APPLICATION OF MOBILE AGENTS IN NETWORK MANAGEMENT

By

HAMIDU ISAH

DEPARTMENT OF MATHEMATICS, FACULTY OF SCIENCE,

AHMADU BELLO UNIVERSITY, ZARIA

JULY, 2014

APPLICATION OF MOBILE AGENTS IN NETWORK MANAGEMENT

BY

Hamidu ISAH

MSc/SCIE/5045/2010-2011

A THESIS SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES, AHMADU BELLO UNIVERSITY, ZARIA

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF A MASTER DEGREE IN COMPUTER SCIENCE.

DEPARTMENT OF MATHEMATICS,
FACULTY OF SCIENCE
AHMADU BELLO UNIVERSITY, ZARIA
NIGERIA

JULY, 2014

DECLARATION

I declare that the work in this thesis entitled **Application of Mobile Agents in Network Management** has been performed by me in the Department of Mathematics, Ahmadu Bello University, Zaria. The information derived from the literature has been duly acknowledged in the text and a list of references provided. No part of this thesis was previously presented for another degree or diploma at this or any other Institution.

Name:_____ Signature: _____ Date:_____

CERTIFICATION

This thesis entitled **APPLICATION OF MOBILE AGENTS IN NETWORK MANAGEMENT** by **Hamidu ISAH** meets the regulations governing the award of the degree of Master of Science (M.Sc.) Computer Science of Ahmadu Bello University, and is approved for its contribution to knowledge and literary presentation.

Dr. S. E. Abdullahi, Chairman, Supervisory Committee	Signature	Date
Dr. M. B. Hammawa, Member, Supervisory Committee	Signature	Date
Dr. B. Sani, Head of Department	Signature	Date
Professor A. A. Joshua Dean, School of Postgraduate Studies	Signature	Date

ACKNOWLEDGEMNT

First and Foremost, my gratitude goes to God for creating and sustaining me throughout my struggles.

I would like to express my sincere gratitude to my supervisors Dr. S. E. Abdullahi and Dr. M. B. Hammawa for the continuous support of my M.Sc. study and research, for their patience, motivation, enthusiasm, and immense knowledge. Their guidance helped me at all the time of research and writing of this thesis. Also acknowledge are the entire academic and non-academic staff members of the Department of Mathematics of Ahmadu Bello University, Zaria for their various levels of contributions towards the actualization of this dream, most especially the departmental coordinator, Postgraduate programs, Dr. A. Yahaya, and Head of Mathematics department, Dr. B. Sani.

I must also acknowledge the support and team spirit exhibited by each and every member of my M.Sc. class. Indeed, we did it all together.

I am indebted to my parents late Alhaji Isah Hussaini and Hajiya Ruqayya Alhassan, for taking care of me and supporting me spiritually throughout my life and to my wife Haj. Maryam who through her support, understanding and patience, I was able to overcome what otherwise stood as an insurmountable huddle.

ABSTRACT

According to Case *et al.,* (1990) and Pinheiro *et al.,* (1999), the present day ways of managing network systems as represented by Simple Network Management Protocol (SNMP) are based on the centralized client-server approach. In this approach, the central server (manager) retrieves raw management data from all the clients (network elements) being managed. This raw data is centrally collated and processed by the server for every management cycle. This is good enough for small, reliable and fast networks that are particularly localized in nature. In present-day situation where networks extend across cities, regions, countries, and continents; as well as in mobile applications where connected links are very slow and expensive, this may lead to inefficiency in terms of bandwidth utilization, resource wastage, processing bottleneck on the server, and other problems that could be caused by communication failures between the server and the clients being managed.

The objective of this work is to demonstrate that a more efficient approach to Network Management could be achieved by implementing a distributed system through the use of Mobile Agents (MA). Typically, Biesoczad *et al.,* (1998) and Gavalas *et al.,* (1999) were among the early researchers who investigated the concept and possibilities of using Mobile Agents in the management of network components. Using this concept, a central manager generates software agents and dispatches them with predefined itinerary and specific network management tasks; visiting each network element to be managed; retrieving, analyzing, and only returning computed results back to the central manager from where the agents originated. This way, the central manager does not need to pull in huge volume of raw management data from the many dispersed network elements being managed for central processing. This conserves expensive network

bandwidth, relieves the central server of the mundane task of processing raw management data that could result in processing bottleneck, and ensures that only computed results get transmitted to the central server for display to the network administrator.

JAVA Agent Development Environment (JADE) platform was used to develop a Mobile Agent based network management system. Issues covered by this research include Status monitoring of network elements, simple network management protocol table filtering, and Global filtering on a UNICEF (United Nations Children's Fund) network environment that spanned across five cities (Abuja, Bauchi, Enugu, Kaduna, and Lagos) in Nigeria. MA's were generated and dispatched by a central server in Abuja, visiting network elements to be managed in Bauchi, Enugu, and Lagos. In each case, volume of data in bytes transferred in both directions as well as transit time in seconds taken by this data were noted. Results were compared with those obtained from the same experiments conducted using SNMP approach. Final analysis shows that while performance was slower using MA as compared to SNMP approach, bandwidth utilization was far much better with MA. Even the slower performance exhibited by MA was mainly due to the JADE platform used during the implementation stage. Other studies have shown that better throughput is attainable using other development platforms. Also covered by this work are some security issues relating to mobile agents in a networked environment.

TABLE OF CONTENTS

DECLARATION ii

CERTIFICATION iii

ACKNOWLEDGEMNT iv

ABSTRACT v

TABLE OF CONTENTS vii

LIST OF FIGURES AND PLATES xi

LIST OF TABLES xii

ABBREVIATIONS xiv

DEFINITIONS xvi

CHAPTER 1 1

GENERAL INTRODUCTION 1

1.1 BACKGROUND OF THE STUDY 1

1.2 STATEMENT OF THE PROBLEM 2

1.3 AIMS AND OBJECTIVES OF THE RESEARCH 3

1.4 RESEARCH QUESTIONS AND HYPOTHESIS 4

1.5 ORGANIZATION OF THE THESIS 5

1.6 CONTRIBUTION TO KNOWLEDGE 6

CHAPTER 2 7

LITERATURE REVIEW 7

2.1 OVERVIEW OF NETWORK MANAGEMENT 7

2.1.1 Simple Network Management Protocol 8

2.1.2 Management by Delegation (MbD) 9

2.1.3 Remote Monitoring (RMON) 10

2.1.4 Mobile Agents (MA) 11

2.2 JADE - JAVA AGENT DEVELOPMENT FRAMEWORK 11

CHAPTER 3 14

MATERIALS AND METHODS 14

3.1 SYSTEM ARCHITECTURE 14

3.1.1 Network Manager 14

3.1.2 Network Element 16

3.2 MANAGER AGENT 16

3.2.1 Communication of components 17

3.2.1.1 Receiving a Register Message 17

3.2.2 Manager Agent Behaviours 19

3.2.2.2 Implementing Agent Bebaviour Classes 20

3.3 DAEMON AGENT (NEAGENT) 21

3.4 HEALTH FUNCTION (PERFORMANCE INDICATORS) 23

3.4.1 Interface Utilization 24

3.4.2 Error Rate and Accuracy 25

3.4.3 Internet Protocol (IP) Output Datagrams Discard Rate 26

3.5 MESSAGE DEFINITION 27

3.6 COMMUNICATION OF MANAGER AGENT WITH DAEMON AGENT 27

3.7 MOBILE AGENT (MA) 29

3.7.1 SNMP table filtering 30

3.7.2 Global Filtering 34

CHAPTER 4 37

RESULTS AND DISCUSSION 37

4.1 SECURITY PROBLEM 37

4.1.1 Protecting NE from malicious MA 38

4.1.2 Protecting the important data in MA 40

4.1.3 Secure the Communication Channel 42

4.2 SCALABILITY 43

CHAPTER 5 46

SUMMARY, CONCLUSION AND RECOMMENDATION 46

5.1 SUMMARY 46

5.2 CONCLUSION 47

5.3 RECOMMENDATION AND FUTURE WORK 47

5.4 REFERENCES.....49

LIST OF FIGURES AND PLATES

FIGURE 1: THE SNMP ARCHITECTURE 8

FIGURE 2: SOFTWARE ARCHITECTURE OF ONE JADE AGENT PLATFORM 13

FIGURE 3: ARCHITECTURE OF MOBILE AGENTS 15

FIGURE 4: CLASS COMPONENTS OF MANAGER AGENT 16

FIGURE 5: REGISTRATION EXAMPLE 18

FIGURE 6: BEHAVIOURS' SCHEDULING OF THE MANAGER AGENT 20

FIGURE 7: CODE SEGMENT OF MANAGERNEINMSGBEHAVIOUR CLASS 21

FIGURE 8: COMMUNICATION BETWEEN MANAGER AND DAEMON AGENT 23

FIGURE 9: MAINGUI OF MANAGER AGENT 28

FIGURE 10: MOBILE AGENT GENERATOR 31

FIGURE 11: SNMP TABLE FILTERING FLOW DIAGRAM 33

FIGURE 12: RESULTS TABLE OF SNMP TABLE FILTERING MA 34

FIGURE 13: EXAMPLE OF GLOBAL FILTERING MA GENERATOR GUI 35

FIGURE 14: RESULTS TABLE OF GLOBAL FILTERING MA 36

FIGURE 15: AUTHENTICATION OF MA 40

FIGURE 16: ENCRYTION OF DATA USING JCE API 41

FIGURE 17: DECRYPTION OF DATA USING JCE API 42

LIST OF TABLES

TABLE 1: MIB II VARIABLES FOR COMPUTING INTERFACE UTILIZATION.....24

TABLE 2: MIB II VARIABLES FOR COMPUTING INTERFACE ERROR RATE AND ACCURACY

.....26

TABLE 3: DEFINITION *OF MIBTREEOBJECTGET* ACL MESSAGE 27

TABLE 4: RESPONSE TIME OF SNMP AND MA 44

TABLE 5: COMMUNICATION OVERHEAD OF SNMP AND DAEMON AGENT 45

ABBREVIATIONS

ACC:	Agent Communication Channel
ACL:	Agent Communication Language
AMS:	Agent Management System
API:	Application Programming Interface
ASN:	Abstract Syntax Notation
CA:	Certificate Authority
CMIP:	Common Management Information Protocol
CORBA:	Common Object Request Broker Architecture
CPU:	Central Processing Unit
DAI:	Distributed Artificial Intelligence
DF:	Directory Facilitator
FIPA:	Foundation for Intelligent Physical Agents
GUI :	Graphical User Interface
HF:	Health Function
HTTP:	Hyper Text Transfer Protocol
ID:	Identifier
IEEE	Institute of Electrical and Electronics Engineers
IFIP	International Federation for Information Processing
IIOP:	Internet Inter-ORB Protocol
IMAP:	Internet Message Access Protocol
JADE:	*JAVA* Agent Development Environment
JCE:	JAVA Cryptography Environment
JVM:	Java Virtual Machine

LAN:	Local Area Network
MA:	Mobile Agent
MAG:	Mobile Agents Generator
MASIF:	Mobile Agent System Interoperability Facility
MbD:	Management by Delegation
mbps:	Megabits per second
MIB:	Management Information Base
NE:	Network Element
NM:	Network Manager
OID:	Object Identifiers
OMG:	Object Management Group
ORB:	Object Request Broker
OS:	Operating Systems
PKI:	Public Key Infrastructure
RMI:	Remote Method Invocation
RMON	Remote Monitoring
SDK:	System Development Kit
SNMP:	Simple Network Management Protocol
SNMPD:	Simple Network Management Protocol Daemon
SSL:	Secure Socket Layer
TCP/IP:	Transmission Control Protocol/Internet Protocol
UDP:	User Datagram Protocol
WAN:	Wide Area Network

DEFINITIONS

T-1
Stands for any data circuit that runs at the original 1.544 mbps line rate.

