i.e. the variables and execution state are restored from the serialized agent. The result of this should be an exact copy of the agent that existed on the originating host just before migration is initiated.

(iv) Decode data: associated data are decoded, i.e. the values of data written to an external persistent representation are read at the destination host.

(v) Identify the agent: the agent is compared with the list of acceptable agent to the system stored in the agent database of the destination host.

(vi) Authenticate agent: the received agent is authenticated, in case of modification to prevent malicious agents from gaining access.

(vii) Accept agent: the agent is given access to the destination host's resources and

(viii) Execute process: the destination host resumes agent execution by starting a new thread of control, the agent make use of the available resources to accomplish it tasks.

3.11 Database Design

Database is a large repository of data and information and it is managed by the distributed database management system (DDBMS). Distributed database is a database that is spread across a network of computers that are geographically dispersed and connected via a communication lines. The database must have a single logical data

model. A distributed database management system (DDBMS) governs the storage and processing of logically related data over interconnected computer systems in which both data and processing functions are distributed among several sites. The database in this research work is a relational database in which the entities involved are represented in tables with their attributes. This work uses the XAMPP for Windows 1.7.1 for its database application. The XAMPP for Windows is developed basically for software developers to run their Internet–based software before going to the Internet, in order to debug and correct errors. XAMPP is a combination of different software packages prominent amongst whom form the XAMPP acronym:

- X cross platform
- A: Apache HTTP server is a web server for WWW. Designed by Rober McCool in 1995 written in XML and C, it is cross platform.
- M: MySQL database is a RDBMS that runs as a server providing multi-user
- access to a number of databases. It is named after developer Michael Widenius' daughter, My, SQL is Structured Query language.
- P: PHP: hypertext Preprocessor is a general purpose server-side scripting language originally designed for Web development to produce dynamic Web pages. PHP is imperative, object oriented, procedural and reflective designed by Rasmus Lerdorf in 1995
- P: Perl: practical extraction and reporting language: is a high-level, general-purpose, interpreted, dynamic programming language developed by Larry Wall in 1987 as a general purpose UNIX scripting language to make report processing easier.

The database is launched from the phpMyAdmin as tool and uses the (SQL) Structured Query Language to query the database.

The weather table

Table 3.1 Weather relation

Weather [id, weather_temperature, weather_condition, weather_location]

Id	Weather_temperature	Weather_condition	Weather_location

Table 3.1 stores the details of the weather, the location with their temperature and the athmospheric conditions. The id is the identity of each location which is the serial number, it is of integer type, can take as much as eleven digits and it automatically increases. The temperature is the athmospheric temperature and can take up to eleven digits while the condition states the athmospheric condition of the particular location, which can be a city, town or village.

Agent Environment table

Table 3.2 Agent environment relation

Agent-environment [id, agent_environment]

Id	Agent_environment

Table 3.2 is the agent environment relation consisting of agent environment which is the name or the IP address of the computers on the network, their id is their identity which is the number given to each computer for ease of referencing.

3.12 Performance Evaluation

In this section, the performance of the proposed mobile agent against JADE an existing mobile agent, running on an agent platform was evaluated.

The performance parameters used are service delay, memory management, fault tolerance, denial of service and turnaround time. Mathematical models were developed for:

(i) Service delay against the number of nodes in the network for JADE and embedded mobile agent
(ii) Memory utilization versus the number of hosts in the network
(iii) Percentage denial of service against number of requests
(iv) Fault tolerance in the face of power failure
(v) Turn around times for varying number of hosts visited.

3.12.1 Service Delay

This is referred to as the overall time required to execute a service. For the purpose of this study, the following components of the delay are defined.

Waiting time: is the time interval between the arrival of a request at the destination and the beginning of execution.

Activation time: is defined as the time taken to activate the mobile agent platform. Activation of mobile agent platform is defined as the process by which the mobile agent platform on the destination hosts senses and is triggered to receive agents for execution.

Transfer delay: is the time interval between the generation of the last bit of packet at the information source and the transfer of agent.

Service time: is the time taken to complete agent's requests i.e. time interval between the beginning and end of execution of a particular service (st).

Transfer delay is suffered twice, when the agent leaves its source on request operation and when it leaves the destination on response operation. In this simulation, the transfer delay consists of two components

(i) Time to save agents internal state (ts)
(ii) Time to sign off from agent platform (ti)

Activation time is the total time it takes the agent to be triggered i.e. the time interval between when the agent arrives on a host and the beginning of its operation. The activation time is broken down into the following components

(i) Time to accept and authenticate incoming agent (ta)
(ii) Time to provide a hierarchical name space for agent (tp)

(iii) Time to allow agent migration and communication (tc)

(iv) Time to restore agent's internal state (tr)

Waiting time: the time interval between the arrival of a request at its destination and the beginning of execution (tw).

Assumptions

Both systems operate on the same principle for transmission, both are java based agents, therefore, the processing delays due to packet transmission through the network is the same, and thus not considered.

For the first request **tw** = 0 since there are no previous requests

For the second request **tw** is the service time of the first request i.e.

$$tw(2) = st(1) \tag{3.1}$$

and for the third request **tw** is the sum of the service times for the first and second requests

$$tw(3) = st(1) + st(2) \tag{3.2}$$

Assume the service times for all requests are equal and it is **st**, the total waiting time for n requests follows an arithmetic progression with a common difference **st**.

Arithmetic progression

$$sum = n/2(2 * a + (n-1)d) \tag{3.3}$$

Where a is the first term corresponding to tw_1 and d is the common difference corresponding to st.

Therefore

$$tw_t = n/2(2 * tw_1 + (n-1)st) \tag{3.4}$$

but tw(1) = 0, then

$$tw_t = n/2(n-1)st \tag{3.5}$$

Service time is the time taken to complete a request i.e time interval between the beginning and end of execution of a particular request in a service, **st**. Assume each request takes equal time to execute, then,

$$st_1 = st_2 = st_3 = \cdots = st_n = st$$

total service time will be

$$st_t = \sum_{i=1}^{n} st_i \tag{3.6}$$

Therefore,

$$st_t = n * st = nst \tag{3.7}$$

JADE operates only on hosts with the Mobile Agent System (MAS) previously installed; therefore it suffers a transfer delay equivalent to (time to save state + time to sign off the platform)

$$TD = ts + ti \tag{3.8}$$

The JADE needs to be activated on getting to its destination by the agent platform. The agent is authenticated and accepted, a hierarchical name space is provided for the agent before the agent is allowed to communicate and migrate and its internal state is restored.

Activation time of the JADE is equivalent to

$$AT = ta + tp + tc + tr \tag{3.9}$$

The service time for n requests

$$st_t = nst$$

waiting time

$$tw_t = n/2(n-1)st \tag{3.10}$$

JADE suffers another transfer delay on the response operation

$$TD_2 = ts + ti \tag{3.11}$$

The total delay

$$D_{TA} = TD + AT + tw_t + st_t + TD_2 \tag{3.12}$$

If we assume that equal delay is suffered during request and response operations, then

$$TD = TD_2$$

Therefore

$$D_{TA} = 2TD + AT + tw_t + st_t \tag{3.13}$$

$$= 2(ts + ti) + ta + tp + tc + tr + n/2(n-1)st + nst$$

$$= 2(ts + ti) + ta + tp + tc + tr + n/2(n+1)st$$

(3.14)

Proposed Embedded Mobile Agent (EMA)

The proposed agent connects directly with the operating system and needs no MAS. It suffers a transfer delay that is equivalent to the time it it takes to save its internal state.

$$TD = ts$$

Activation time

Since there is no agent platform, the mobile agent interacts directly with the operating system on the destination host. Its activation time is limited to the time taken to authenticate and accept agent and the time to restore the agent internal state.

$$AT = ta + tr \tag{3.15}$$

Service time for n requests remains the same as for JADE.

$$st_t = nst \tag{3.16}$$

Waiting time also remains the same

$$tw = n/2(n-1)st \tag{3.17}$$

Transfer delay on response operation is

$TD = ts$

The total delay for the proposed EMA agent is equivalent to

$$\begin{aligned} D_{TB} &= TD + AT + tw_t + st_t + TD \\ &= 2TD + AT + tw_t + st_t \\ &= 2(ts) + ta + tr + n/2(n-1)st + nst \\ &= 2(ts) + ta + tr + n/2(n+1)st \end{aligned} \tag{3.18}$$

Assuming X number of hosts are visited, the delay on X host will be $X*D_{TA}$ and $X*D_{TB}$.

$$D_{TA} = X(2(ts + ti) + ta + tp + tc + tr + n/2(n+1)st)$$

$$D_{TB} = X(2ts + ta + tr + n/2(n+1)st)$$

For simplicity, lets assume that it takes equal amount of time to save agents' internal state and sign off from agent platform i.e

$$ts = ti = t_h$$

also, lets assume

$$ta = tp = tc = tr = t_d$$

then

$$D_{TA} = X(4t_h + 4t_d + n/2(n+1)st) \quad = X(4(t_h + t_d) + n/2(n+1)st) \qquad (3.19)$$

$$D_{TB} = X(2t_h + 2t_d + n/2(n+1)st) \quad = X(2(t_h + t_d) + n/2(n+1)st) \qquad (3.20)$$

3.12.2 Memory Utilization

This refers to the amount of memory space requirement for each mobile agent. This performance is measured using the memory requirement for each system against the number of nodes in the network. Agents tend to be small in size, they do not constitute a complete application by themselves, but they form one by working with agent host and other agents. Agents are small and have limited functionality on their own.

JADE

The communicating computers in the network have the mobile agent system MAS (agent platform) previously installed on them to be able to communicate with the mobile agent, the platform on one computer and several containers distributed over the other computer systems in the network. For the purpose of this study and for uniformity same size is assumed for both the platform (main container) and other containers.

Assuming the size of the MAS is XMB

The size of the mobile agent code is YMB

Size of request or response is Rmb

On a machine, the memory space required by architecture A will be

$$M_A = X + Y \qquad (3.21)$$

If we assume n number of communicating computers

$$TM_A = nX + Y + R \qquad (3.22)$$

and if we assume m number of requests and or response, then

$$TM_A = nX + Y + mR \qquad (3.23)$$

Note that mobile agent is assumed to be on one host at any point in time.

Proposed Embedded Mobile Agent (EMA)

The proposed agent is enhanced with the capability to interact directly with the operating system on any host and needs no MAS, rather a lightweight static agent embedded in the operating system

Assume the size of the mobile agent code is YMB

Let the size of the embedded infrastructure, the lightweight static agent that runs as the OS service be PMB, (P is so small compaired to the size of a MAS). This is because the static agent runs continuously and will take a part of the main memory.

Then, the memory space requirement for the enhanced agent will be

$$M_B = Y + P + R \tag{3.24}$$

Since there is no need for MAS

For m number of requests and n communicating computers

$$TM_B = Y + nP + mR \tag{3.25}$$

3.12.3 Denial of Service

Denial of service represents the fraction of unexpectedly terminated services out of the total number of services executed during the simulation time. For example, a service will be denied if while executing on a host, the computer shuts down, the execution is terminated abruptly or a failure occurs at the node or the network fails. In this work, the node shut down or node failure is simulated by a number generator that has a time variant probability distribution modelled after the Bernoulli Random Variable with p and q as the variables. Bernoulli Random Variable is a discrete probability distribution which takes success probability p and failure probability $q = 1 - p$. The derivation of this parameter is adapted from (Aderounmu, 2003).

To carry out this performance measure, we assume the probability of failure to be 0.1, i.e. one in every 10 nodes fails, which implies that the service has a state that is modeled after the BRV with $p = 0.9$ (90%) and $q = 0.1$ (10%).

So If X is a random variable with this distribution

$$\Pr(X = 1) = 1 - \Pr(X = 0) = 1 - q = p$$

The probability mass function of this distribution is

$$P_x(x) = P\{X = x\} \tag{3.26}$$

Where x $\Rightarrow \{0,1\}$

Generally, X can be an arbitrary real number,

$P_x(X) = 0$

if the random variable X never takes the value of x.

Note however that from the Normal Axiom for probabilities for any discrete random variable X with

X = { $x_1, x_2, \ldots\ldots\ldots\ldots, \}$

$$\sum_{i=1}^{\infty} P_x(x_i) = 1 \qquad (3.27)$$

The cumulative distribution function ***F*** is also given by

$$F_x(a) = \sum_{x=a}^{\infty} P_x(x) \qquad (3.28)$$

For the special case of Bernoulli Random Variable

$$\sum_{i=0}^{1} P_x(x_i) = 1$$

since i takes on values 0 and 1. Parameters ***p*** and ***q*** are defined as follows:

$P = P_x(1)$ is a number in the range [0,1] and $q = 1 - p$, where the event $\{X = x_1 = 1\}$ is called a success, or network availability and occurs with probability ***p***.

The event $\{X = x_2 = 0\}$ is called a failure and occurs with probability ***q***. In the simulation, it was assumed that the network has a state that is modeled after the BRV with $p = 0.9$ **(90%)** and $q = 0.1$**(10%)**.