JAVA
Java is a programming language originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java is a general-purpose, concurrent, class-based, object-oriented language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere"

FIPA: Foundation for Intelligent Physical Agents
FIPA is a body for developing and setting computer software standards for heterogeneous and interacting agents and agent-based systems. FIPA was founded in 1996 as a non-profit organization with the ambitious goal of defining a full set of standards for both implementing systems within which agents could execute (agent platforms) and specifying how agents themselves should communicate and interact.

CORBA
The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together (that is, it supports multiple platforms).

Mobile Agents
A mobile agent is a composition of computer software and data which is able to migrate (move) from one computer to another autonomously and continue its execution on the

destination computer. Mobile Agents have the feature of *autonomy*, *social ability*, *learning*, and most importantly, *mobility*.

OMG: Object Management Group

An international organization founded in 1989 to endorse technologies as open standards for object-oriented applications. The OMG specifies the Object Management Architecture (OMA), a definition of a standard object model for distributed environments, more commonly known as CORBA.

CHAPTER 1

GENERAL INTRODUCTION

1.1 BACKGROUND OF THE STUDY

According to Case *et al*., (1990) and Pinheiro *et al.,* (1999), the present day ways of managing network systems as represented by Simple Network Management Protocol (SNMP) are based on the client-server architecture, where a central station, called the manager collects and analyses raw data retrieved from physically distributed network elements. In those systems, management data are stored in a standard structure maintained on the network elements (NE) to be managed, such as Management Information Base (MIB) Objects Tree in SNMP. Under this setup, there is always a routine/task at each network element, such as SNMPD (Simple Network Management Protocol Daemon) running on *Linux*, which periodically fetches and returns raw management data in response to requests from the network manager.

Network management system based on Client/Server paradigm like this normally requires transferring large amount of management data between the manager and managed network elements. The large amount of data not only requires considerable bandwidth, but also can cause a processing bottleneck at the manager. This is particularly disturbing considering the fact that the bulk of the transmitted data may end up being irrelevant and hence, discarded by the manager on arrival. As current networks grow larger and more complicated, the problem becomes more severe (Case *et al*., 1990).

1.2 STATEMENT OF THE PROBLEM

As highlighted in section 1.1, network management based on Client/Server approach poses a number of challenges due to its centralized nature. Some of these challenges as extensively discussed by Bivens *et al.,* (1999) and Bohoris *et al.,* (2000) are:

a. **Bandwidth intensive:** For larger networks, management data being transferred between the manager and the remotely located network objects being managed becomes huge and consumes a lot of communication bandwidth. Bandwidth today stands out as one of the most expensive resource in a typical networked environment. This is especially the case when the network is geographically dispersed and leased communication channels have to be utilized.

b. **Processing bottleneck at Network Manager:** As the data is raw, the network manager spends a lot of time analyzing, hence, sometimes causing processing to be slower than desired. This impact negatively on the entire management domain where similar processing needs to be performed by the same manager for other managed network elements. It becomes more serious when results are needed on real-time basis and the manager is not forthcoming.

c. **Resource Wastage:** Worse again, in most cases, substantial part of the data is not needed and therefore discarded by the network manager after processing, resulting in overall resource wastage (bandwidth, processing time, storage, memory, and so forth).

d. **Effect of Communication failure:** As is the case with all networked environments, sometimes, communication breakdown does occur between the manager and some or all the network objects being managed. This impact negatively on the ability of the network manager to collate management data for processing.

Together, these challenges form the overall motivation for this research work and at the same time, help in defining the objectives set to be achieved by the research as detailed in section 1.3 below.

1.3 AIMS AND OBJECTIVES OF THE RESEARCH

Overall, this project work tries to solve the problems listed in section 1.2 by applying a distributed management mechanism which overcomes the limitations posed by the traditional centralized Client/Server architecture as presented by SNMP. The work demonstrates that mobile agents could be used to perform network management functions more efficiently in terms of bandwidth utilization as compared to the traditional SNMP approach that requires a lot of communication bandwidth which effectively reduces its utilization span in terms of network size. According to Bivens *et al.,(*1999); Gavalas *et al.,* (2000); Paliafito and Tomarchio (2000); Bohoris *et al.,* (2000); and Griffin *et al.,* (2001); Mobile Agents could be used as a decentralized approach to network management which greatly reduces the workload at the management server location by otherwise delegating network management responsibilities to the network elements being managed. To be able to achieve these desired objectives, the following approaches to network management were extensively reviewed:

a. Remote Monitoring (RMON);
b. Management by Delegation (MbD); and
c. Mobile Agents.

Each of these three (3) alternatives mentioned above introduces some degree of decentralization. However, the third solution, the use of Mobile Agents (MA)

technology to distribute and delegate management tasks has been used to implement the following aspects of network management:

a. Network elements' status monitoring,
b. SNMP table filtering, and
c. Global filtering.

1.4 RESEARCH QUESTIONS AND HYPOTHESIS

Distinguished scholars such as Bivens *et al.,(*1999); Gavalas *et al.,* (2000); Paliafito and Tomarchio (2000); Bohoris *et al.,* (2000); Griffin *et al.,* (2001) have extensively investigated the use of Mobile Agents (MA) technology to distribute and delegate management tasks in the past.

Additionally, mobile Agent frameworks have already attracted a lot of attention in recent years. A lot of research is currently being carried out to assess the applicability of agent technology to network management and control environment. It has been argued by Gavalas *et al.,* (2000); Paliafito and Tomarchio (2000); Rubinstein *et al.*, (2002) that MAs have some superior features over SNMP, MbD and RMON. There is a general agreement that MA can be used to alleviate the network manager workload and reduce the bandwidth usage by delegation of authority from the manager to MA. Those advancing this argument believe that MA is more flexible and could be instantly customized by user's requirement and launched from the manager. It can visit each network element according to the itinerary table, computing and compressing the management data locally, only returning the result to the network manager. By moving a portion of the "intelligence" to the nodes where data are resident, many of the management decisions could be taken locally, thus avoiding the transfer of large

amounts of data from the remote nodes to the central manager (Paliafito and Tomarchio 2000).

Applicability of Mobile Agents in Network management was implemented using JAVA Agent Development Environment (JADE) platform. Applications implemented in this system are:

a. Network element's status monitoring,
b. SNMP table filtering, and
c. Global filtering.

This thesis described the architecture, design and implementation details of these applications. Common security problems of mobile agent were also investigated. Security is a crucial problem to the feasibility of mobile agent, especially to the network management domain which has great security requirements. Finally, the performance of the system was tested and potential problems analyzed.

1.5 ORGANIZATION OF THE THESIS

An introduction to some background knowledge related to this research is given in chapter 2. This includes brief overview of JADE, SNMP; a brief description of Remote Monitoring (RMON), Management by delegation (MbD) and Mobile Agents (MA).

The overall overview of the system's architecture is given in Chapter 3. Also discussed in this chapter are the detailed design and implementation issues.

Results, discussions, as well as identified related problems, such as securing the whole system and scalability test are explained in Chapter 4.

Chapter 5 summarized, concluded and highlighted possible future work on this research area.

1.6 CONTRIBUTION TO KNOWLEDGE

The thesis contributes to the area of network management by demonstrating that an entirely new emerging concept of Mobile Agents technology could be used to delegate and distribute network management responsibilities across network objects; as opposed to the traditional Clients/Server central approach which though simple, poses a number of operational challenges as regards to bandwidth utilization, processing bottleneck at manager location, and difficulty in handling single point of failure. Although, an entirely new area in modern computing, a number of studies have been undertaken by others in the area of Mobile Agents application in network management. Bieszczad *et al.,* (1998) described theoretical views on applications of MA for network management. They discussed potential uses of MA's in network management and defined software agents and a navigation model that determines agents' mobility. They further listed potential advantages and disadvantages MA's. Gavalas *et al.,* (1999) presented the application of MA in bulk transfers of network monitoring data, data aggregation and acquiring atomic SNMP table views. They analyzed the usage of mobile agents in network management with regards to the bandwidth utilization. Pinheiro *et al.,* (1999) described a conceptual model which collects management related data across a changing set of networked components and periodically computes aggregated statistics using MA. This thesis draws ideas from above concepts and applies them differently in the area of MA application to network management; with particular interest in Network elements' status monitoring, SNMP table filtering, and global filtering.

CHAPTER 2

LITERATURE REVIEW

An introduction to some background knowledge related to this research is provided in this chapter. It begins with a general overview of the present system of network management, namely, Simple Network Management Protocol (SNMP). A brief description of Remote Monitoring (RMON); Management by delegation (MbD); and Mobile Agents (MA) follows. Finally, a detailed overview of the agents software development platform, JADE (Java Agents Development Environment) used to implement the proposed system is given.

2.1 OVERVIEW OF NETWORK MANAGEMENT

According to Bohoris *et al.,* (2000), network management has been the subject of intense research over the last decade, with the relevant progress being twofold: on one hand, architectures and algorithms for solving management problems have been devised; on the other hand, different management technologies have been proposed and standardized.

Currently, the network management systems adopt a centralized paradigm, represented by SNMP. Because of its simplicity, it has been adopted worldwide as a standard protocol for managing network objects such as network routers, Local Area Network (LAN) switches, bridges, hubs, workstations, servers, printers, and so forth. Although SNMP is a standard protocol, it has many limitations and inefficiencies. Most of these problems are rooted in SNMP's centralized architecture. The rational approach to overcome most of these limitations is to distribute the network management operations. There are several ways which have been under investigation, namely, Management by

Delegation (MbD), Remote Monitoring (RMON), and the use of Mobile Agents (MA). The following will give brief overviews of those approaches.

2.1.1 Simple Network Management Protocol

The Simple Network Management Protocol (SNMP) was developed in the late 1980's to Provide network operators with a simple tool they could use to manage their networks. It has gained widespread acceptance since 1993, making it a standard to manage TCP/IP Networks, including individual network devices, and devices in aggregate. The more sophisticated Common Management Information Protocol (CMIP) never replaced SNMP. CMIP has only been deployed in the telecommunication networks and not IP networks. Case *et al.,* (1990) used Figure 1 below to depict the architecture of SNMP. It defines a centralized client/server relationship.

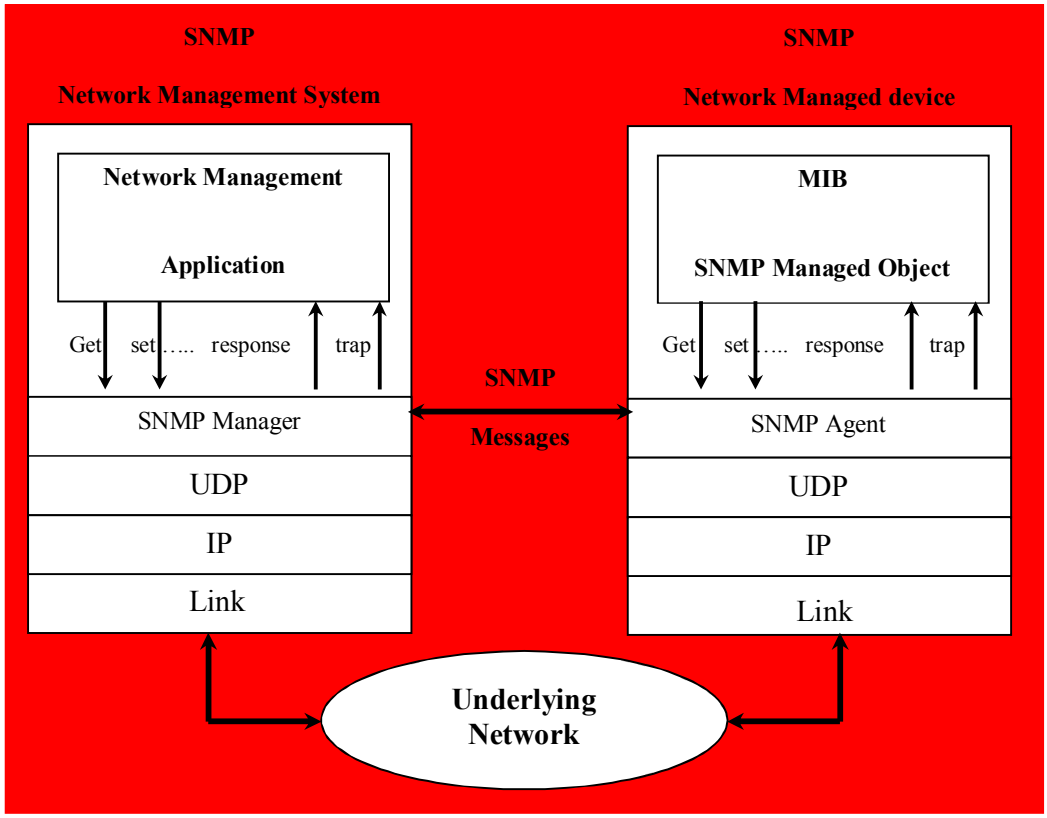


Figure 1: The SNMP Architecture

The SNMP Manager makes the connections to a SNMP Agent which executes on a remote network device, and serves information to the manager regarding the device's

underlying Network status. The database, controlled by SNMP agent, is referred to as the SNMP Management Information Base (MIB), and is a standard set of statistical and control values. Directives, issued by the network manager to a SNMP agent, consist of the identifiers of SNMP variables (referred to as MIB object identifiers or MIB variables) along with instructions to either GET the value for an identifier, or SET the identifier to a new value.

Through the use of private MIB variables, SNMP agents can be tailored for a countless of specific devices, such as network bridges, gateways, and routers. According to Case *et al.,* (1990), the definitions of MIB variables supported by a particular agent are incorporated in descriptor files, written in Abstract Syntax Notation (ASN.1) format.

The popularity of SNMP is due to a number of features. It could cover a large range of devices to be managed, and it is a very flexible and extensible management protocol. It is also proved to be good under poor network conditions. However, SNMP is not a particularly efficient protocol. Bandwidth is wasted with needless information, such as the SNMP version (transmitted in every SNMP message) and multiple length and data descriptors scattered throughout each message. Those shortcomings did not stop the widespread use of it.

2.1.2 Management by Delegation *(MbD)*

The Management by Delegation model was proposed in 1991 to address the difficulty of network management using centralized systems like SNMP. According to Picco (1998), the key idea behind the MbD approach is to delegate management functions to remote devices in order to reduce communication costs, to avoid a single point of failure, and to increase the scalability of management applications.

The management architecture of *MbD* still includes a management protocol and agents, yet an elastic process run-time support is assumed on each device. Instead of exchanging simple messages, the management station can specify a task by packing into a program a set of commands to agents and send it to the devices involved, thus delegating to them the actual execution of the task. This execution is completely asynchronous, enabling the management station to perform other tasks in the meantime and introducing a higher degree of parallelism in the management architecture. Moreover, since the code fragments (which the authors claim can be written in any language) are not statically bound to devices, they can be changed and re-sent by the management station at any time. This enables more flexibility, because the management station can customize and enhance dynamically the services provided by the agents on the devices (Picco, 1998).

2.1.3 Remote Monitoring (RMON)

RMON assumes the availability of network monitoring devices called monitors or probes. By monitoring packet traffic and analyzing the headers, probes provide information about links, connections among stations, traffic patterns, and status of network nodes. Hence, RMON can be regarded as traffic-oriented approach because the status of the network is determined by direct inspection of the packets flowing in it, rather than inspection of the status of each device. According to Baldi *et al.,* (1997), a probe in RMON can detect failures, misbehaviors, and identify complex relevant events even when not in contact with the management station, which is likely to happen when the network is overloaded or in critical conditions. In addition, the agent on the probe can also do periodic checking and semantic compression, which further increases decentralization.

2.1.4 Mobile Agents (MA)

The emergence of mobile agent frameworks has led many researchers to examine their applicability to network management and control environments. Gavalas *et al.,* (2000) suggested that mobile agents can provide better solutions, at least, to performance and fault management problems, given the large amount of data that needs to be transferred in respect of solutions based on traditional approaches.

According to Bellifemine *et al.,* (1999); mobile agent frameworks are currently addressed by two standard bodies. The Federation of Intelligent Physical Agent (FIPA) looks at high-level semantically rich interactions between software agents that deploy some form of intelligent adaptability. It has its roots in Distributed Artificial Intelligence (DAI). The second, Object Management Group (OMG) looks mostly at the issue of mobility according to a standard interoperable framework through its Mobile Agent System Interoperability Facility (MASIF) (Bellifemine *et al.*, 2001). In the latter, agent systems model the execution environment are able to host mobile agents.

2.2 JADE - JAVA AGENT DEVELOPMENT FRAMEWORK

JADE is a software framework fully implemented in Java language. It simplifies the implementation of multi-agent systems through a middle-ware that supports the debugging and deployment phase. The agent platform can be distributed across machines which not even need to share the same Operating Systems (OS) and the configuration can be controlled via a remote Graphical User Interface (GUI). The configuration can be even changed at run-time by moving agents from one machine to another one, as and when required. The only system requirement is the Java Run Time version 1.4 or later.

According to Bellifemine *et al.*, (1999) and Caire *et al.*, (2010), JADE is totally in compliance with the Foundation for Intelligent Physical Agents (FIPA) for inter-operable intelligent multi-agent systems. FIPA is an international non-profit association of companies and organizations sharing the effort to produce specifications for generic agent technologies. JADE complies with FIPA, which includes the Agent Management System (AMS), the default Directory Facilitator (DF) and the Agent Communication Channel (ACC). JADE automatically activates these three agents when the agent platform starts-up. The platform provided by JADE is distributed. It can be split over several hosts with one of them acting as a front end for management and inter-platform communication (Bellifemine *et al.*, 2001). A JADE platform comprises of one or more

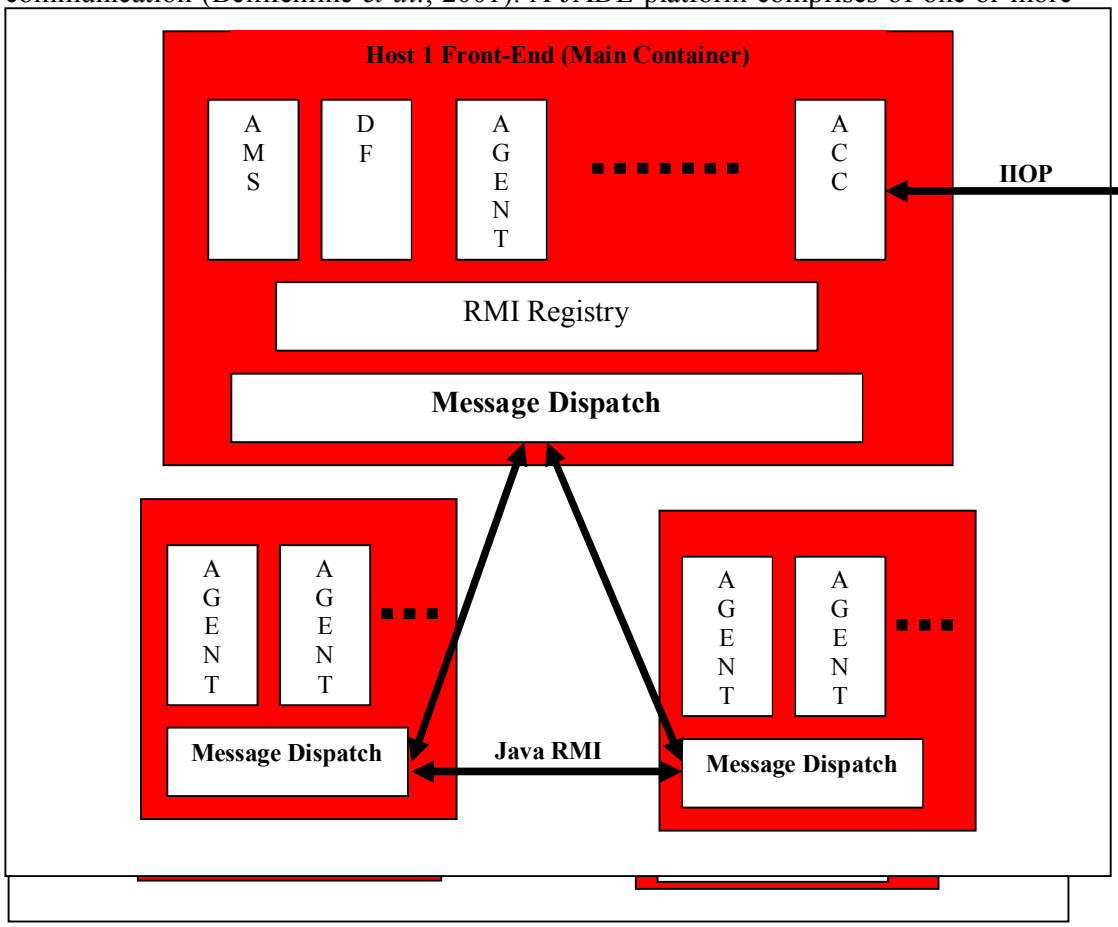


Figure 2: Software Architecture of One JADE Agent Platform

According to Bellifemine *et al.,* (2001) and Caire *et al*., (2010), message passing is used to communicate between agents. The FIPA Agent Communication Language (ACL) is the language used to represent messages. JADE tries to choose the most efficient way to pass messages between agents as the message protocol. The goal is to minimize the communication overhead. JADE uses three ways to pass messages:

a. Receiver in the same container of the same platform: Java events are used; the cost is a cloning of the *ACLMessage* object and a local method call.
b. Receiver in a different container of the same platform: Java RMI is used; the cost is a message *serialization* on the sender side, a remote method call and a message *unserialization* on the receiver side.
c. Receiver on a different platform: CORBA IIOP is used; the cost is the conversion of the *ACLMessage* object into a String object and an IIOP *marshalling* on the sender side, a remote method call and an IIOP *unmarshalling* followed by ACL parsing on the receiver side.