Let

$f_A^0(t)$ be a time variate function representing denial of service at time t for architecture A

$f_A^1(t)$ be a time variate function representing successful services at time t for architecture A

Then,

$$\int_{t0}^{tf} f_A^0(t)dt$$

represents the number of denied services between simulation times t_0 and t_f for architecture A

Similarly,

$$\int_{t0}^{tf} f_A^1(t)dt$$

represents the number of total successful services between the times t_0 and t_f spent for architecture A. Hence, the total number of services during the period is given by:

$$T = \int_{t0}^{tf} f_A^0(t)dt + \int_{t0}^{tf} f_A^1(t)dt = \int_{t0}^{tf} (f_A^0(t) + f_A^1(t))dt \quad (3.29)$$

The percentage denial is thus given by

$$D_A = \frac{\int_{t0}^{tf} f_A^0(t)dt}{\int_{t0}^{tf} f_A^0(t)dt + \int_{t0}^{tf} f_A^1(t)dt} * 100 \quad (3.30)$$

For architecture time span t_0 and t_f.

Introducing φ, the life of an agent on each node, which is the maximum time an agent can spend on a node before moving to another. An agent may be denied services if the system is busy doing something else or the user fails to explicitly lauch the host platform or the agent exceeds the maximum allowable time on the host. The percentage timed out services thus becomes,

$$\varphi = \frac{\int_{tf}^{\infty} f_A^0(t)dt}{\int_{t0}^{tf} f_A^1(t)dt + \int_{tf}^{\infty} f_A^0(t)dt} * 100 \quad (3.31)$$

The denied services due to agent time out is included in the total denial of service for both architectures, therefore, the total percentage denied services becomes

$$TD_{JADE} = \frac{\int_{t0}^{tf} f_A^0(t)dt}{\int_{t0}^{tf} f_A^0(t)dt + \int_{t0}^{tf} f_A^1(t)dt} * 100 + \frac{\int_{tf}^{\infty} f_A^0(t)dt}{\int_{t0}^{tf} f_A^1(t)dt + \int_{tf}^{\infty} f_A^0(t)dt} * 100 \quad (3.32)$$

It is also assumed that f is directly related to the probability distribution values of p and q. However, EMA has the capability to take up part of the CPU time any time it arrives its destination, it does not need human intervention, the agent starts execution as soon as it is authenticated. Operating Systems services, being part of the Operating Systems and running in the kernel mode, have higher priorities compared to other applications on top of the operating system, the agent time out is equal to zero with EMA, thus

$$\varphi = \frac{\int_{tf}^{\infty} f_A^0(t)dt}{\int_{t0}^{tf} f_A^1(t)dt + \int_{tf}^{\infty} f_A^0(t)dt} * 100 = 0 \quad (3.33)$$

thus for EMA,

$$TD_{EMA} = \frac{\int_{t0}^{tf} f_B^0(t)dt}{\int_{t0}^{tf} f_B^0(t)dt + \int_{t0}^{tf} f_B^1(t)dt} * 100 \quad (3.34)$$

3.12.4 Fault Tolerance

Fault tolerance is the ability of a system to respond gracefully to an unexpected hardware or software failure. A fault tolerant system as in computer networks, has the ability to continue operation in the event of a failure. Fault tolerance is a measure of robustness or adaptability of a system to breakdown (Oyatokun, 2004). According to Aderounmu (2001), a fault tolerant system degrades gracefully in the face of failure, even though at a lower level of performance. Hosts, agent platforms or agents themselves can fail by crashing, other faults could occur due to programming errors or violation of security systems. Faults that occur in a platform cause all the agent on the platform to fail, fault occuring on the computer host causes the platform to fail which implies that the agents on the platform also will fail. Fault occuring in connection or network will cause loss of message or mobile agents. In cases where the network is unreliable and power supply is also unreliable, once the power supply is off the computer host goes off and both the plaform and agents on it fail.

JADE

In JADE, once power fails, both the platform and agents fail, agents suffer abrupt termination, this can lead to loss of agent in which case no state is saved. Failures inherent in JADE inlude:

i. Platform failure: the JADE containers distributed over the network can fail, thus all agents residing in the containers also fail. When the power is restored the platforms need to be explicitly started or restarted by human user to continue operation. The platform can also fail due to natural or artificial causes like programming errors or violation of security systems.

ii. Host failure: the host computer on which the platform resides fails with power failure, it can also fail due to other causes like virus attack, violation of security systems and programming errors as well, which in turn causes the platform and the agent on it to fail.

Assumption

Assume there are n failed platforms, time to a restore platform $T_{p,}$ if we assume uniform recovery time for all platforms, the time to restore n platforms will be $nT_{p.}$ Let's also assume that there are L nodes that fail, and failure recovery time for each node is equal and it's T_h , then total recovery time for L nodes will be LT_h. If the time to restore an agent to continue its operation is $T_{a,}$ then, the total recovery time will be

$$T_f = L(T_h + T_p) + nT_p + nT_a \quad (3.35)$$

Let T_w be the time to restore power or total down time, adding $T_{w,}$ then, equation (3.35) becomes

$$T_f = L(T_h + T_p) + nT_p + nT_a + T_w \quad (3.36)$$

Proposed Embedded Mobile Agent

The proposed agent is running as a part of the Operating System, (OS service), it takes advantage of the autosave and autorecovery facilities of Windows Operating Systems to save states. Once the power supply returns, and the host is started, the agent's state is restored automatically and the agent can continue its tasks without user's intervention, since there is no platform involved, the failure is restricted to the host alone. In case of a node failure due to other occurences, EMA can determine alternative route to the next node in its itinerary. Mobile agents have the ability to dynamically determine alternative route in the face of failure, in both cases mobile agent could determine alternative route in the case of hosts' failure. If we assume a probability of failure to be 0.1, i.e., one out of every ten hosts fails naturally, and the node at which failure occurs is randomly generated.

Using the same notation as for JADE

T_h time to restore the host, for L nodes, total time will be LT_h

T_a is the time to restore agent

$$T_f = LT_h + T_a \quad (3.37)$$

It also suffers a delay equivalent to the time to restore power, T_w

$$T_f = LT_h + nT_a + T_w \quad (3.38)$$

Where T_w is the delay equals to the total down time. The EMA agents are automatically restored as soon as the host is started, the time to restore the agent is therefore equal to zero, thus the total recovery time for EMA will be

$$T_f = LT_h + T_w \quad .(3.39)$$

The total down time and host recovery time in both systems are generated by a radom number generator, unknown to the system apriori.

3.12.5 Turn Around Time

Turn around time is the interval between when the agent is sent form the origin to visits remote nodes and when it returns with the result to the origin, in other words it is the round trip migration time. The JADE agents when disparched, visits each node and suffers a delay associated with the platform activation.

Assume time to travel between node A and node B is Tv, if it visits X number nodes, it will take X*Tv time to visit all the X nodes, assuming the distances between the nodes are equal, and the speed of the agent is constant. Assume it takes Rt time to return to the origin. In jade the agent suffers a delay equivalent to the total delay measured in (equation 3.19) above.

The total turn around time for JADE agent

$$X * Tv + D_{TA} + Rt \quad (3.40)$$

Cosidering the network traffic and the bandwidth, the speed of the agent is directly proportional to the bandwidth, speed is the distance covered per unit time, while the bandwidth is the amount of data that can be transmitted in a fixed amount of time measured in bit per second (bps), thus

Speed **S** = Distance / time = d/t . Time t to travel between a node P and Q is Tv

$$S = d / Tv \quad (3.41)$$

$$S \alpha B \quad \text{then,} \quad S = CB \quad (3.42)$$

where C is the constant of proportionality and equating the two equations, we have

$$S = d / Tv = CB$$

$$B = (d / Tv) / C = d / Tv * 1 / C \quad (3.43)$$

Where B is the bandwidth and **C** is the proportionality constant, for simplicity, lets assume C = 1, then

$$B = d / Tv \quad (3.44)$$

JADE

The total turn around time at varying bandwidth will be equal to

$$X * Tv * (1/B) + D_{TA} + Rt \quad (3.45)$$

If $Tv = Rt$ and

$$D_{TA} = X(4(t_h + t_d) + n/2(n+1)st)$$

Then,

$$X * Tv * (1/B) + X(4(t_h + t_d) + n/2(n+1)st) + Rt \quad (3.46)$$

If we assume that it takes equal amount of time to transport agent between nodes and on the return trip to the origin, i.e. Tv = Rt, then,

$$TAT_{JADE} = X * (Tv+1) * (1/B) + X(4(t_h + t_d) + n/2(n+1)st) \quad (3.47)$$

$$B = d/Tv$$

it follows that,

$$1/B = 1/d/Tv = Tv/d \text{ then}$$

$$TAT_{JADE} = X * (Tv+1) * (Tv/d) + X(4(t_h + t_d) + n/2(n+1)st) \quad (3.48)$$

if we lump all the requests into one service, i.e n =1, thus n/2(n+1)st = st, then

$$TAT_{JADE} = X * (Tv+1) * (Tv/d) + X(4(t_h + t_d)) + st \quad (3.49)$$

Proposed Embedded Mobile Agent

Using the same assumption as used in JADE, EMA travels to a node in time equivalent to Tv, it will visit X nodes in time X*Tv, its return trip is also equal to Rt, the delay is equal to the total delay derived in equation (4.20). Considering the network bandwidth, the total turn around time for EMA will be equal to

$$X * Tv * 1/B + TD_B + Rt \quad (3.50)$$

$$D_{TB} = X(2t_h + 2t_d + n/2(n+1)st) \quad = \quad X(2(t_h + t_d) + n/2(n+1)st)$$

$$X * Tv * (1/B) + X(2(t_h + t_d) + n/2(n+1)st) + Rt \quad (3.51)$$

Lets assume it takes the same amount of time to travel from node to node and back to the origin,

$$Tv = Rt$$

$$TAT_{EMA} = X * (Tv+1) * (1/B) + X(2(t_h + t_d) + n/2(n+1)st) \quad (3.52)$$

Substituting d/Tv for B

$$TAT_{EMA} = X * (Tv+1) * (Tv/d) + X(2(t_h + t_d) + n/2(n+1)st) \quad (3.53)$$

If n = 1; then

$$TAT_{EMA} = X * (Tv + 1) * (Tv / d) + X(2(t_h + t_d) + st) \quad .(3.54)$$

3.13 Conclusion

These design principles of the embedded mobile agent presented in this chapter demonstrate the provisions Windows Operating System made to extend its services. This implies that the operating system capability can be extended to run mobile agents at the same time the functionality of mobile agents could be extended by improving its architecture.

CHAPTER FOUR

IMPLEMENTATION AND SIMULATION OF EMBEDDED MOBILE AGENT

4.1 Introduction

This section presents the practical implementaion of the proposed system and its functionality. The agents designed in chapter three are implemented in Java, an Object Oriented Programming language. Java is chosen because of its unique capability for network programming, its facility for object serialization and dynamic class loading. The embedded mobile agent stores and retrieves weather information from dispersed databases connected by a network, in this case a local area network. The chapter also presents the results of performance analysis of the developed model.

4.2 The System Overview

This work is directed at retrieving weather information in particular. The system is menu driven as shown in Figure 4.1, and consists of:

(i) Initialize Agent operation: takes the user to the agents' configuration module, which is basically to search for weather information given certain criteria. The module is parametric in nature; the user needs only select search options.

(ii) Update Agent store: takes the user to the weather manager's menu, where weather information is stored in the databases.

(iii) Hide me: hides the mobile agent from the main window so user can do other things.

(iv) Shutdown and exit: explicitly shuts down the agent. This can be restated by launching the mobile agent from the desk stop explicitly and at the restart or boot time of the computer.

4.3 System Components

The system model has the following components, a brief description of the system's components is given.

4.3.1 Weather manager

The weather manager is responsible for recording the weather information at a particular location, e.g. temperature, arthmospheric condition. The weather manager stores weather information in the database, as shown in Figure 4.2. The attributes stored are the location: the town or city of interest, Temperature : the degree of hotness or coldness of the location and the atmospheric conditions.