Regarding the agent execution model, JADE uses a thread-per-agent concurrency model instead of a thread-per-behavior model in order to keep small the number of threads required to run the agent platform. JADE uses Behaviour abstraction to model the tasks that an agent is able to perform. Therefore, the agent developer should extend the Agent class and implement the agent-specific tasks through one or more behaviour classes. Finally behaviours are *instantiated* and added to the agent. In addition, JADE also provide ready-to-be-used library of FIPA interaction protocols. JADE is still an ongoing project, its development and improvement is continuing.

CHAPTER 3

MATERIALS AND METHODS

This chapter describes the design and implementation of the mobile agent-based network management system. First, the chapter described the system architecture and design; then, presented the implementation details. This system is created on top of the JADE agent platform. The entire code is written in Java.

3.1 SYSTEM ARCHITECTURE

As initially proposed by Bieszczad *et al.,* (1998) and depicted on Figure 3, the system consists of two main parts: Network Manager (NM), and Network Element (NE). At the NM, there is a Manager Agent which first needs to be created and started up. Network administrator uses it to monitor the network status and control the manipulation of the system through its Graphical User Interface (GUI). At each NE, there is a daemon agent which is dispatched by the NM when the NE registers itself into the management domain. At the NM, there are also Mobile Agent Generators (MAG) which could be used to create customized mobile agents.

3.1.1 Network Manager

The NM is the control center for the system. It is responsible for launching mobile agents and displaying the results returned by them. Considering its importance and also for security purpose, users need to be authenticated before logging into the system. After the system checks the correctness of the provided user ID and corresponding password, it shows the first GUI (register GUI) to users, waiting for the NE to register and join into the management domain. Once one NE has joined the system, the NM dispatches one type of mobile agent, the daemon agent, to the registered NE which

remains there until it is killed by the NM or the NE is down. After initialization, the NM communicates with the dispatched daemon agents to do the network status monitoring.

At the NM, users can also use the Mobile Agent Generators (MAG) to create the mobile agent skeleton and use the MAG GUI to customize it for specific purposes. After customization, it is dispatched to the whole network to do some domain or global management. Figure 3 below shows this interaction (Bieszczad *et al.,* 1998).

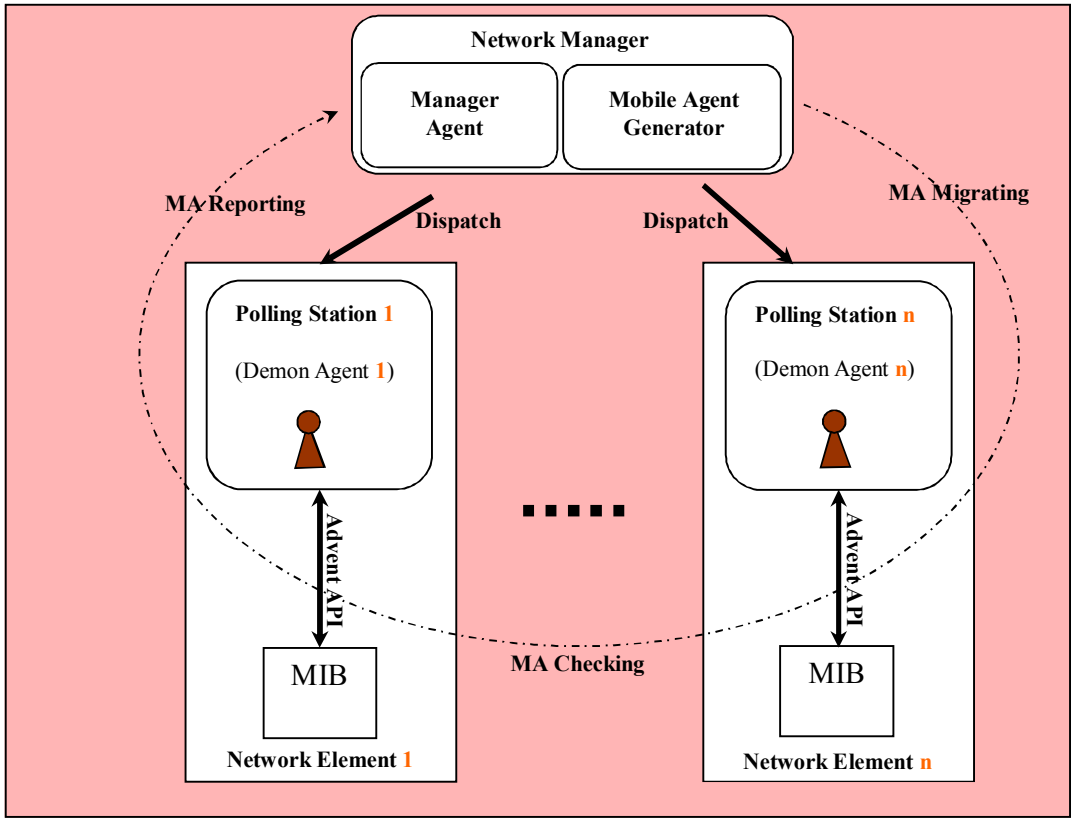


Figure 3: Architecture of Mobile Agents

The functionalities realized by the Manager Agent and related parts are described in the following sections.

3.1.2 Network Element

If some NE, such as a router, switch, or workstation, wants to join into the management domain, the mobile agent container needs first to be deployed at the NE or at a nearby polling station if the deployment is inhibited. The container at that NE is then started and the register agent is used to send REGISTER message to NM. Once the message is accepted by NM, NM dispatches a daemon agent to the NE which resides there for getting and setting the Management Information Base (MIB) values.

For the device to be managed, it needs to support SNMP. At its polling station, the *AdventNetAPI* package needs to be deployed as well to serve as the SNMP Application Programming Interface (API) to contact with SNMP agent.

3.2 MANAGER AGENT

The Manager Agent is the core part of NM. It performs monitoring and control operations through its interaction with devices running agent processes. It contains several GUI and bebaviour classes as shown in Figure 4.

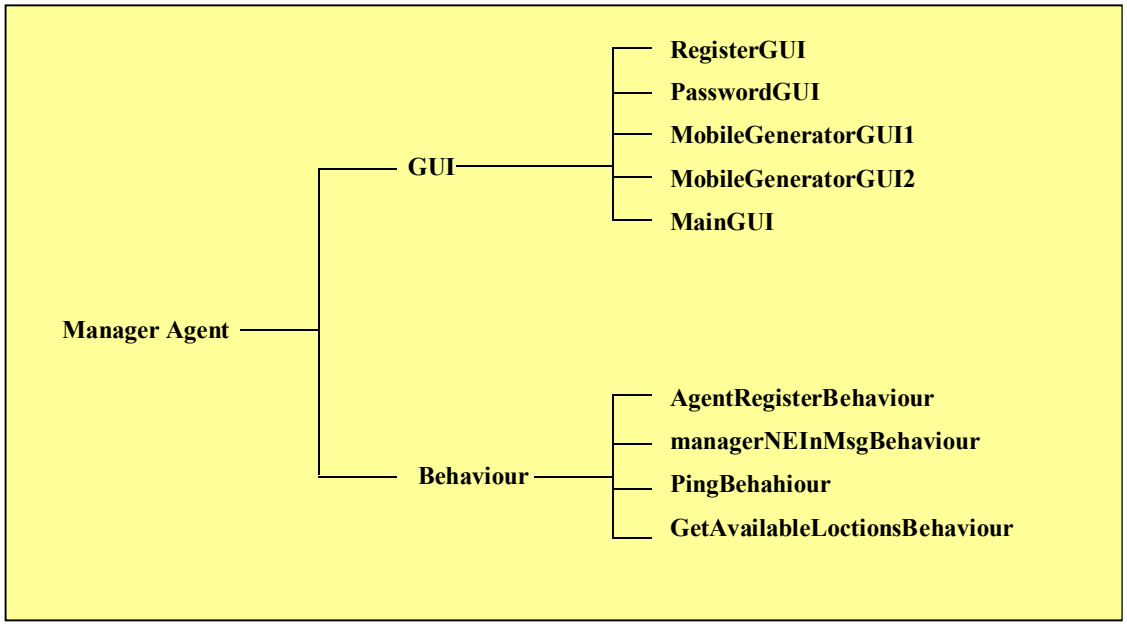


Figure 4: Class Components of Manager Agent

GUIs include *PasswordGUI, RegisterGUI, MainGUI, MobileGeneratorGUI1* and *MobileGeneratorGUI2* which are used by NM to view the network. The bebaviour classes include *GetavailableLocationBehaviour, ManagerNEInMsgBehaviour PingBehaviour*, and *AgentRegisterBehaviour*.

3.2.1 Communication of components

Because Manager Agent has GUIs, it is an extension of *jade.gui.GuiAgent* class, which is a simple extension of *jade.core.Agent* class. In JADE, it defines that: A thread (in particular the GUI) wishing to notify an event to an agent should create a new object of type *jade.gui.GuiEvent* and pass it as a parameter to the call of the method *postGuiEvent()* of the *jade.gui.GuiAgent* object. After the method *postGuiEvent()* is called, the agent reacts by waking up all its active behaviours, and in particular the behaviour that causes the agent thread to execute the method *onGuiEvent()*. As a consequence, an agent wishing to receive events from the GUI should define the types of events it intends to receive and then implement the method *onGuiEvent()* (Bellifemine *et al.,* 2001).

In the Manager Agent, there are two possible means of interaction. First, modifying the GUI when an Agent Communication Language (ACL) message is received. Second, executing some behaviour according to the posted event from the GUI thread because of the user's action through the GUI. The following section illustrates some possible interactions.

3.2.1.1 Receiving a Register Message

AgentRegisterBehaviour is responsible for waiting for the coming register message sent by the NE. After receiving a message, it fetches the content of the message, gets the NE host name and site location, and calls the *doRegister(Location site, String hostname)* of

the Manager Agent. In this function, it first creates a *NEAgent(daemon agent)* and sends it to the NE. It then calls the *registerGui*'s method *addlist(agentname, hostname)* to reflect the updated list in the *registerGUI*. Finally, it adds the *PingBehaviour* corresponding to the NE site in the Manager Agent by calling function *addBehaviour(new PingBehaviour(this,agentid))*. It is responsible for checking the status of the NE just registered. For example, if one *NE(host finance.ict.unicef.org)* wants to register itself into the management domain, it sends the register ACL message to the Manager Agent. After the message is received by the *RegisterBehaviour*, it informs the Manager Agent. Manager Agent sends daemon agent *NEAgent2* (there is a NE *adminServer.ict.unicef.org* already registered and *NEAgent1* has been sent to it) to *finance.ict.unicef.org* and update list in the *RegisterGUI* as shown in Figure 5. "Ok" means the registered NE is running fine. It will show "DOWN" whenever the manager finds the NE is down.