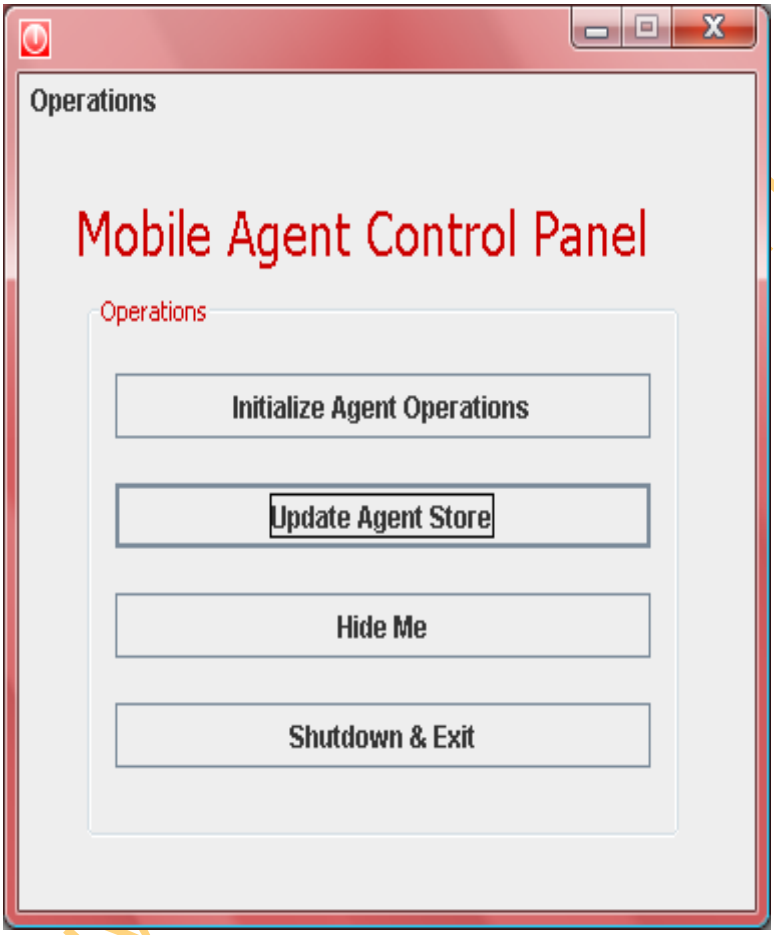


Figure 4.1 Mobile Agent Control panel

Figure 4.2: Weather Manager

4.3.2 Db server

The computer in each location where the weather information is stored, the database maintains the integrity of the information stored in the database. In this work, phpMYadmin was used as Relational Database Management System (RDBMS) access layer to store the databases created. It could be accessed through the xampp web server.

4.3.3 Class config

Class Config Creates the systems configuration, i.e the user interface, where the user interacts with the agent system. This class consists of methods such as config(), the control panel for the system, the control panel is menu driven, choices are made by a click on the available options. The method setView() initiates the originating host environment and opens the network port for communication. The updateView() is called within the constructor to initialize the configuration form.

4.4 The Embedded Mobile Agent (EMA) Implementation

The system of agents implemented in this research work consists of the following:

- Static agent
- Mobile agent
- Agent server

This section presents the implementation details of these agents since their functions are already discussed in the previous chapter.

4.4.1 Static agent

The static agent is written in java programming language and implemented on all the machines in the system. The static agent is installed in the kernel of the operating systems on which it resides, as an extension of the services of the operating systems. In the configuration panel, the name of the machine where the static agent is installed is specified, this is necessary for the agent to be able to communicate with incoming mobile agents.

4.4.2 Mobile Agent

The mobile agent in this work was written in java programming language, taking advantages of java features for serializing and deserializing objects. The mobile agent is received and interacted with by static agent in the kernel of the operating systems on the host visited to read information from the databases of the visited hosts and presents the result to the user.

4.4.3 Agent server

The agent server is implemented in java and is installed on all the machines in the system; it is the interface between the static agent, mobile agent and the network. It opens the port for connection and triggers the static agent to receive incoming mobile agent. The agent server is started at the boot of the computer system.

4.4.4 Agent Creation

Mobile agent is coded through the MobileAgent class containing the logic of the agent. This class trigers the listener methods in the whole system. It automatically initializes and registers the agent and initialize the thread handling using any initialization arguments supplied by the creator.

4.4.5 Agent Removal

The remove(host) class removes a host from the list of the agentEnvironment. closeALL() in MobileAgent class (Superclass), officially closes a mobile agent properly and saves its state. The method oos.close() closes object output stream while soc.close() closes the socket. Closeoperation() closes the “Agent Configuration Panel” and disposes the agent. The agent can also be disposed by the computer shut down.

4.4.6 Migration Process

The system Initiate migration via runAndMove() method, the agent Prepares for moving, selects the next host to migrate to (selectHost()), Interrupts agent by stopping

the thread, serializes the agent (serializeAgent()) and transfers agent's data state and additional info (agent id), registers its presence and throws exceptions in case the destination host in unavailable (logStatus()). The hosts on the network are referenced by the system names or their IP addresses. On its arrival at the destination host, it creates a new instance at the destination with the serialized data and starts the thread of the agent (new Thread(this).start()).

4.4.7 Agent action

After migration, the agent executes run() class which calls the initDB() method that connects the agent to the Database, the logStatus() is initiated to indicate the status of the connection to the database or otherwise. If the db is connected, the search can begin. The seach action is implemented with SQL query embeded in the btSearchActionPerformed(). Two options are available for the search, the temperature and the weather conditions, rbTemp.isSelected() searches for temperature within certain specified range, and rbCondition.isSelected() searches for wheather condition. The btAddActionPerformed() method takes care of the scalability property of the system, it joins new host(s) on the network to the system.

The Figure 4.3 presents the main window where the query is entered by the user. The system is a parametric search system where the user only enters search parameters and these are transformed into structured query language (SQL) by the system and a search is initiated. The list of machines on the network is indicated by the agent's environment, this implies the machines on the network have to be registered before they can be visited. To the right of the window is the result window, where the results of the search are to be displayed. The bottom part of the window is meant to present the report of the search, which includes the host visited, the successful and unsuccessful service requests.

Figure 4.4 presents search results with temperature while figure 4.5 presents the results of searching using atmospheric condition.

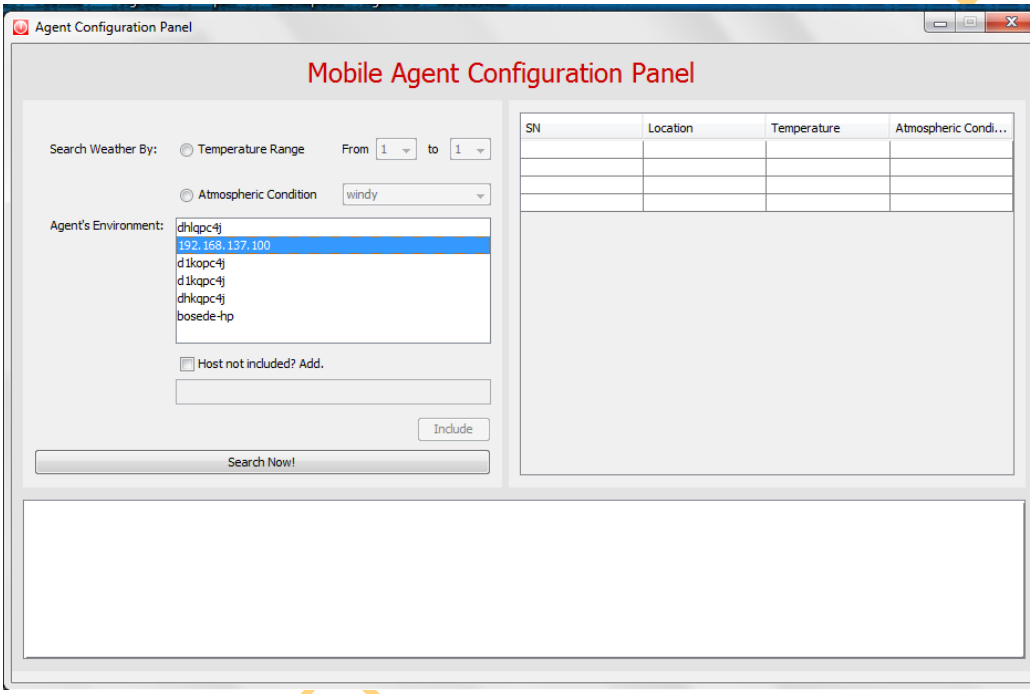


Figure 4.3: Mobile Agent Configuration panel

Search example

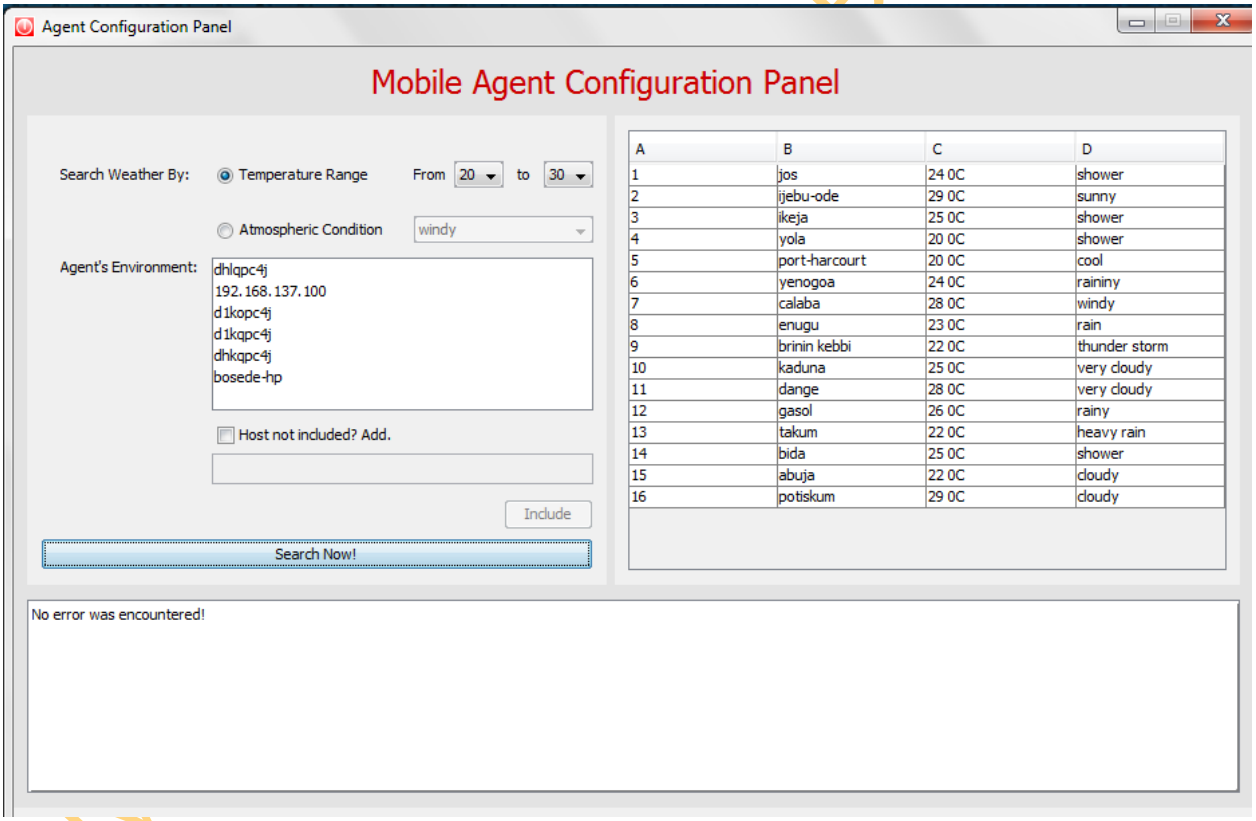


Figure 4.4: Searching with Temperature Range

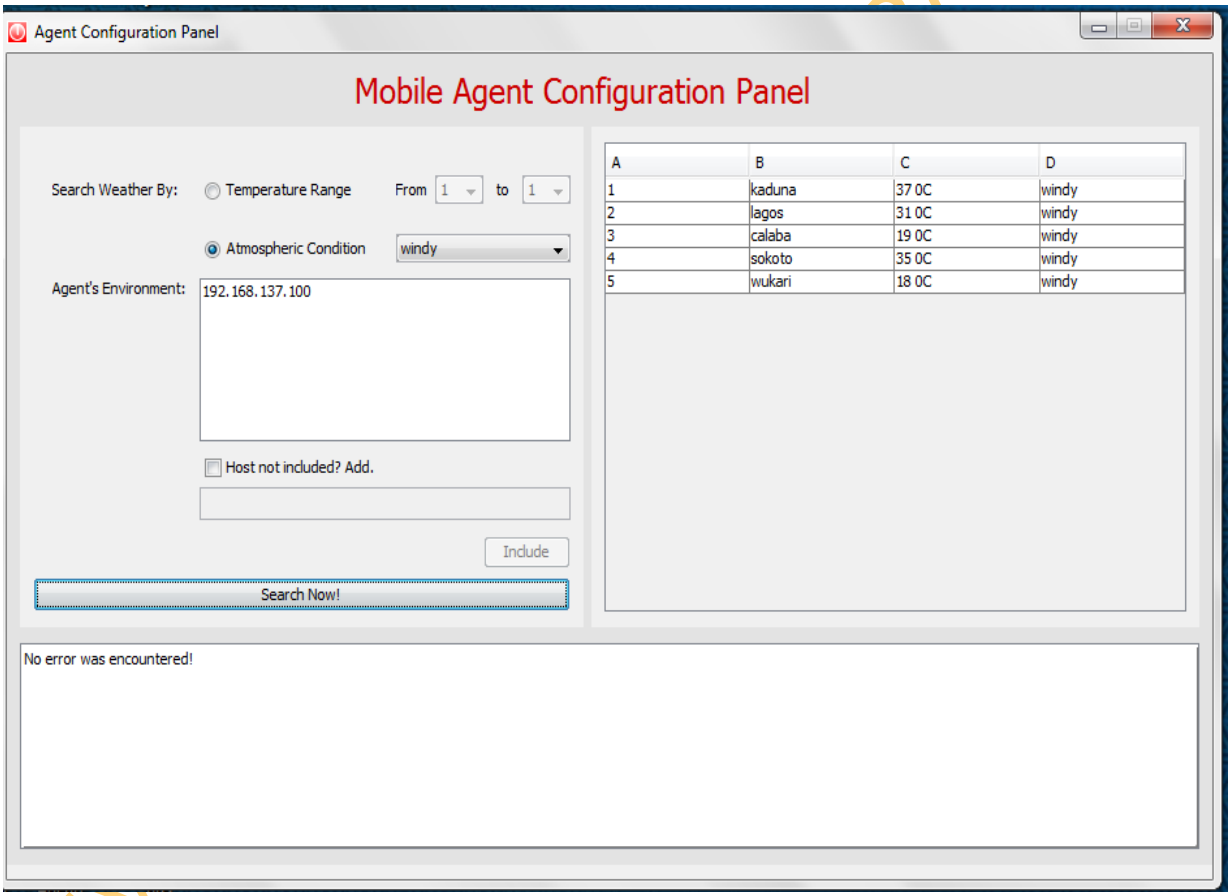


Figure 4.5: Searching with Atmospheric Condition

4.4.8 System Installation Procedure

- install Mysql and java compiler
- Use phpmyadmin to run the SQL in magents.sql
- Install MobileAgentSetup.exe (It will prompt you to run after installation. It will fail)
- Goto start -> programs -> notepad; Right-click it and choose runas administrator to open a blank notepad.
- From the blank notepad, goto File->open and navigate to the installation folder(should be MobileAgent under Program Files if you do not change it)
- Look for the lib folder. Inside it is config.properties
- Edit this file change the host, username, port and password as apply to your database.
- After change, save and launch MobileAgent from the desktop or restart your system. If everything is fine, it will show in the system tray.
- If it fails to run or you encounter further problems, check the log file under the user directory\MobileAgent\ for instance, on my system, the log is under bosede\MobileAgent\.

4.5 Performance model of the Proposed System and JADE

This section presents a performance model of the proposed embedded agent with an existing agent system, (Java Agent DEvelopment framework, JADE). JADE is a widely accepted, open source FIPA compliant mobile agent platform and it is java based thus providing simple and friendly Application Programming Interface. Performance measurements tested includes service delay, memory utilization, fault tolerance, denial of service and turn around time. Simulated results show that the embedded agent offers a superior performance compared to JADE. It offers a lower delay and turn around time, consumes less storage and has reduced percentage denial of service.

Performance management includes a set of activities which ensure that goals of a system are consistently met in an effective and efficient manner. The performance of a system is used to denote its processing power, which is measured in terms of the time

used to solve a given problem or the size and number of problems solved in a fixed amount of time. Performance also includes, the reliability of the system, for example its availability and unavailability times or the times between failures and the functional aspects of performance which includes the correctness of solutions and efficiency with minimal discomfort or physical efforts, this is supported by Kotsis (1999). The traditional performance evaluation of computer systems starts with the characterization of the system under study and a characterization of the load. Then, a performance model is built and performance results are obtained by applying performance evaluation techniques which could be analytical, mathematical or simulation techniques. This is necessary with a view to improving on the existing systems and to define new techniques of solving computational problems taking advantage of technological advancements. According to Aderounmu (2001), performance management is necessary to perform some functions which include to:

(i) continuously evaluate the principal performance indicators of network operation
(ii) verify how service levels are maintained
(iii) identify actual and potential bottlenecks and
(iv) Establish and report trends for management decision making and planning.

Stuck and Arthurs (1985) opined that it is necessary to analyze the performance of a system model for two reasons, which are

i. to improve productivity: that is to increase the number of jobs done in the same unit of time
ii. To add functionality: new functions will be performed that offer the potential for new productivity gains or new revenue.

In view of all these, the researcher performed a performance analysis of the proposed system using mathematical modelling and later compared the results with those of an existing agent system, precisely JADE.

4.6 Simulation and Analysis of Results

Simulation is an attempt to model a real-life or hypothetical situation on a computer so that the system can be studied to see how it works and behaves. Aderounmu (2001) defined simulation as evaluation technique that represents the behaviour of a system by

a model in the time domain. Simulation is a powerful and versatile tool to carry out experiments on the behaviour of a real world situation or system, either because the actual system is too expensive to implement, very difficult or dangerous to achieve or hazardous to human lives. The simulation technique is employed in this research work to model the behaviour of the proposed system against an existing system for information retrieval application. This section presents the simulation program developed to provide performance evaluation of the proposed embedded mobile agent (EMA) against that of an existing scheme (JADE).

4.6.1 Service Delay versus Number of hosts

Service delay was measured against the number of hosts on the network for the two schemes. The mathematical models for the service delay for both JADE and EMA were obtained in, Equations 3.19 and 3.20 respectively. In the simulation the researcher assumed same number of requests for both schemes. Borrowing from the idea of El-Gamal *et al*., (2007) and Mobaideen (2003), assumed time to save agent internal state and time to sign off from the platform **(t_h)** = 5milli-secs, time to activate platform, authenticate and accept agent **(t_d)** = 5milli-secs and service time **(st)** = 5milli-secs. The mean service delay generated by EMA was 15067.50 while that of JADE was 15697.50, as derived from Table 4.1. The result of the simulation as shown in Figure 4.6 showed that the proposed scheme generated slightly lower delay compared to JADE for lower number of nodes up to 40 nodes and clear differences began to appear with increased number of nodes. This was due to the fact that both systems were agent based and the agents in the two systems were written in the same programming language, java. It can be deduced that the EMA generates lower delay as the number of nodes to visit increases, it thus follows that EMA performs better with higher number of nodes in the network.

Table 4.1 Service Delay for the two schemes

No_of_nodes	JADE	EMA
5	1495	1435
10	2990	2870
15	4485	4305
20	5980	5740
25	7475	7175
30	8970	8610
35	10465	10045
40	11960	11480
45	13455	12915
50	14950	14350
55	16445	15785
60	17940	17220
65	19435	18655
70	20930	20090
75	22425	21525
80	23920	22960
85	25415	24395
90	26910	25830
95	28405	27265
100	29900	28700

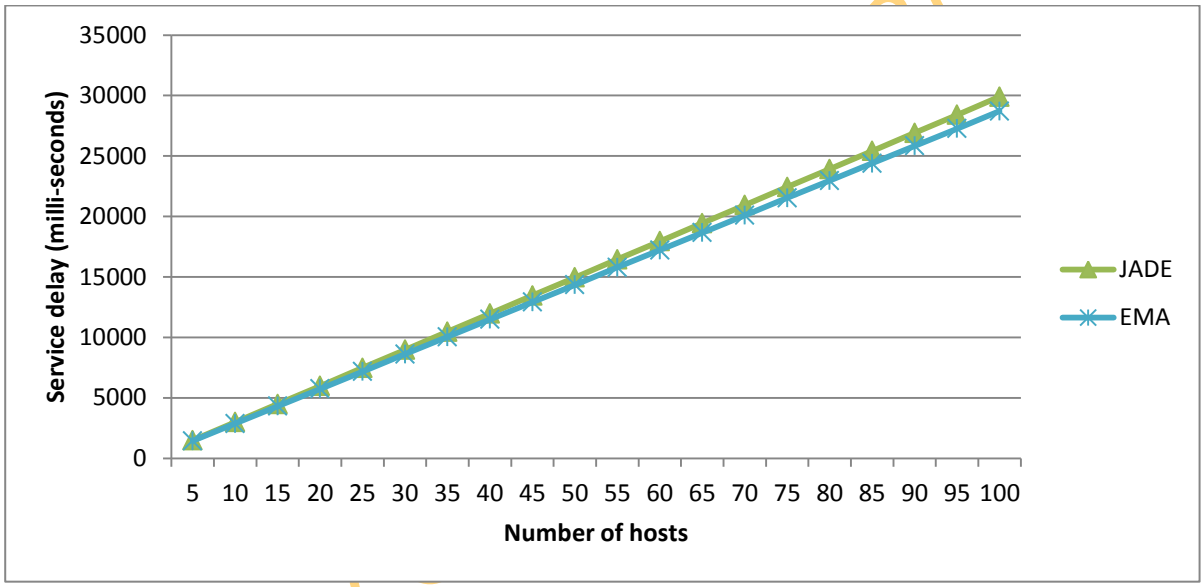


Figure 4.6 : Service delay versus number of host on the network

4.6.2 Memory utilization versus the number of nodes

In measuring the memory utilization of the two approaches, the total memory requirements for each scheme were compared. The total amount of memory required to store the Jade platform and the containers on all the hosts involved in the system and the memory requirement for the mobile agent at any point in time. The mathematical model of the memory requirements was developed in Equations 3.23 for JADE and 3.25 for EMA. The size of the embedded static agent running was so small compared to the size of the JADE platform, precisely 1KB (1022bytes) and the agent server was approximately 2KB (1.27KB), totalling 3KB whereas the JADE platform required about 2.83×10^3 KB. The enhaced mobile agent's size was approximately 6KB (5.52KB) like other java based mobile agents. Bearing in mind that memory is an expensive resource, the increased processing speed of EMA being a prime motivator. Figure 4.7 shows the graph representing the memory requirements for both systems. The simulation results from Table 4.2 show that the embedded Mobile agent scheme utilized a small amount of memory compaired to JADE platform. EMA utilized a mean of 132.50 units of memory while JADE required a mean of 380.00 units for a sample size of 10 computers. The JADE system consumed a whole lot of memory for storage of agent platform and JADE containers on all the systems the agent is to run and this increased as the number of host on the network increased on the average.

Table 4.2: Memory utilization against number nodes

No of nodes	JADE	EMA
5	155	110
10	205	115
15	255	120
20	305	125
25	355	130
30	405	135
35	455	140
40	505	145
45	555	150
50	605	155

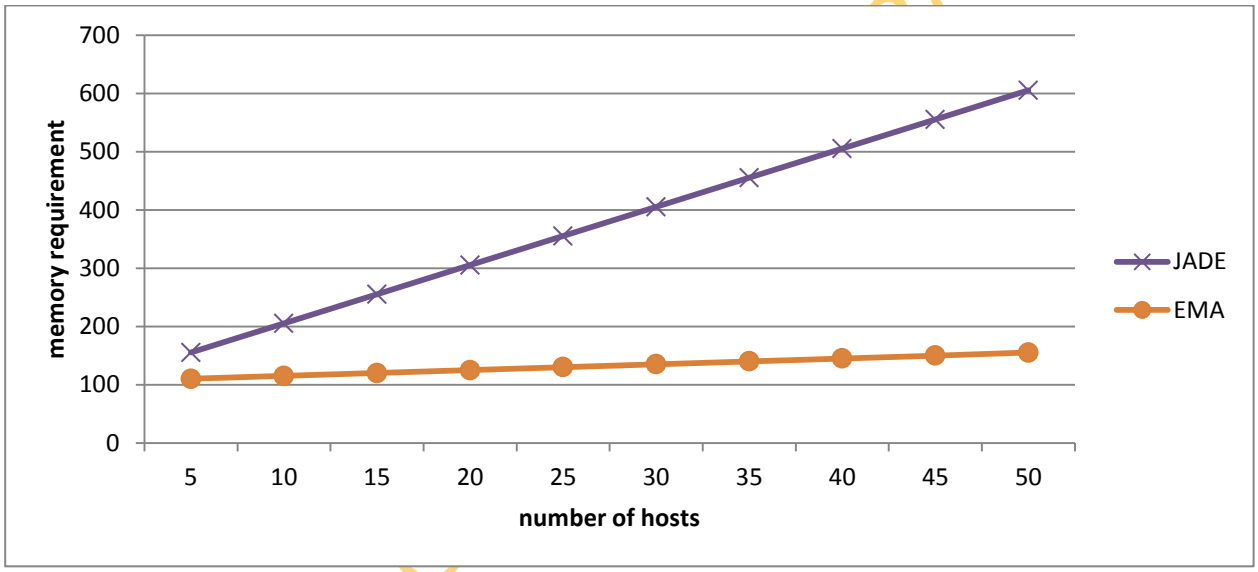


Figure 4.7: Graph of memory utilization against number of hosts on the

4.6.3 Denial of service versus number of request per service

In this simulation, the researcher refers to an abrupt termination of service as denial of service, the researcher measured the adaptability of the two systems at a fixed network bandwidth. The numbers of failed and successful services were measured against the total number of services to measure the percentage denial of service for the two schemes. The mathematical model was developed for the two schemes in Equations 3.32 for JADE and 3.34 for EMA. The number of failed services was randomly generated using a random number generator that follows the Bernoulli Random Variable (BRV) with a probability of 0.1 failure. The result of simulation as presented in table 4.3 shows an improvement in the reduction of denied services for EMA, with a mean percentage denial of service at 14.28% while that of JADE was 24.74 %. From the graph for the result as shown in Figure 4.8, it can be deduced that at every point in time, the total number of services denied for JADE were higher because of the many points of failure that contribute to services being denied. EMA on the other hand showed a consistent reduction in the total number of services denied throughout the simulation period with a maximum of 40.55% denied services while JADE is inconsistent with a maximum of 72.56% denied services.