Figure 5: Registration Example

At the *registerGui* shown above, administrator may try to view the monitor panel for the chosen NE element from the list by clicking on the View Monitor Panel button. This

action creates a *GuiEvent REG_VIEW_EVENT*, sets the parameters as *agentID* and hostname and posts it to the Manager Agent. Manager Agent reacts to this event by creating the *MainGui* and communicating with the chosen NE by sending and receiving Agent Communication Language (ACL) messages.

3.2.2 Manager Agent Behaviours

3.2.2.1 Manager Agent Behaviours' Scheduling

As previously mentioned, a JADE agent is implemented as a single Java thread. Although it has its own internal thread of control, it can engage in multiple simultaneous conversations. JADE uses the Behaviour abstraction class to model the tasks to that an agent is able to perform and agents instantiate their behaviours according to the needs and capabilities.

The Manager Agent must be able to carry out several concurrent task in response to different external events. This implies that the behaviours of the Manager Agent must be scheduled cooperatively. A scheduler, implemented by the base Agent class and hidden to the programmer, carries out a round-robin non-preemptive policy among all behaviours available in the ready queue, allowing the execution of a Behaviour-derived class until it releases the execution control on its own. If the task relinquishing the control has not yet completed, it will be rescheduled for the next round unless it is blocked.

The Manager Agent has two behaviours, *RegisterBehaviour* and *ManagerNEInMsgBehaviour* which are added at the agent setup stage. The behaviour, *GetavailableLocationBehaviour*, is added when the mobile agent is first created.

PingBehaviour is added after a NE is registered with the Manager Agent. Each NE has one corresponding *PingBehaviour*. The scheduling of behaviours after all behaviours have been added is shown in Figure 6.

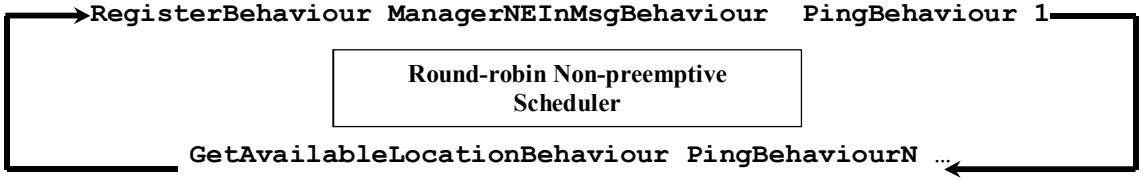


Figure 6: Behaviours' scheduling of the Manager Agent

3.2.2.2 Implementing Agent Bebaviour Classes

In detail, the agent scheduler executes the *action()* method of each behaviour present in the ready behaviours queue, when *action()* returns, the method *done()* is called to check whether the behaviour has completed its task or not. If so, the behaviour object is removed from the queue. If not, the behaviour is scheduled for next round. There is no stack to be saved and there is no way to interrupt a behaviour in the middle of its *action()*. As a result, it is important to avoid to use endless loops and to perform long operations within *action()* method. If an operation is too long to be run in a single step, it is broken into several sub-operations. For example, Figure 7 lists the code segment of the *ManagerNEInMsgBehaviour*.

In *ManagerNEInMsgBehaviour*, it is necessary to wait for several kinds of ACL messages from NE, such as *PerformanceIndicator, MIBTreeObjectSet, MIBTreeObjectGet, SNMPTablePolling* and Warning. As the code illustrates in Figure7, the problem can be solved by just calling to receive one kind of message at a time. While in the interval, it could give other behaviours, such as *PingBehaviour*, the chance to get the CPU.

```
class ManagerNEInMsgBehaviour extends SimpleBehaviour
{ ……
   final int FIRST = 1;
   final int SECOND = 2;
   final int THIRD = 3;
   private int state = FIRST;
    ……
   public boolean done(){ return false; }
   public void action() {

        switch (state){
        case FIRST: {op1(); state = SECOND; break;}
        case SECOND:{op2(); state = THIRD; break;}
        case THIRD: {op3(); state = FORTH; break;} ……
        }
}
private void op1(){
   MessageTemplate m1 = MessageTemplate.MatchPerformative(ACLMessage.INFORM);
   MessageTemplate m2 = MessageTemplate.MatchLanguage("PlainText");
   MessageTemplate m3 = MessageTemplate.MatchOntology("performanceIndicators");
   MessageTemplate m1andm2 = MessageTemplate.and(m1,m2);
   MessageTemplate andm3 = MessageTemplate.and(m1andm2,m3);
   ACLMessage msg = myAgent.receive(andm3);
   if (msg!= null){
        System.out.println("per_indi="+msg.getContent());
        //post the indicator to the appropriate place
        ((Manager)myAgent).postIndicator(msg.getContent());
   }
   else return;
}
private void op2(){ …… }
private void op3(){ …… } ……
```

Figure 7: Code segment of ManagerNEInMsgBehaviour Class

3.3 DAEMON AGENT (NEAGENT)

NEAgent is a mobile agent which is created at the time of NE registration. After it is created, it is launched to the NE site and stays there as a daemon agent. As mentioned in

the introduction, mobile agents could be made to move around to fetch data while doing some filtering. Why is it still necessary to have a daemon agent at the NE site? The reason is that there is the need to monitor the status of the NE and obtain feedback in real time while Mobile Agents could be launched anytime as required. However, if MA is not sent out in time, tendencies are there that detection of certain spontaneous faults could be missed. It may cause more problems if not handled as soon as possible. With the daemon agent always resident at the NE however this problem could be solved by the daemon agent which will check the current status of the NE, get values and send them back to manager at the time interval set previously by users.

NEAgent has no GUI but has several behaviours, *NEAgentInMsgBehaviour, NEPingBehaviour*, and *CheckPIBehaviour*. The class implementation and behaviours scheduling are similar to Manager Agent. *NEPingBehaviour* is responsible for responding to the ping inquiry from Manager Agent and sends the pong message back. *NEAgentInMsgBehaviour* checks the messages from Manager Agent, such as *setPollingInterval, getMibValue, stopPolling. CheckPIBehaviour* continues computing the required performance indicators at the polling interval after the *NEAgent* arrived at the NE site.

After *NEAgent* arrived at its destination, it instantiates the *InterfaceUtilization* class, which tries to negotiate with the SNMP Agent to get the MIB values by using the *AdventNetAPI*. After computation, it sends the result back to the manager. If the value is higher than a fixed threshold as defined by the user, it sends a warning message back to the NM to notify some bad things that had happened or will happen.

The *NEAgentInMsgBehaviour* listens to the coming GET/SET requirements from the Manager Agent. In the case of a GET message, it checks the value for the specified MIB

node in the MIB tree and creates another ACL reply message to send the results back. In the case of the SET message, it tries to set the MIB value according to the user specified new value. If value is successfully set, it informs the manager of the success. Otherwise, it informs the manager of the failure of this operation.

Figure - 8 shows the communication between the manager and daemon agents. It also shows the health function calculation and MIB value getting/setting by the daemon agents.

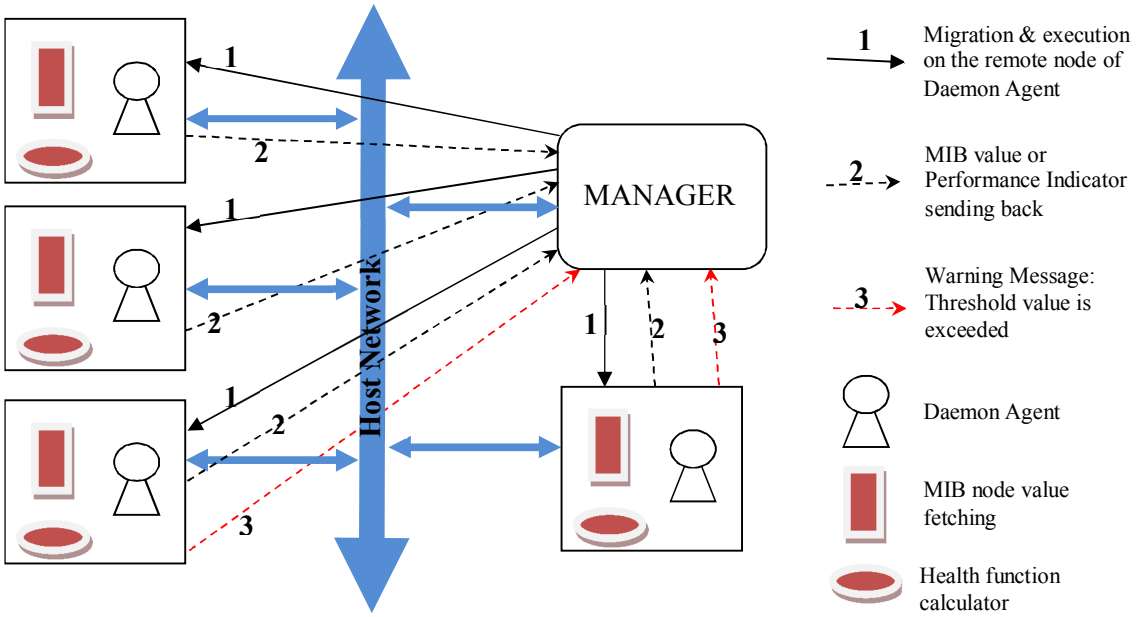


Figure 8: Communication between Manager and Daemon Agent

3.4 HEALTH FUNCTION (PERFORMANCE INDICATORS)

Polling is a frequent operation in network management as there are often several object values that require constant monitoring. There are cases however, where one or two MIB variables are not a representative indicator of system state and hence an aggregation of multiple variables is required, known as its health function (HF). There are several important health functions computed by the daemon agent.

3.4.1 Interface Utilization

Utilization measures the use of a particular resource over time. The measure is usually expressed in the form of a percentage in which the usage of a resource is compared with its maximum operational capacity. Through utilization measures, administrators can identify congestion (or potential congestion) throughout the network.

The primary measure used for network utilization is interface utilization. The following formulas are used, depending on whether the connection you measure is half-duplex or full-duplex. Shared LAN connections tend to be half-duplex mainly because contention detection requires that a device listens before transmitting. WAN connections are typically full duplex because the connection is point to point; both devices can transmit and receive at the same time since they know there is only one other device sharing the connection.

For half duplex media, use the following formula for interface utilization, the meaning of each parameter is shown in Table - 1.

$$\frac{(ifInOctet + ifOutOctects) * 8 * 100}{ifSpeed * SysUpTime}$$

Equation 1: Interface utilization, Half Duplex

Table 1: MIB II Variables for Computing Interface Utilization

NOTATION	Description
IfInOctets	The count of inbound octects of traffic
IfOutOctets	The count of outbound octets of traffic
IfSpeed	The speed of the interface
SysUpTime	The time duration after the system start up

For full duplex media, the following formula is used for interface utilization. For example, with a full T-1 serial connection, the line speed is 1.544Mbps. This means that T-1 interface can both receive and transmit 1.544Mbps for combined possible bandwidth of 3.088 Mbps.

$$\frac{\max(ifInOctets, ifOutOctets) * 8 * 100}{ifSpeed * SysUpTime}$$

Equation 2: Interface utilization, Full Duplex

However, this method hides the utilization of the direction (in or out) that has the lesser value and provides less accurate results. A more accurate method would be to measure the input utilization and output utilization separately.