Table 4.3: Percentage denial of service for JADE and EMA

Noofservices	JADE	EMA
1	11.93258	10.716640
2	21.58643	12.784850
3	58.66181	23.811520
4	26.58483	11.414270
5	32.61682	21.763570
6	23.29993	13.281320
7	59.20310	30.122150
8	11.83280	1.854740
9	25.30223	12.043000
10	20.65293	8.570012
11	16.06390	13.805890
12	10.11700	9.797118
13	8.65884	2.129710
14	30.41261	22.685170
15	8.01970	1.721273
16	19.18677	17.108090
17	20.58117	17.590060
18	13.12286	9.622582
19	4.30178	4.269979
20	72.56191	40.552730

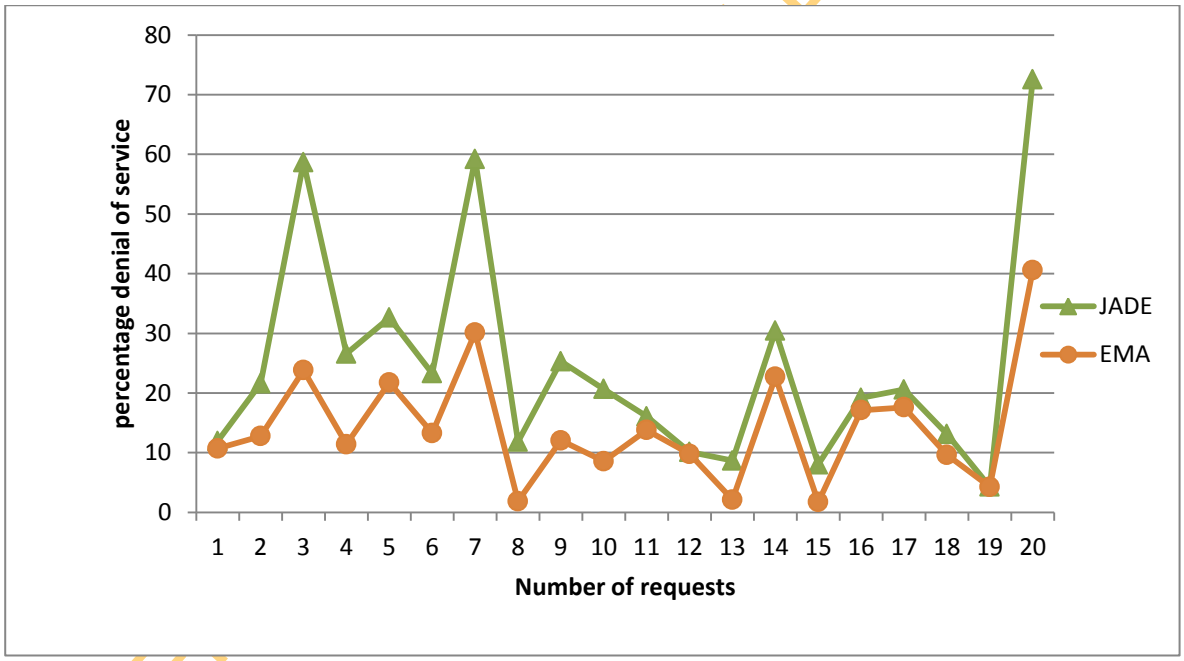


Figure 4.8: percentage denial of service for varying number of

4.6.4 Fault tolerance

In measuring fault tolerance, the failure recovery times for various number of nodes were measured. In cases of power failure,(which is an issue in this environment) JADE agents suffered abrupt termination and this can lead to loss of agents in which case, no state is saved, the platform needs to be explicitly restarted when power is restored. When power fails, EMA saves states and automatically recovers when power is restored this is enhanced by the auto recovery facility of Windows operating systems. The result of simulation as presented in Table 4.4, EMA shows a superior performance with a mean of 180.31, over JADE with a mean of 327.84, this was attributed to the time it took to activate the platform and restart the agents on the platforms which did not apply to EMA. The failure recovery time for EMA was about 50% that of JADE. Figure 4.9 shows that as we have more nodes on the network, there is the possibility of fault occuring which resulted in higher recovery times. The total failure recovery time was generated by a number generator unknown to the system apriority. The failure recovery times for EMA were almost linear for lower number of nodes up to 15 and increased as the number of nodes increased.

Table 4.4: Fault tolerance measured in term of failure recovery times for JADE and EMA

Number-of-nodes	JADE	EMA
1	95.05459	94.51451
2	209.92050	126.65540
3	209.05810	26.24645
4	469.92660	234.80920
5	267.50970	114.05570
6	95.88279	86.98524
7	392.45230	116.65720
8	120.68890	94.61911
9	274.89330	194.70500
10	330.37450	76.50851
11	172.06060	73.40134
12	183.22440	120.36270
13	179.99110	140.36110
14	571.97770	260.50240
15	203.29790	148.71410
16	518.31100	348.45130
17	592.73400	366.86050
18	485.44400	313.15810
19	798.83160	363.56290
20	385.19440	305.05180

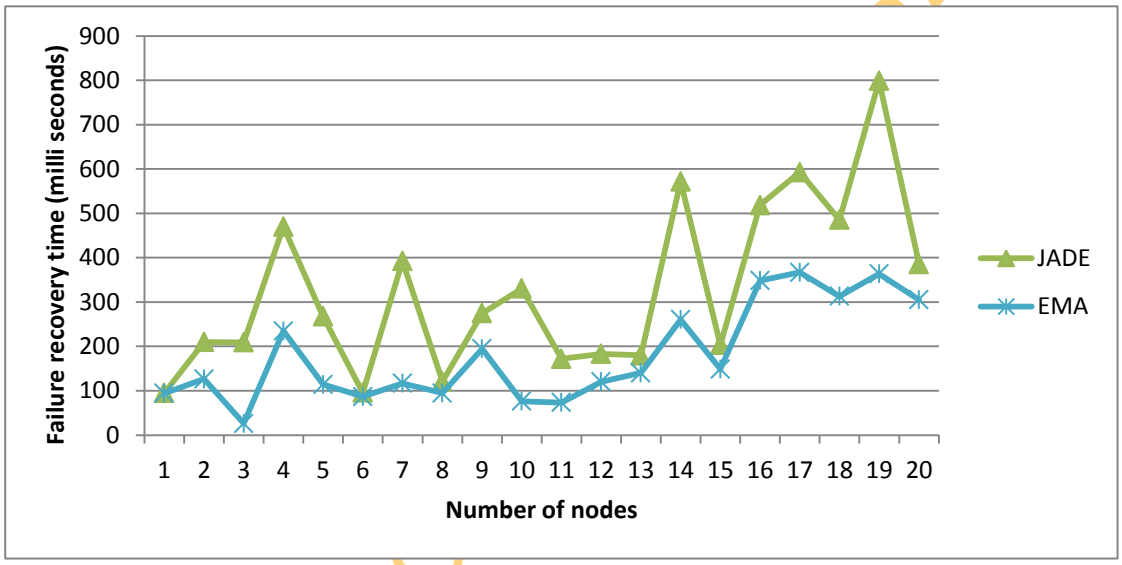


Figure.4.9: Failure recovery time for different number of nodes

4.6.5 Turnaround Time

The turn around time is the total time it takes an agent to visit all the hosts in its itinerary and return to the origin. In measuring the agent turn around time, the round trip times of mobile agent were measured with respect to the number of nodes it has in its itinerary to complete its tasks. The speed of the agent depends largely on the network bandwidth and/or network speed. The network speed is simulated with a stochastic model while measuring the times of agent itinerary. The result of simulation as presented by the graph of Figure 4.9 shows that the turnaround time increased linearly for the first three nodes, but as the agent migrates to more nodes the behaviour is affected by the network bandwidth and it became nonlinear. The turnaround times for EMA are consistently lower than that of JADE for same number of nodes, this implies that the EMA agents ran and returned to the origin earlier than JADE agents. The two schemes have similar behaviour because they are both java-based agents thus reacted same way to network bandwidth, but the enhanced agent showed an improvement in reduction of its round trip times. From the results as shown in table 4.5, the mean turnaround time for EMA was 499.71 while the mean for JADE was 843.33. This was attributed to the time JADE spent activating the platform on each node. EMA agent took considerably lower times for its round trips at varying number of hosts compared to JADE agent.

Table 4.5: Turnaround times for JADE and EMA systems

Numberof hosts	JADE	EMA
1	482.247	317.8067
2	761.4957	434.2923
3	1041.479	545.8186
4	378.6712	234.4612
5	1103.012	634.3623
6	617.6008	379.6173
7	646.1122	429.5757
8	955.9235	592.7742
9	854.1171	554.7585
10	932.2553	583.6081
11	489.4374	305.4458
12	1103.371	722.2738
13	414.1125	254.2512
14	1599.922	893.1526
15	1068.697	562.6228
16	769.3093	501.5302
17	868.3988	542.8115
18	953.6566	565.5141
19	1341.773	679.7233
20	485.0498	259.7475

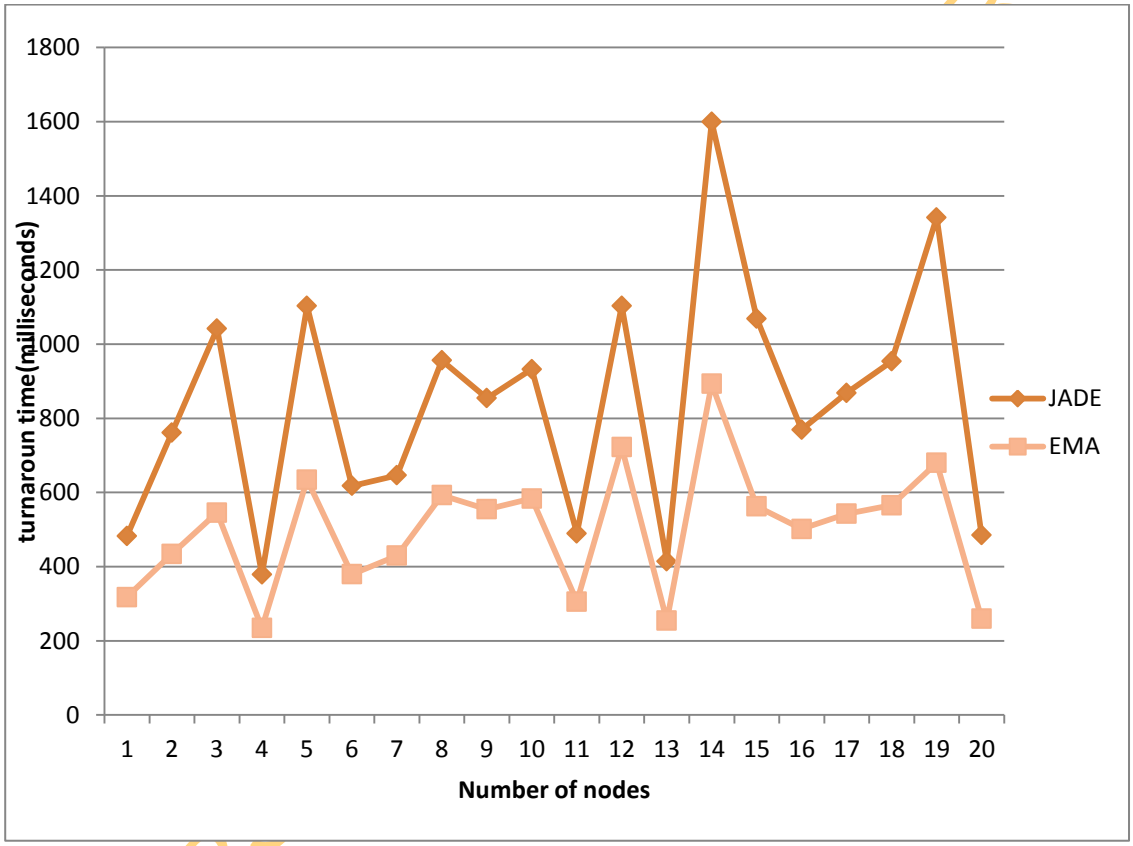


Figure 4.10: Mobile agent turn around times for various number of host visited

4.7 Statistical Analyses

Statistical analyses were carried out based on the simulated data to determine the significance level of the proposed system and the relationship between the proposed scheme and the existing scheme. Independent T-Tests and correlation analyses were performed on the data obtained in the previous section in order to ascertain the significant difference level between the two schemes at a significance level of 0.01.
Student T-test (independent samples): compares two small sets of quantitative data when samples are collected independently one of another. This fits the data obtained from the simulation as the two schemes are independent of one another as well as their data. The results are presented in the Tables 4.6 to 4.9.

4.7.1 T-Test and Correlation

The statistics of the two data sets are presented in table 4.6 below showing the mean, standard deviation and the mean of standard error.
Statistical correlation is used to evaluate the strength of relations between two variables or a set of data. According to Olubusoye *et al* (2001) correlation measures the degree of linear association between two or more variables when a movement in one variable is associated with the movement in another variable either in the same direction or the opposite direction. One of the most common correlation coefficients, the Pearson Correlation coefficient, *P* or *r,* which is sensitive to a linear relationship between two variables, was employed in this analysis. Where *P or r* is the measure of dependence, or the measure of the degree of correlation. The Pearson's correlation corresponds to the following values

+1: the case of perfect positive linear relationship
-1: the case of perfect decreasing or negative linear relationship
Values between -1 and +1: indicate degree of linear dependence and
0: the case of no relationships.

Table 4.6: Statistics of Data

Samples Statistics

		Mean	N	Std. Deviation	Std. Error Mean
Fault Tolerance	JADE	**327.8413968**	**20**	**193.07697001**	**43.17332298**
	EMA	**180.3091321**	**20**	**109.14322721**	**24.40516753**
Percentage Denial of Service	JADE_1	**24.7349991**	**20**	**18.51132154**	**4.13925733**
	EMA_1	**14.2822339**	**20**	**9.84454775**	**2.20130780**
Turn Around Times	JADE_2	**843.3319936**	**20**	**321.64317727**	**71.92160089**
	EMA_2	**499.7073844**	**20**	**172.97856373**	**38.67918271**
The Service Delay	JADE_3	**15697.5000000**	**20**	**8844.53927573**	**1977.69910502**
	EMA_3	**15067.5000000**	**20**	**8489.57448875**	**1898.32656569**
Memory Utilization	JADE_4	**380.0000000**	**10**	**151.38251770**	**47.87135539**
	EMA_4	**132.5000000**	**10**	**15.13825177**	**4.78713554**

Table 4.6 shows the analysis of data, the mean value of both systems for each of the performance parameters, their standard deviation about the mean and standard mean errors. Table 4.7 presents the result of the correlation analysis at a significance level of 0.01 for all the parameters tested. The results obtained show that there is a strong positive to perfect positive correlations between the two systems for the parameters tested.

4.7.2 Student's Independent T-Test

The t-test compares two small sets of quantitative data when samples are collected independently of one another. The t-test measures the level of significant differences between two sets of data collected independently. T-test was also applied to the data obtained for the performance metrics of the two schemes at a significant level of 0.01.

4.7.3 Interpretation of Results

1. For the fault tolerance of the two systems, the t-value of 5.726 and p-value of 0.000 indicate that the test is statistically significant. This implies that there is significant difference between the two systems. The average fault recovery time for EMA was (180.31 ms) and the standard deviation was 109.14 while that of JADE was (327.84 ms) with193.08 standard deviation. The correlation value of 0.852 and p-value of 0.000 indicate that there is a strong positive significant relationship between the two systems.
2. For the percentage denial of service for the two systems, the t-value of 4.533 and p-value of 0.000 indicates that the test is statistically significant. This implies that there is significant difference between the two systems. The mean percentage denial of service for EMA was 14.28 with a standard deviation of 9.8 while that of JADE was 24.73 with a standard deviation of 18.5. The correlation value of 0.914 and p-value of 0.000) indicate that there is a strong positive significant relationship between the two systems.

Table 4.7: Correlations Table

		N	Correlation	Sig.
Fault Tolerance	JADE & EMA	**20**	**.852**	**.000**
Percentage Denial of Service	JADE_1 & EMA_1	**20**	**.914**	**.000**
Turn Around Times	JADE_2 & EMA_2	**20**	**.970**	**.000**
The Service Delay	JADE_3 & EMA_3	**20**	**1.000**	**.000**
Memory Utilization	JADE_4 & EMA_4	**10**	**1.000**	**.000**

Table 4.8: Samples Test Table

		Paired Differences				
					95% Confidence Interval of the Difference	
		Mean	Std. Deviation	Std. Error Mean	Lower	Upper
Fault Tolerance	JADE – EMA	**147.53226470**	**115.22369865**	**25.76480228**	**93.60591377**	**201.45861563**
Percentage Denial of Service	JADE_1 - EMA_1	**10.45276525**	**10.31346517**	**2.30616092**	**5.62591498**	**15.27961553**
Turn Around Times	JADE_2 - EMA_2	**343.62460911**	**159.42989310**	**35.64960786**	**269.00912232**	**418.24009589**
The Service Delay	JADE_3 - EMA_3	**630.00000000**	**354.96478699**	**79.37253933**	**463.87136592**	**796.12863408**
Memory Utilization	JADE_4 - EMA_4	**247.50000000**	**136.24426593**	**43.08421985**	**150.03672346**	**344.96327654**

Table 4.9: T-Test Samples Test

		T	Df	Sig. (2-tailed)
Fault Tolerance	JADE – EMA	**5.726**	**19**	**.000**
Percentage Denial of Service	JADE_1 - EMA_1	**4.533**	**19**	**.000**
Turn Around Times	JADE_2 - EMA_2	**9.639**	**19**	**.000**
The Service Delay	JADE_3 - EMA_3	**7.937**	**19**	**.000**
Memory Utilization	JADE_4 - EMA_4	**5.745**	**9**	**.000**

3. For the turn around times of the two systems, the t-value is 9.639 and p-value of 0.000 indicates that the test is statistically significant. This implies that there is significant difference between the two systems. The mean turnaround time for EMA was 499.71 ms with standard deviation of 172.98 while that of JADE was 843.33 ms with 321.64 standard deviation. The correlation value of 0.970 and p-value of 0.000 indicate that there is a strong positive significant relationship between the two systems.
4. For the service delay of the two systems, the t-value is 7.937 and p-value of 0.000 indicates that the test is statistically significant. This implies that there is significant difference between the two systems. The mean service delay for EMA was 15067.5 ms with 8489.57 standard deviation while that of JADE was 15697.0 ms with 8844.54 standard deviation. The correlation value of 1.000 and p-value of 0.000 indicate that there is a perfect positive significant relationship between the two systems.
5. For the memory utilization of the two systems, the t-value is 5.745 and p-value of 0.000 indicates that the test is statistically significant. This implies that there is significant difference between the two systems. The mean memory requirement for EMA is 132.5 bytes with 15.14 standard deviation while that of JADE was 380.0 bytes with 151.38 standard deviation. The correlation value of 1.000 and p-value of 0.000 indicate that there is a perfect positive significant relationship between the two systems.

The outcome of this work shows a significant difference between EMA and JADE in terms of memory utilization, denial of service and greater fault tolerance in the face of failure, turnaround time and service delay. The correlation analysis shows there is a strong positive to perfect positive relationships between the two systems. The standard deviation according to Spiegel and Stephens (1999) is a measure of dispersion, thus the standard deviations of almost all the parameters' tested are high due to the fact that the data obtained at simulation times for each parameter are spread out over a broad range. This is because any slight alteration in the parameters has a significant cumulative effect on the performance, since both systems are agent based, they respond similarly to changes in parameters.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATION

5.0 Introduction

This chapter provides the summary of this thesis. The overall picture of the thesis, problem solved, solution approach and issues examined are presented; the contributions of the thesis to knowledge are also highlighted. This is followed by the conclusions reached and finally, we discuss a number of future research directions identified in the work.

5.1 Summary

Mobile agent paradigm provides an open and generic framework for distributed applications development. Agents solve complex software problems in distributed environments where protocols, operating systems, hardware and runtime environments are heterogeneous. Agent technology has received attention in the academic community over the years, but it is yet to be adopted by the commercial community as a natural evolution of object oriented world (Bellavista *et al*., 2001). This is largely due to its complexity and lack of standardization and interoperability (Stoian and Popirlan,

2010). Existing agents operate and execute on computers with the mobile agent platforms previously installed, the platforms provide runtime execution for the mobile agents. These platforms differ in architecture, language and implementation and are not interoperable, i.e. an agent built on one platform cannot execute on another platform. The fact that mobile agent are used in distributed environment where hardware, protocols, Operating Systems and platforms are expected to be heterogeneous, necessitates a new way of implementing mobile agents to make them operate independent of the agent platforms.

The work focuses on enhancing the architecture of mobile agents so as to extend their functionalities. The model developed employ multi-agent mechanism to take care of agent authentication, migration and any changes that may occur as the mobile agent migrates from node to node. The model was implemented in java and the target operating system is Windows XP, the model was also compatible and run efficiently on Windows Vista and Window7. The agent is automatically started at the boot of the computer system, it does not need to be explicitly initiated like the existing agents and it is instantiated to perform its tasks. The mobile agent developed in this work is intelligent, it visits the nodes listed in its itinerary and in case of failure or non availability of a node in the list, it dynamically determines alternative route to the next node until all the nodes listed in its itinerary are exhausted, then it returns to its origin with the results. At the origin the mobile agent delivers the result of its actions and also reports any unavailable node in its itinerary. The agent designed in this research work migrates through the local area network and retrieves information stored in the databases located in the computers on the network.

The proposed system was compared with an existing system, JADE, that is also java-based agent platform. Simulation program was developed to provide performance analysis between the existing scheme and the proposed scheme. Mathematical models were developed for the following performance metrics: service delay, memory utilization, denial of service, fault tolerance and turnaround times. The parameters used for the measurements are as follows:

(i) Service delay against the number of hosts on the network
(ii) Memory utilization against the number of hosts
(iii) Percentage denial of service against the number of requests

(iv) Failure recovery times against the number of nodes

(v) Turnaround time against the number of hosts visited

5.2 Contribution to Knowledge

In this work, an embedded mobile agent was implemented and deployed as an operating system service; the execution of the EMA needs no agent platform, thus preserving memory. This operating system service runs continuously in the background while users run other programs simultaneously. EMA is initiated as soon as the operating system boots, eliminating delay associated with platform activation. Furthermore, this work shows that users programs can be embedded in the kernel of Windows Operating System extending the OS services giving an impression of programming the operating systems. In the same vein, enhancing the architecture of mobile agent can improve its performance. Since all computers run on operating system, the model developed can be deployed and used across organisations, therefore, improving the wide acceptability of mobile agent paradigm.

5.3 Conclusion

In this work, the researcher proposed a new way of implementing mobile agents (EMA), that does not need the installation of agent platforms on the hosts on the network, rather a light-weight static agent that runs as a windows service each time the system boots.

The results obtained in this research show that the functionalities of mobile agents can be extended by improving its architecture and that the proposed embedded mobile agent can significantly improve the use of mobile agent technology in the distributed applications. The system successfully eliminates the agent platforms on which mobile agents rely for execution and provides a light-weight agent that's resides in the

operating system. Thus, saving memory that would have been used for the agent platform, reducing access time for agent operations, reducing the turn around times for mobile agent and increasing the fault tolerance capability of the system as a whole. The fact that the agent is embedded in the Operating System running on the host computer reduces the complexity of agent technology, which the OS hides from the user, makes agents from different vendors with different design and language interoperate, as well as makes the system portable across different organizations. Therefore, it could be deduced that the EMA for information storage and retrieval provides a superior, efficient and autonomous scheme with a high level of flexibility than the existing JADE scheme.

5.4 Recommendations for Future Research

This thesis focuses on enhancing the architecture of mobile agents to directly interact with the operating system. Windows XP was chosen as a test case and the implementation was extended to Windows Vista and Windows 7. The system implemented in this research could be incorporated into new versions of operating systems for universal distributed information retrieval. Therefore, the implementation of the system on other Operating Systems such as UNIX, Linux, Mac OS, and Solaris is recommended for future research.

The Embedded Mobile Agent was applied to information retrieval in distributed environment. Future research could investigate the application of EMA to complex and more sophisticated operations such as data mining, intrusion detection and so on. Furthermore, provision of adequate security for the embedded mobile agent would be a subject for future research. The interoperability capability of this model with other mobile agent systems is also recommended for investigation.

REFRENCES

Aderounmu, G.A. 2001. Development of an intelligent mobile agent for computer network performance management. Unpublished PhD thesis, Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria.

Aderounmu, G.A. 2003. Performance Comparison of Remote Procedure Calling and Mobile Agent Approach to Control and Data Tranfer in Distributed Computing Environment. *Journal of Network and Computer Applications, Elsevier*, 27: 113-129.

Aderounmu, G.A, Oyatokun B.O. and Adigun M.O. 2006. Remote Method Invocation and Mobile Agent: a Comparative Analysis. Issues in Informing Science and Information Technology, 3. Available at *http://informingscience.org/proceedings.INSITE2006/IISTAder188.pdf*

Adewunmi R. 2002. Distributed system: Concept, model and Issues. Seminal paper: International School on Industrial Software Engineering, held at the University of Lagos, Akoka, Lagos, Nigeria.

Admassu T. 2008. Threats and trusted countermeasures, using a security protocol, in the agent space. Unpublished M Sc thesis, Department of Computer Engineering, Addis Ababa University, Ethiopia.

Ahmed M.E. 2007. A new approach in learning for intelligent multi agent systems. Proceedings of 21st European Conference on Modelling and Simulation, Ivan Zelinka, Zuzana Oplatkova Orsoni ECMS 2007.

Ajay Kr. S, Ravi S. and Vikram J. 1999. Design Patterns for Mobile Agent Applications. In workshop on Ubiquitous Agents on Embedded Wearable and Mobile Devices, Italy, 1999.

Angeletti M., Culmone R. and Merelli E. 2001. An intelligent agents architecture for DNA-microarray data integration. Proceedings of the NETTAB workshop on Corba and XML: towards a bioinformatics integrated network environment, Genova.

Aridor Y. and Lange D. 1998. Agent design patterns: elements of agent application design. Proceedings of the Second International Conference on Autonomous agents (Agents '98), ACM press, 1998, 108-115.