3.4.2 Error Rate and Accuracy

Accuracy is the measure of interface traffic that does not result in error and can be expressed in terms of a percentage that compares the success rate to total packet rate over a period of time. It is first necessary to measure the error rate. For instance, if two out of every 100 packets result in error, the error rate would be 2% and the accuracy rate would be 98%. Some common causes of interface errors include:

a. Out-of-specification wiring
b. Electrical interference
c. Faulty hardware or software

The variables shown in Table 2 are used in accuracy and error rate formulas:

Table 2: MIB II Variables for Computing Interface Error Rate and Accuracy

NOTATION	DESCRIPTION
IfInErrors	The count of inbound packets with an error
IfInUcastPkts	The count of inbound unicast packets
IfInNUcastPkts	The count of inbound non-unicast packets (muticast and broadcast)

The formula for error rate is usually expressed as a percentage:

$$ErrorRate = \left(\frac{\Delta ifInErrors}{\Delta ifInUcastPkts + \Delta ifInNUcastPkts}\right) * \mathbf{100}$$

Equation 3: Interface Error rate

The formula for accuracy takes the error rate and subtracts it from 100.

$$Accuracy\ rate = 100 - \left(\frac{\Delta ifInErrors}{\Delta ifInUcastPkts + \Delta ifInNUcastPkts}\right) * \mathbf{100}$$

Equation 4: Interface Accuracy rate

3.4.3 Internet Protocol (IP) Output Datagrams Discard Rate

The discard rate defines the percentage D_r of IP output datagrams discarded over the total number of datagrams sent during a specific time interval. It is calculated by combining the five MIB-II objects *(Gavalas et al., 2000)* and is shown on equation 5 below.

$$\text{Dr} = \left(\frac{(ipOutDiscards + ipOutNoRoutes + ipFails)}{(ipOutRequests + ipForwDatagrams)}\right) * 100$$

Equation 5: (IP) Output Datagrams Discard Rate

As described in section 3.4 above, those health functions are computed by daemon agent. Only the final result will be sent back. While in traditional network management, all the MIB II values will be fetched by the centralized manager. All the computation workload are done at the manager which degrades its efficiency. Although the daemon agent becomes the static agent once it has arrived at its destination, it also helps release some burden of the manager by doing the health function computation locally.

3.5 MESSAGE DEFINITION

ACL Messages are transferring back and forth between Manager Agent and Daemon Agent. The ACL messages are one of the following types:

a. Register and Unregister
b. Ping and Pong
c. Performance Indicators
d. MIBTreeObjectGet
e. MIBTreeObjectSet
f. MIBSNMPTablePolling
g. Warning

For example, the ACL Message for *MIBTreeObjectGet* is defined in Table - 3:

Table 3: Definition *of MIBTreeObjectGet* ACL Message

Language	***PlainText***
Ontology	*MIBTreeOjectGet*
Content (Manager *NEAgent*)	.1.3.6.1.2.1.2.1.0
Content (*NEAgent* Manager)	2

3.6 COMMUNICATION OF MANAGER AGENT WITH DAEMON AGENT

As an example, Figure 9 shows the Monitor Panel (*MainGui*) of host finance.ict.unicef.org.

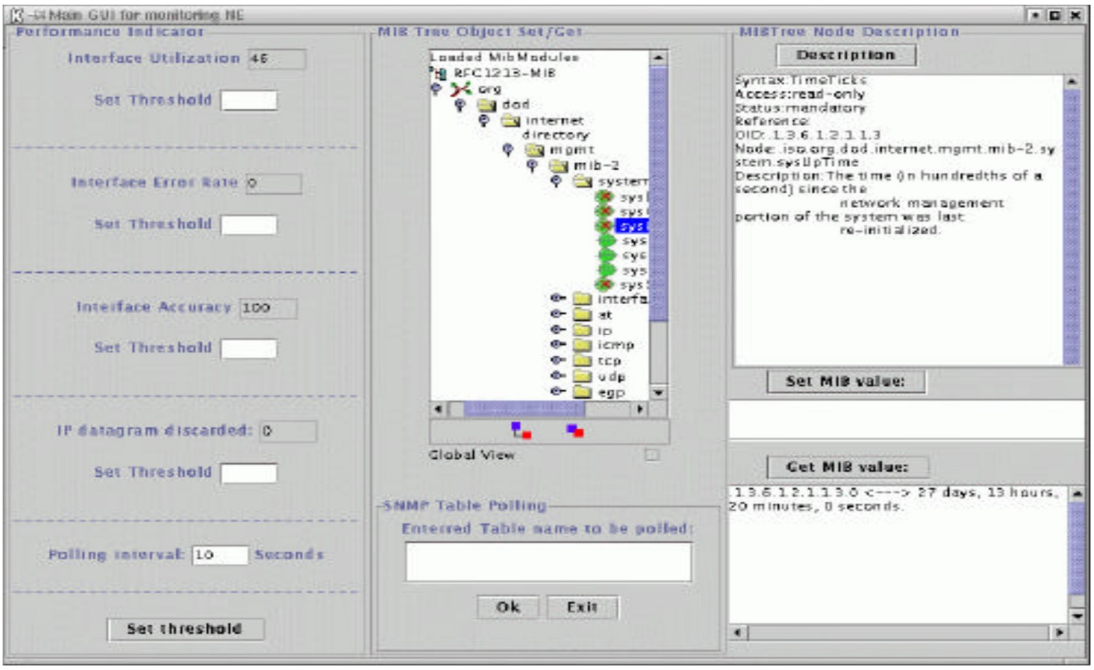


Figure 9: MainGui of Manager Agent

In the health function panel, it shows that the current utilization is 46%, the error rate is 0% and accuracy is 100%, the IP output discard rate is 0%. These data are refreshed at the specified polling interval which will be set by the user and sent to the daemon agent. In the middle panel, the user could choose from the MIB tree which node he/she want to poll or set new value in the set *TextField*. Users can also poll the table by specifying the table name in SNMP table *TextField*. In the description *TextField*, user can also get more detailed information about this chosen MIB tree node. User's requirement will be sent to the daemon agent as the ACL message. After the daemon agent gets the message, checks the value, replies back with the value, the result will be shown at the manager side as Figure 9 shows.

3.7 MOBILE AGENT (MA)

Two types of mobile agents have been implemented in the system to do some intelligent filtering applications. The MA can be created through the Mobile Agent Generator (MAG). One, for SNMP table intelligent filtering, the other for global filtering. Those ideas are based on the work in the paper: Advanced Network Monitoring Applications Based on Mobile/Intelligent Agent Technology Gavalas *et al., (2000)*. The design and implementation is based on the JADE Agent Environment.

To quickly create MA and easily tailor it to some specific requirement, mobile agent skeleton are created beforehand. Users can customize the service-oriented MA through the MAG GUI. A MA object is identified by its code (behavioural description), state information (modifiable variables) and attributes (static/permanent information). From the programmer's perspective, MAs are Java classes extending the *jade.core.Agent* class. As defined in JADE, MAs need to be location aware in order to decide when and where to move. The public method *doMove()* of the Agent class allows a JADE agent to migrate elsewhere. The method takes a *jade.core.Location* as its single parameter, which represents the intended destination for the migrating agent.

Moving an agent involves sending its code and state through a network channel, as a result, user defined mobile agents must manage the *serialization* and *unserialization* process. Among the various resources, some used by the mobile agent will be moved along, while some others will be disconnected before moving and reconnected at the destination. This is the same distinction between transient and non-transient fields used in the Java Serialization API.

For agent migration, the *beforeMove()* method is called at the starting location just before sending the agent through the network (with the scheduler of behaviours already

stopped), whereas the *afterMove()* method is called at the destination location as soon as the agent has arrived and its identity is in place. MA also needs to register with the Jade-mobility-ontology by using *registerOntology(MobilityOntology.NAME, MobilityOntology.instance())*. It contains all the concepts and actions needed to support agent mobility.

3.7.1 SNMP table filtering

Most network management applications normally involve the transferring of bulk monitoring data, for example, large SNMP tables. In SNMP, the SNMP table is obtained by repeatedly using the GET-NEXT operation or using the GET-BULK operation. Those methods have some drawbacks. For each GET-NEXT operation, there is the need to wait for its response before the next GET operation is invoked. But for each GET, only one table row is received. If the table was large, containing thousands of entries, this operation would have apparent impact on network resources. It also experiences significant latency, especially when the management of remote LANs is considered. In addition, the non-negligible time intervals between acquisitions of each individual object value leads to potential inconsistencies (Gavalas *et al.,* 2000).

The *GET-BULK* operation improves this situation by transferring bulk of data. But users need to guess previously the maximum number of data to be got. If it is larger than the number needed, more useless data will be fetched and the network usage will be wasted. However, if it is smaller, it may result in too more message exchanges.

In addition, the object identifiers (OIDs) of the objects involved in bulk transfers also result in higher overhead because there is a lot of redundancy. MA has the potential to improve the retrieval of SNMP tables in terms of network overhead. MA could be

launched to NE where the network operator wishes to retrieve SNMP values. Using the successive GET-NEXT request, MA gets the snapshot of the table, encrypts and saves it.

Traditionally, those bulk data fetched back would be filtered by the NM for further analysis. By pushing the filtering operation down onto NE, the latency and bandwidth needed to obtain the results could be significantly reduced. Once the MA has the SNMP table it can flexibly filters the values according to a filter saved in it. The filtering expression was defined by user through the MA Generator GUI. For example, Figure10 shows one customized MA Generator GUI.

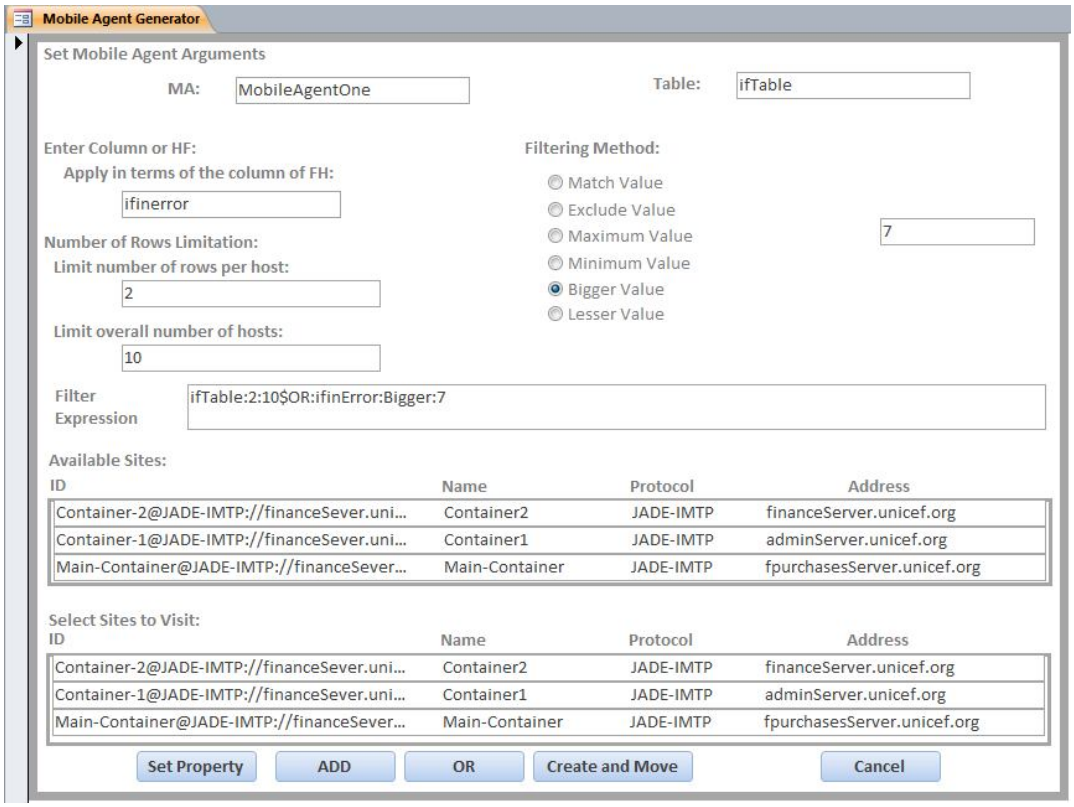


Figure 10: Mobile Agent Generator

As shown in the Figure10, the user needs to give a name to the MA, which would be used to instantiate itself at the destination. At the table and column *TextField*, user sets which SNMP table needs to be fetched and which column needs to be filtered. Several

columns' filtering could be combined by using "AND", "OR". All the available NE sites are listed when the MAG GUI shows up. The itinerary table is formed by choosing from the available NE sites and the last destination needs to be set as the Main Container (where Manager Agent exists). Then MA moves from NE to NE and returns to the starting place to report the results. The example shown in Figure10 is to fetch the *ifTable* at *finance.ict.unicef.org, adminServer.ict.unicef.org* and filter according to column *ifInError(ifInError.value >7)*. The filtering method can be specified in terms of the following operations: *Match, Exclude, Max, Min, Bigger, Less*. SNMP table acquired will be fed into those filtering operations, only keeping the row which corresponding columns meet the criteria. Figure11 shows the flow diagram of SNMP table filtering by MA.

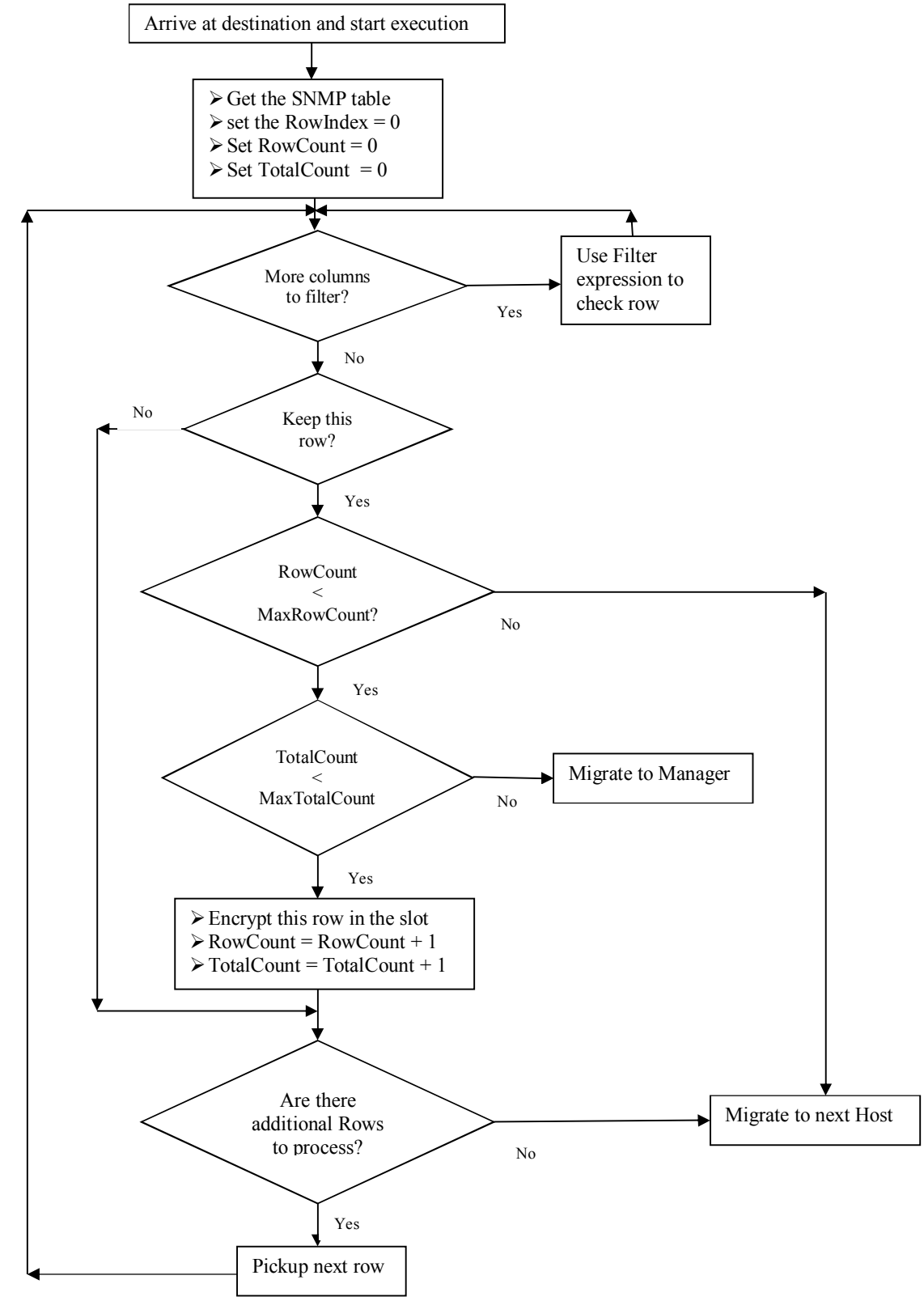


Figure 11: SNMP Table filtering Flow Diagram

The results for the example SNMP table filtering by MA is listed in the table GUI shown in Figure - 12.

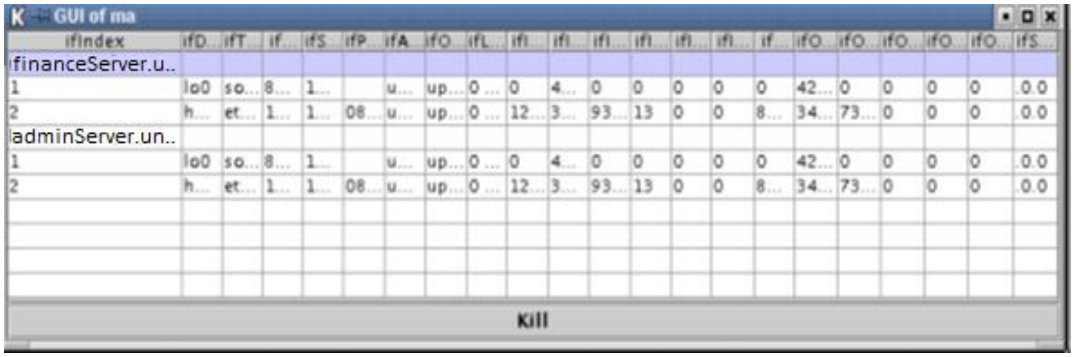


Figure 12: Results Table of SNMP Table Filtering MA

3.7.2 Global Filtering

Another kind of MA can be used for domain or global filtering by comparing/merging the results already saved with those that have just been fetched by either the performance indicator/health function or SNMP filters through mobile agent. In this level, MA could bring more benefits. Not only the manager is being relieved from handling heavy processing tasks, but also the MA's state size is prevented from growing rapidly. Right now, in the implementation, only domain/global filtering are being supported based on the following health functions: *InputUtilization, OutputUtilization, IfInErrorRate, IfAccuracy, IPOutputDiscardRate*. For example, there may need to fetch the two most heavily loaded interfaces, the results of which would be returned back to manager site and displayed in a table GUI.

The skeleton for this type of MA is also created beforehand. After user entered the specific requirement in the MAG GUI and click on the *CreateandMove* button, this MA will be instantiated and customized with those specific features saved in it and brought

along with it while migrating. Figure 13 shows one example of this type of agent generator.

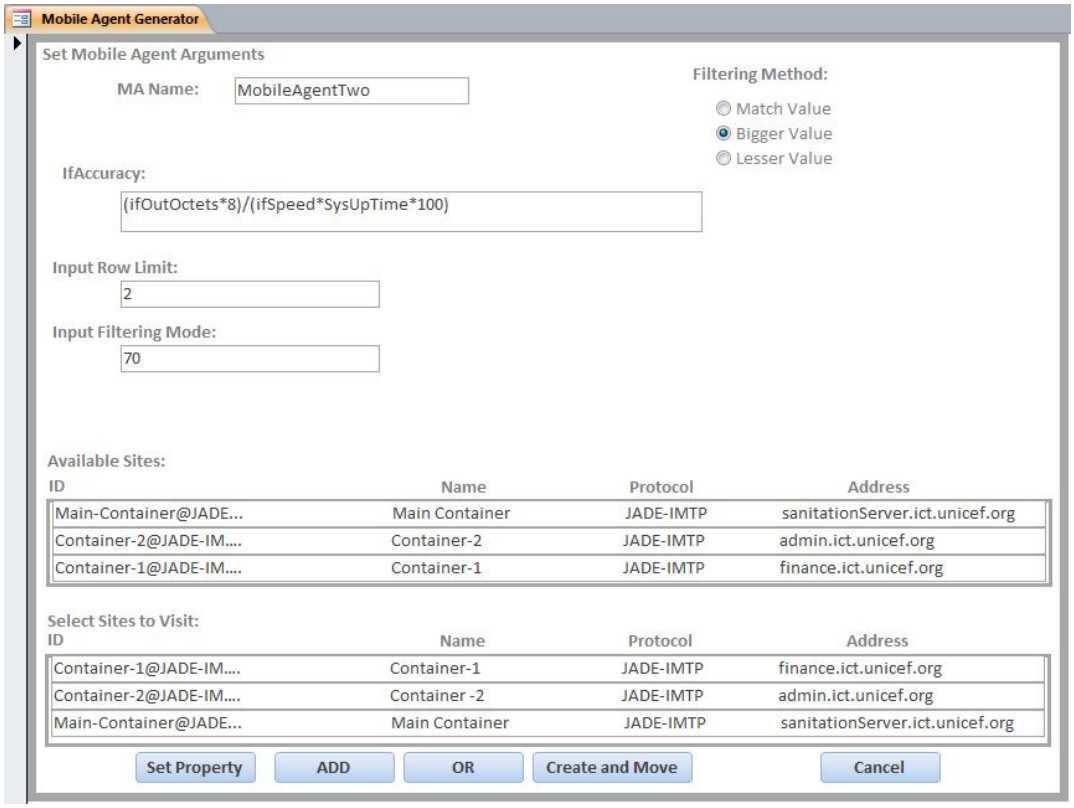


Figure 13: Example of Global Filtering MA Generator GUI

As the example shown, there is the need to fetch the two NEs whose interface accuracy are the highest among the chosen NE list with at least 70% accuracy. After MA get this task, it will go around, compute and compare, finally return the two NEs names and interface accuracy values back and show in a graphical table.

The results brought back by global filtering MA is listed in the table GUI shown in Figure 14.