Ashvin G. 2004. Advances in distributed system: an introduction. Date of last access: 8 February, 2008 at http//www.eecg.toronto.edu/~ashvin/coursesece1746/2004/introduction.pdf

Bellavista P., Corradi A. and Stefanelli C. 2000. Protection and interoperability for mobile agents: a secure and open programming environment. IEICE Trans. Commun, E83-B (5): 961-972.

Bellavista P., Corradi P. and Stefanelli C. 2001. Mobile agent middleware for mobile computing. IEEE Computer Society, Washington DC, USA, 73-81.

Bellifemine, F.L, Greenwood D and Caire G. 2007. Developing Multi-agent systems with JADE. John Wiley & Sons Ltd, England.

Biermann E. 2004. A Framework for the Protection of Mobile Agents Against Malicious Hosts. Unpublished Ph.D thesis, Universityf South Africa, South Africa.

Bohoris C. 2003. Network Performance Management Using Mobile Software Agents. Unpublished PhD thesis, University of Surrey, Guildford, Surrey, UK.

Borselius N. 2002. Mobile Agent Security. Electronics and communication Engineering Journal, 14 (5),IEEE, London, UK, pp 211-218.

Braun P. and Rossak W. 2005. Mobile Agents basic concepts, mobility models and Tracy toolkits Elsevier Inc (USA) and dpunkt.verlag (Germany)

Brewington B., Gray R., Moizumi K., kotz D., Cybenco G and Rus D. 1999. Mobile Agent in Distributed Information Retrieval. Thayer School of Engineering, department of Computer Science Dartmouth College Hanover, new Hampshire. Firstname.lastname@dartmouth.edu

Carzaniga A., Picco G. P and Vigna G. 1997. Designing Distributed Applications with Mobile Code Paradigms. Proceedings of the 19th International Conference on Software Engineering (ICSE '97), 22 -32, ACM press, 1997. Retrieved on 20 July, 2011 from www.cs.ucsb.edu/~vigna/.../1997_carzaniga_picco_vigna_ices97.ppt

Chalopin J., Godard E., Metivier Y and Ossamy R. 2006. Mobile Agent Algorithms versus Message Passing Algorithms. Proceedings of tenth International Conference OPODIS, 2006, Bordeaux, France, December, 2006, 187 – 201, Springer-Verlag.

Chen B., Chen H. H and Palen J. 2009. Integrating mobile agent technology with multi-agent systems for distributed traffic detection and management systems. Transportation Research Part C, 1-10.

Chess D., Harrison, C and Kershenbaum A. 1994. Mobile Agents: are they a good idea? Technical Report, IBM Research Division, T.J Watson Research Centre, Yorktown Heights, New York.

Christopher K. and Thomas T. 2001. Applying Mobile Agent technology to Intrusion Detection. Distributed Systems Group, Technical University Vienna, Austria.

Clark K. L. and Lazarou V. S. 1997. A Multi-Agent System for Distributed Information retrieval on the World Wide Web. Retrieved on May 15, 2012 from http://www.inf.ed.ac.uk/teaching/courses/irm/reviews/clark.pdf.

Cossentino M. 2011. IEEE Foundation for Intelligent Physical Agents (FIPA) Design Process Documentation Template. Retrieved on January 9, 2013, from www.pa.icar.cnr.it/cossentino/fipa-dpdf-wg/docs/Process_Documentation_Template_20110615_Experimental.pdf

Craswell N. E. 2000. Methods for Distributed information retrieval. Unpublished Ph D thesis, Australian national University.

Daintith J. 2009. 'IT', A dictionary of physics, Oxford University press. Retrieved on 13 September, 2012 from en.wikipedia.org/wiki/information_technology.

Dale J. and DeRoure D. C. 1997. A Mobile Agent Architecture for Distributed Information Management. Proceedings of the International workshop on the virtual Multicomputer. Retrieved on April 10, 2010 from http://www.mmrg.esc.soton.ac.

Danny G. 2008. Agent-design pattern for building distributed service bus applications. Technical Report, Microsoft Corporation. Retrieved March 3, 2011 frommsdn.microsoft.com/en-us/library/dd334420.aspx

David R.A. 2004. Cross-Platform Generative Agent Migration: An Agent Factory Approach. Unpublished M Sc thesis, Department of Computer Science, Vrije Universiteit Amsterdam.

Dilyana S. and Petya G. 2002: Building Distributed Applications with Java Mobile Agent. Proceedings of Next Generation Network technologies International Workshop (NGNT, 2002), 103-109, Rousse, Bulgaria, 2002.

Dunne C.R. 2001. Using Mobile Agents for Network Resource Discovery in Peer-to-Peer Networks. In newsletter of ACM SIGecom Exchanges, 2(3):1-9

Ehrig M., Schmitz C., Staab C.,Taneand J. and Tempich C. 2002. Towards Evaluation Of Peer-To-Peer-Based Distributed Information Management Systems. Retrieved from http://www.aifb.uni-karlsruhe.de/WBS/cte/html/publications/pdf/ehrig02towards.pdf

El-Gamal Y., El-Gazzar K and Saeb M. 2007. A Comparative Performance Evaluation Model of Mobile Agent versus Remote Method Invocation for Information Retrieval. World Academy of Science, Engineering and Technology, 27, 286 – 291, 2007

Emerson, F. L, Patricia D.M., Jorge C. F and Flavio R. S. 2003. Implementing Mobile Agent Design Patterns in the JADE framework. Retrieved from http://jade.tilab.com/papers/EXP/Ferreira.pdf. (Date of last access 23rd December, 2013).

Farmer W.M., Guttman J.D and Swarup V. 1996. Security for mobile agents: Issues and Requirements. Proceedings of the National Information Systems Security Conference (NISSC'96).

Feyadat. 2008. Network Topologies. Retrieved 5th April, 2013, from www.csudh.edu/feyadat/../Networking...Network%20Topologies.ppt

Finin T. and Nicholas C. 2000. Software agents for information retrieval. Technical report, Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County.

Fischmeister S. 2004. Software Technologies, Mobile Code (2004). Technical report, Software Research Laboratory, University of Salzburg.

Fong P.W.L. 2003. Proof Linking: A Modular Verification Architecture for Mobile Code system. Unpublished PhD thesis, submitted to the School of Computing Science, Simon Fraser University.

Fortino G. and Russo W. 2003. High-level interoperability between java-based mobile agent systems. A report of the project 'Giovane Ricercatore 2003', University of Calabria.

Franklin S. and Graesser A. 1996. Is it an agent, or just a program? a taxonomy for autonomous agents. Proceedings of the third International workshop on agent theories, architectures and languages. Springer-Verlag, 1996.

Fuggetta A., Picco G., and Vigna G. 1998. Understanding code mobility. IEEE transactions on Software Engineering, 24(5): 342-361.

Gawali, R.D and Meshram, B.B. 2009. Agent-Based Autonomous Examination Systems. Proceedings of the Intelligent Agent and Multi-agent systems, 2009, (IAMA 2009) International Conference.

Geetha N. 2004. Database Management Systems for Information Mangement and Access. Proceedings of the 2nd international CALIBER-2004, New Delhi, 464- 472

Genco A. 2008. Mobile Agent: Principle of Operation and Application. Advances in management information..6. Retrieved on June 4, 2012 from www.lavoisier.fr/notice/gb334882.html

General Magic. 1995. Telescript language Reference. October,1995. Retrieved on January 9, 2011 from bitsavers.trailing-edge.com/pdf/generalMagic/Telescrip_Language_refrence_Oct95.pdf.

Gherbi T., Borne I. and Meslati D. 2009. MDE and mobile agent: Another reflection on the agent migration. Proceedins of the 11th International Conference on Computer Modelling and Simulation (UKSim 2009), Cambridge, United Kingdom.

Gilani N. 2012. Hybrid network topology. Retrieved on 13 March, 2013 from www.ehow.com/about_6495481_hybrid-networks.html.

Giovanni C. 2009. JADE Programming for Beginners. TILAB, S.P.A. Retrieved on June 4, 2012 from jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf.

Giovanni V. 2004. Mobile agents: Ten Reasons for failure. In proceedings of the 2004 IEEE International Conference on Mobile Data Management (MDM'04), USA. IEEE Computer Society Press, 298-299. www.cs.ucsb.edu/~vigna/publications/2004_vigna_MDM04.pdf

Gray R. S. 1997. Agent Tcl: a flexible and secure mobile-agent system. Unpublished PhD thesis in Computer Science at Dartmouth College, Hanover, New Hampshire.

Gray R., Cybenko G., Kotz D. and Rus D. 1996. Agent TCL. In Itinerant Agents: Explanations and Examples with CD-ROM, Manning Publishing.

Gray R.S., 1995. Agent TCL: a transportable agent system. In proceedings of the CIKM'95 Workshop on Intelligent Information Agent. J. Ousterhout TCL and the TK Toolkit Addison-Wesley, 1995.

Gray R.S., kotz D, Ronald A. P. Jr, Bartoon J., Chacon D., Gerken P, Hofmann M., Bradshaw J, Breedy M., Jeffers R., and Niranjan S. 2001. Mobile agent versus Client/server Performance: Scalability in an Information retrieval task. Lecture notes in Computer Science, Springer Verlag, 229-243

Grimstrup A., Robert R., Kotz D., Breedy M., Carvalho M., Cowin T., Chacon D, Barton J, Garrett C and Hofmann M. 2002. Toward interoperability of mobile agent systems. Proceedings of the sixth IEEE international Conference on Mobile Agent, Barcelona, Spain. Springer-Verlag, 106-120.

Gupta R. and Kansal G. 2011. A survey on comparative study of mobile agent platforms. *International journal of engineering, science and technology* (IJEST), 3(3): 1943 - 1948.

Halls D. A. 1997. Applying Mobile Code to Distributed Systems. Unpublished PhD dissertation Submitted to Computer Laboratory, University of Cambridge

Hiemstra D. 2000. Using Language Models for Information Retrieval. Unpublished Ph.D. thesis Centre for Telematics and Information Technology, Neitherlands.

Htoon H. and Thwin, M.M.T. 2008. Mobile Agent for Distributed Information Retrieval System. Proceedings of Electrical Engineering / Electronics, Computer, Telecommunication and Information technology Conference (ECTI-CON) 2008, 1: 169 – 172.

Huang Y. and Ravishankar C. 1996. URPC: A Toolkit for Prototyping RPC. *The computer Journal*, 1996, 39(6): 525 – 540.

Huhns M. N. and Singh M. P, (editors). 1997. Readings in Agents. morgan Kaufmann Publishers, 1997.

Ibharalu F. T., Sofoluwe A. B. and Akinwale A. T. 2011. A reliable protection architecture for mobile agents in open network systems. *International journal of computer applications* (0975-887), 17(7).

Ismail, L. and Hagimont D. 1998. A performance evaluation of the Mobile Agent Paradigm. *Proceedings of the 14th ACM SIGPLAN conference on OOP systems Languages and Applications. Denver, Colorado, USA, 306-313.*

Iyilade J. S. 2005. Development of multi-agent architecture for dynamic scheduling of jobs in grid computing systems. Unpublished M Sc thesis in Computer Science, Obafemi Awolowo University, Ile-Ife, Nigeria.

Iyilade, J. S., Aderounmu G. A. and Adigun M. O. 2005. An agent-based approach for finding a supervisor in an academic environment. *Proceedings of the 3rd*

international conference on parallel and distributed computing and systems (PDCS 2005), Phoenix, USA

Jansen W., Mell P., Karygiannis T. and Mark D. 1999. Technical Report, NIST Interim Report (IR), National Institute of Standards and Technology, Computer Security Division.

Jennings N.R. and Wooldridge M. 1998. Applications of Intelligent Agents. Queen Mary and Westfeild College, University of London.

Johansen D,Van Renesse R, and Schneider F. 1995. An introduction to the TACOMA Distributed system version 1.0. Technical Report, Institute of Mathematical and Physical Sciences, Department of Computer Science, Unviversity of Tromso, Norway. June 1995.

Jordi C., Benno O.J., Michel O.A., Joan B., and Frances, B.M.T. 2007. Abstract Software Migration Architecture Towards Agent Middleware Interoperability. *Proceedings of the International Multi-conference on Computer Science and Information Technology, 27-37. Retrieved March, 2011 from http://jipms.sourceforge.net.*

Joseph, A. D., DeLespinasse A.F., TauberJ.A., Gifford D.K., and Kaashoek M.F. 1995. Rover: A Toolkit for Mobile Information Access. *In proceeding of the Fifteeneth Symposium on Operating System Principles.*

Katrina M. H., Levine B.N., and Mammatha R. 2003. Mobile distributed Information Retrieval for Highly-Partitioned Networks. *Proceedings of the 11th IEEE international Conference on Network Protocols (ICNP'03).*

Kevin H. 2000. Distributed Computation with java Remote Method Invocation. Objective Viewpoint Retrieved on April 5 2012 from http://www.acm.org/crossroads/xrds6-5/ovp65.html.

Kotsis G. 1999. Performance management in parallel and distributed computing systems. Unpublished thesis institute fur Angewandte Informatic und Informations systeme, Abteilung Advanced Computer Engineering, Universitat Wien, Osterreich.

Kotz D. and Gray R.S. 1999. Mobile Agent and the Future of Internet. *Proceeding of the Workshop on Mobile Agents in the Context of Competition and Cooperation (MAC3) at Autonomous Agent '99, Seattle, Washington, USA, May 1999.*

Kretser O., Moffat A., Shimm T. and Zobel J. 1998. Methodologies for distributed information retrieval. *Proceedings of the 18th International Conference on Distributed Computing Systems ICDCS'98, Amsterdam. IEEE Computer Society Washington, DC, USA.*

Lange D. B. 1998. Mobile objects and mobile agents: the future of distributed computing? *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'98), 1998.*

Lange D.B and Oshima M. 1999. Seven Good Reasons for Mobile Agents. Communication of the ACM, .42(3), pp 88-89, March 1999.

Lesk M. 1996. Seven ages of information retrieval. Occasional paper of the International Federation of Library Associations and Institutions. Universal Dataflow and Telecommunications Core Program. Retrieved on 4th March, 2012 from http://www.ifla.org/udt/op/

Lieberman H. 2001. Letizia: An Agent that Assists Web Browsing. Media Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA.

Lister J. 2012. Hybrid network topologies advantages and disadvantages. Retrieved on 13 March, 2013 from www.ehow.com/list_7224727_hybrid-topology-advantages - disadvatages.htm

Lovrek I. and Sinkovic V. 2001. Performance evaluation of Mobile agent Network. Retrieved on 18 April, 2013 from www.fer.unizg.hr/_download/repository/KES2001lovsin.pdf

Mak E and Fukuda M. 2005. A development of resource/commander agents used in agent teamwork grid computing middleware. An inter-mediate report on faculty research internship, funded by National Science Foundation.

Maninda K. Computer network topologies. Retrieved February 15, 2013 from www.computer_network/topologies/maninda.ppt

Manning C.D., Raghavan P. and Schutze H. 2009. Introduction to Information Retrieval. Online edition (c) 2009 Cambridge University Preess.

Margaret R. 2005. IT (Information Technology). Retrieved September 18, 2012 from www.searchdatacenter.techtarget.com/definition/IT

Michael B. and Takanori U. 1997. Comparison of autonomous mobile agent technologies. Technical report by APM Limited, United Kingdom.

Milojicic D, Breugst M., Busse I., Campbell J., Covaci S, friedman B, Kosaka K., Lange D, Ono K, Oshima M, Tham C, Virdhagriswaran S. and White J. 1998. MASIF: The OMG Mobile Agent System Interoperability Facility. Personal technologies, 2(2):17-129, Springer-Verlag.

Mitrovic D., Ivanovic M., Budimac Z. and Vidakovic M. 2011. An overview of agent mobility in heterogenous environments. *Proceedings of the workshop on applications of software agents: 52-58, 2011.*

Mobaideen W.A. 2003. Performance Evaluation of Mobile Agents Paradigm for Wireless Networks. Technical report UBLCS-2003-04. Department of Computer Science University of Bologna, Italy Available at http://www.cs.unibo.it/pub/TR/UBLCS/2003/2003-04.ps.gz.

Neeran M. K. and Anand R. T. 1998. Design issues in Mobile Agent Programming System. IEEE Concurrency Journal, 6: 52-61. Retrieved on January 07, 2013 from http://www.cs.umn.edu/Ajant.

Nguyen H. V. 2004. Mobile Agent Application in Computer network. Technical report submitted to DSV Stockholm University.

Nikolaos V. K., Vassili L. and Alexander T . 2004. Mobile Agent Assisted Value-Adding Communities for Mass Customisation. *International Journal of simulation, 7 (7): 56-65. (ISSN 1473-804x online).*

Nikos M., William J.B. and Kevia A. M. 2003. Mobile Agent for Routing, Topology Discovery'and Automatic Network Reconfiguration in Ad-Hoc Networks. School of computing, Napier University, scotland U.K. *Procedings of the 10th IEEE International Conference and Workshop on Engineering of Computer-Based Systems, April 2003, 200 – 206.*

Nitin J., Kamlesh, and Neeraj S. 2011. Security issues in mobile agent paradigm. *International journal of computer science and management studies, 11(1): 43-46.*

Nwana H. S. 1996. Software Agents: An Overview. Knowledge Engineering Review, 11(3): 205-244. Cambridge University Press.

O'Brien P.D. and Nicol R.C. 1998. FIPA- towards a standard for software agents. *BT Technology Journal, 16(3). 51-59. www.unalmed.edu.co/~dovalle/FIPA.pdf*

Oak M. 2011. Types of Network Topologies. Retrieved on 06 February, 2013 from www.buzzle.com/articles/types-of-network-topologies.html..

Olubusoye, O. E, Olaomi J.O. and Shittu O.I. 2001. Statistics for engineering, Physical and Biological Sciences. A divine touch publication, Nigeria. ISBN: 978-35606-7-0.

Outtagarts A. 2009. Mobile agent-based applications: A Survey. *International Journal of Computer Science and Network Security, 331-339.*

Oyatokun B.O. 2004. Remote Method Invocation and Mobile Agent: A comparative Analysis. Unpublished MSC thesis, Department of Computer Science, University of Ibadan, Nigeria.

Pears S. 2005. Using mobility and exception handling to achieve mobile agents that survive server crash failures. Unpublished PhD thesis submitted to the Department of Computer Science, University of Durham.

Peters C. 2012. Networking 101: Concepts and definitions. www.techsoup.org/support/articles-and-how-tos/networking-101-concepts-and-definitions.

Picco G.P. 2005. Understanding code mobility. Technical report, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy. [date of last access: 18 June, 2013]

Picco, G. P., Roman, G. and McCann, P. J. 2001. Reasoning about code mobility with mobile unity. ACM Transaction on Software Engineering and Methodology (TOSEM),

10(3), 338 – 395, 2001. www.disi.unitn.it/~picco/papers/tosem97.pdf [date of last access: 15 January, 2014]

Pinsdorf U. and Roth V. 2002. Mobile Agent Interoperability Patterns and Practice. Retrieved on April 9, 2012from http://jade.tilab.com/papers/EXP/pinsdorf.pdf.

Pleisch S. 1999. State of the art of mobile agent computing- security, fault tolerance and transaction support. Technical report, IBM Research, Zurich Research Laboratory, Switzerland.

Priya, B.G; Suba, S.; Bensal, T.; and Boominathan, P. 2009. Enhanced Communication Scheme for Mobile Agent. *Proceedings of the Intelligent Agent and Multi-agent systems, 2009, (IAMA 2009) International Conference.*

Proctor, K. S. 2011. Optimizing and assessing information technology improving business project Execution, John Wiley and sons. ISBN 978-118-10263-3. Retrieved on 10 October, 2012 from en.wikipedia.org/wiki/information_technology.

Rahul J. and Scrdhar I. 2001. Performance Evaluation of mobile Agent for E-Commerce Applications. Kanwal School of Information technology, Indian Institute of Technology, Bombay, Powai, Mumbai.

Rampur S. 2011. Computer networking basics. Retrieved on 13th March, 2013 from www.buzzle.com/articles/computer-networking-basiscs.html.

Roberto S. S. F. 2001. The Mobile Agents Paradigm. A research paper in the department of Information and Computer Science, University of Califonia, Irvine.

Rouse M. 2010. Network topology. Networking and communication glossary. Retrieved on 13 March, 2013, from http://www.whatis.techtarget.com/definition/network-topology

Salton G. and McGill, M.J. 1983. Introduction to Modern Information Retrieval. McGraw-Hill

Seng W. L. 1999. Mobile Agent Technology for Enterprise Distributed Applications: An overview and an Architectural Perspective. CRC for Distributed System Technology, Monash University, Cualfield Campus, Caulfield East, Victoria 3145, Australia.

Shinder D. L 2001. Computer Networking Essentials for Educational Institutions (Cisco System). cisco press, Indianapolis, USA.

Shoham and Layton-Brown. 2009. Multiagent system: Algorithmic, game theoretic and logical foundation. Cambridge University Press, 2009.

Shoham, Y. 1993. Agent oriented Programming, Artificial Intelligence, 60(1): 51-92, in Pears S. 2005. Using mobility and exception handling to achieve mobile agents that survive server crash failures. Unpublished PhD thesis submitted to the Department of Computer Science, University of Durham.

Silberschartz A., Galvin P. and Gagne G. 2009. Operating System Concepts. 8th edition, John Wiley & sons, Inc

Silberschartz A., Korth, H. F. and Sudarhan S. 1997. Database System. Concepts. Third edition. McGraw – Hill (Series in Computer Science)Companies, inc.

Silva A. and Delgado J. 1998. The agent pattern for mobile agent system. *Proceedings of the 3rd European Conference on pattern language of programming and computing, EuroPLPO'98.*

Singh Y., Gulati K. and Niranjan S. 2012. Dimensions and Issues of Mobile Agent Technology. International Journal of Artificial Intelligence and Applications (IJAIA), 3(5), 51-61.

Spicer K. L. 2000. A successful example of a layered-architecture based embedded development with Ada 83 for Standard-Missile Control. Retrieved February 11, 2014 from www.sigada.org/ada_letters/dec2000/spicer-paper.pdf

Spiegel M. R and Stephens L. J. 1999. Theory and problems of Statistics. Third edition, Schaum's Outline Series, McGRAW-HILL, New York .

Sridhar I. and Vikram J. 2001. Designing Distributed Applications using Mobile Agent. *Procedings of International Conference on High Performance Computing, 2001. Hyderabad, India*

Stoian G and Popirlan C.I. 2010. A proposal for an enhanced mobile agent architecture. *Annals of the University of Craiova, Mathematics and Computer Science Series, 71-79.*

Stuck, B.W and Arthurs, E. 1985. A Computer and Communication Network Performance Analysis Primer. Prentice-Hall, Inc.

Sullins J. 2012. Information technology and moral values. The Stanford Encyclopaedia of philosophy (fall 2012 edition), Edward N. Zalta (ed). Retrieved on 13 September, 2012 from http://plato.stanford.edu/achives/fall2012/entries/it-moral-values/

Sycara K. P. 1998. Multiagent Systems. A publication of the American Association for Artificial Intelligence, 445 Burgess Drive, Menlo Park, California. 78-92.

Syed A., John D. and Pavana, Y. 2000. A survey of Mobile Agent Systems. Student Report, Department of Computer Science and Engineering, University of California San Diego. (Date of last access: 03 July, 2013 from cseweb.ucsd.edu/classes/sp00/cse221/reports/dat-yal-and.pdf).

Tanenbaum , A. S. and Steen M. V. 2007. Distributed systems: Principles and Paradigms. Second edition, Pearson Educational, Inc. Pearson Prentice Hall.

Tanenbaum, A. S. 2003. Computer Networks. Fourth edition, Peason Education, Inc. Pearson Prentice Hall.

Tudor M., Bogdan D., Mihaela D. and Ioan S. 2004. A Framework of Reusable Structures for Mobile agent Development. *Proceedings of the 8th IEEE international Conference on Intelligent Engineering Systems (INES '04), Cluj-Napoca, 2004.*

van Rijsbergen C. J. 1979. Information retrieval. Department of Computing Science, University of Glasgow, second edition.

Venners B. 1997. Under the hood: The architecture of aglets. Java-World, January 1997. http://java-world/aglets.source/ge.net

Vitek J. 1997. New Paradigms for Distributed Programming. *Proceedings of the European Research Seminar in Advanced Distributed Systems, (ERSADS'97), Zinal (Valais, Switzerland) March 17-21, 1997.*

Wallin A. 2004. Secure auction for mobile agents. Technical report of VTT Technical Research Centre of Finland.

Wayne A. J. 2000. Countermeasures for mobile agent security. *Journal of computer communications*, Elsevier Science Publishers B.V. Amsterdam, 23 (17): 1667 – 1676.

Wenjuan W., Tong L., Weidong Z. and Weihui D. 2009. Mobile agent system for supply chain management. *Proceedings of the second symposium of International Computer Science and Computational technology (ISCSCT'09), Huangshan, P.R China, 525-528.*

William E. W. 2007. Design and Evaluation of a Pipelined Distributed Information Retrieval Architecture. A M.Eng thesis, department of Computer Science and Software Engineering, university of Melbourne.

WIN133. 2009. The big picture…which makes more sense now. Retrieved on 20 *September*, 2012, from www.microsoft.com.windows/default.mspx .

Winkelman R. 1997. An Educator's Guide to School Networks. Florida Centre for Instructional Technology, University of South Florida. Retrieved on 5th April, 2013 from fcit.usf.edu/network/chap5/chap5.htm.

Wooldridge M and Jennings N.R. 1995. Intelligent agents: theory and practice. *The knowledge engineering review, 10(2): 115 -152.*

Yepes A. J. J. 2009. Ontology Refinement for Improved Information Retrieval in the Biomedical Domain. Unpublished Ph D thesis, Dpto. De Lenguajes Y Sistemas Informaticos, Universitat Jaume I.

Zeghache L. Badache N. and Elmaouhab A. 2002. An architectural model for mobile agent system interoperability. H. Labiod and M Badra (eds). *New Technologies, Mobility and Security, 555-566, 2007 Springer.*