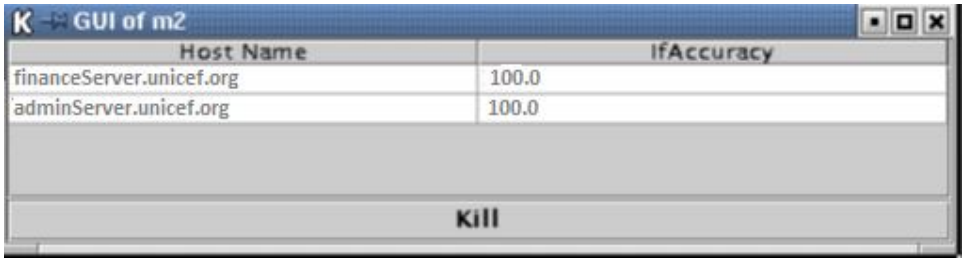


Figure 14: Results Table of Global Filtering MA

CHAPTER 4

RESULTS AND DISCUSSION

The current prototype has demonstrated the usefulness of the properties provided by MA to help NM to do the network management. In this Chapter, some related problems that have arisen during the work have been discussed. There are also several interesting future directions that could be explored as an extension of this work. These are:

a. security problem, and
b. Scalability problem.

4.1 SECURITY PROBLEM

Network management normally involves transferring many important management data through the network. Some are critical to the network safe status and disasters may happen if they are eavesdropped and improperly set by malicious users. Reiser and Vogt (2000) assert that introducing the MA into this picture brings new security problems because automatically executing code on any host is potentially dangerous. Thus, security plays a decisive role in terms of acceptance and applicability of agent-based network management system.

Security issues have been considered in SNMP. But in SNMPv1 and SNMPv2, only a community-based simple security scheme is used. The practice of transmitting community strings (passwords) in clear text posed a severe security problem. The new version, SNMPv3 was developed mostly to address this shortcoming. New security features were introduced to ensure message authenticity, integrity and confidentiality, and to provide fine-grained control over access to SNMP capabilities. In the developed

system that, the MA needs to communicate with the SNMP to GET/SET MIB object. It uses *AdventNetAPI* SNMPv3 to ensure more security protection. However, securing the MIB value polling at the NE end is not enough, the whole system is based on Agent-platform, especially relies on MA which needs more consideration of security. Security is a relative concept: it makes no sense to claim a system is secured without explicitly stating what kinds of threats the system can successfully withstand. The threats could be considered in three different ways. First, security threats by malicious MA; Second, security threats by malicious host; Third, security threats from network communication (Fu, 2001). Considering those kinds of threats, an attempt has been made to protect the network management system in three different ways.

4.1.1 Protecting NE from malicious MA

In JADE, after MA arrived at its destination, the NE, it needs to be first instantiated and then executed. It is the container that acts as the complete run time environment for agent execution. Each Agent Container is a Remote Method Invocation (RMI) server object that locally manages a set of agents. It controls the life cycle of agents by creating, suspending, resuming and killing them. According to Reiser and Vogt (2000), there is no security mechanism at this point in the current version of JADE and therefore, they proposed the following suggestion and plan to secure the MA under such circumstance.

After the NE received the serialized instance of the MA, before reconstruct it, the Container at that location needs to verify the authenticity of the received MA through the use of security keys, ensuring only the trusted agents, dispatched by authorized manager, the received MA's are allowed to be instantiated. The RSA algorithm, based on the public-private pair of keys paradigms could provide both authentication and

encryption features. Certificates and digital signatures are used by MA's for authentication to ensure data integrity of the MA's. In digital signature, instead of encrypting the data itself, the signing software creates a one-way hash of the data, and then uses the private key to encrypt the hash. The encrypted hash, along with other information, such as hashing algorithm, is known as its digital signature. The digital signature will be sent along with the data to the receiver. The receiver first uses the signer's public key to decrypt the hash. Then receiver uses the same hashing algorithm that generated the original hash to generate a new one-way hash of the same data. Finally, the receiver compares the new hash against the original hash. If the two hashes match, the data has not changed since it was signed. Receiver can also be confident that the public key used to decrypt the digital signature corresponds to the private key used to create the digital signature. A certificate is an electronic document used to identify an entity associated that identity with a public key. It is like the passports identify a person with an ID. A certificate is normally issued by Certificate authorities (CAs).

The following describes the authentication of MA for the network management system. An assumption here is that the communication channel has already been secured through the use of Secure Socket Layer (SSL) connection as described in section 4.1.3. Also assumed is that the manager has received the certificate from the Certificate Authority. Figure 15 shows the authentication step:

a. Manager plans to send a MA agent. It retrieves its private key and uses it to encrypt the MA's code as the digital signature.
b. Send the certificate and digital signature along with the MA across the network.
c. After NE container gets the MA, it retrieves the public keys from the certificate and tries to make sure that the arriving MA is sending from authorized host

(Manager) and it has not been tampered with during the traveling. If it passes all the checks, the MA can then be instantiated and further executed.

d. Travel to the next NE according to the itinerary table.

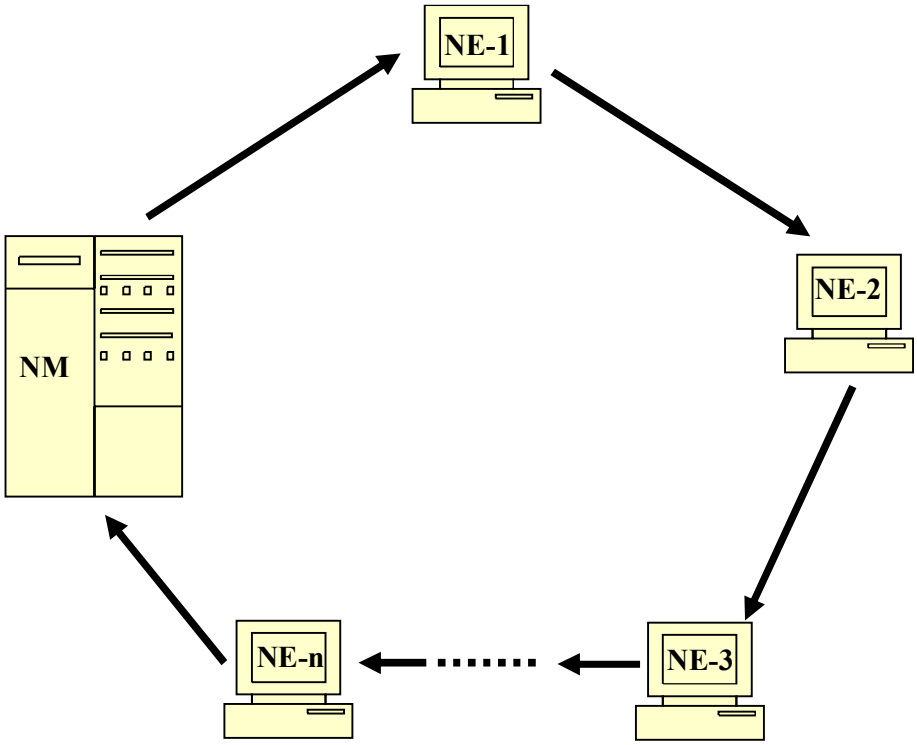


Figure 15: Authentication of MA

There are many related issues regarding the Public Key Infrastructure (PKI), such as public key distribution, private key distribution and protection, and so forth. These are out of the scope of this thesis.

4.1.2 Protecting the important data in MA

The whole system's function is based on MAs, both daemon agent and filtering mobile agent. They transfer or carry important management data which needs to be protected from eavesdropping and tampering. As proposed by Fu, (2001) I choose the Java

Cryptography Environment (JCE) API to do the encryption and decryption of the data in MA. In the daemon agent, after it gets the MIB value or computes the health function, it needs to encrypt the data, and send it as the content of an ACL message to the Manager Agent. After the message is received by the Manager Agent, it is decrypted for use.

In the filtering mobile agent, after it fetches the MIB node value and computes the result, it encrypts the data before moving to the next one. At the next stop, if the comparison is needed, it decrypts the data first, does the comparison, re-encrypts the data, and repeats these steps until finally returns back to the manager. At the last destination, the message decrypts and shows in the table GUI.

In Figure -16 and Figure -17, the encryption and decryption of an important message in daemon agent are illustrated:

```
SecureRandom sr = new SecureRandom("password".getBytes());
KeyGenerator kg = KeyGenerator.getInstance("Blowfish");
kg.init(sr);
SecretKey sk = kg.generateKey();

Cipher bf = Cipher.getInstance("Blowfish");
bf.init(Cipher.ENCRYPT_MODE,sk);
byte[] cleartext = "This is the message to be
encrypted".getBytes();
byte[] ciphertext = bf.doFinal(cleartext);
```

Figure 16: Encrytion of Data Using JCE API

```
// decrypt the data back
// fetch the sk
bf.init(Cipher.DECRYPT_MODE,sk);
byte[] decryptedtext = bf.doFinal(ciphertext);
```

Figure 17: Decryption of Data Using JCE API

4.1.3 Secure the Communication Channel

Although efforts have been made to protect the NE and management data, the effort would be useless if the network communication is insecure. The attacker may change the content of the message and intercept the message, though; he may not be able to read the content because of the encryption. Hence, the ultimate need to secure the communication channels.

In JADE, the communication is normally through RMI (Caire *et al*., 2010). RMI is a distributed object broker above TCP/IP socket. To make it secure, only the TCP/IP socket needs to be secured, for example, through the use of Secure Socket Layer (SSL). The SSL protocol runs above TCP/IP and below higher-level protocols such as HTTP or IMAP (Internet Message Access Protocol). It uses TCP/IP on behalf of the higher-level protocols, and in the process allows an SSL-enabled server to authenticate itself to a SSL-enabled client, allows the client to authenticate itself to the server, and allows both machines to establish an encrypted connection. To facilitate this, the standard TCP/IP socket is replaced with the SSL sockets. This is possible because the version of RMI included in Java 2 SDK enables the RMI developer to use custom socket types such as

SSL. All an application has to do is to install the RMI socket factory that creates sockets of the desired type (SSL socket in this case), (Poggi *et al., 2001*).

4.2 SCALABILITY

Two different solutions for gathering MIB-II variables on managed elements were compared to test network overhead imposed by the mobile Agent. SNMP table filtering MA is compared to the centralized SNMP using *AdventNetAPI* SNMPv3.

The topology used on this experiment consists of one management station (*PurchasesServer.ict.unicef.org*) and two managed NEs (*finance.ict.unicef.org, educationServer.ict.unicef.org*) interconnected through a 100Mbps Ethernet LAN. All machines run Linux Red Hat7.2. The deamon snmpd, which is included in the Linux Red Hat, is an SNMP agent that responds to SNMP request packets.

In order to evaluate the performance of two prototypes for a great number of managed elements, two network elements were alternately repeated in the itinerary table while defining the MA's. For example, to manage 5 network elements, an *itinerary{finance, educationServer, finance, educationServer,finance,PurchasesServer}* was used to simulate or mimic the presence of 5 NEs when actually, there are only two physical NEs.

The MA approach used the SNMP table filtering MA. It fetches the SNMP table and does some filtering based on the user's requirement. The counterpart, centralized solution is implemented using *AdventNet* SNMPv3 package. The manager sends SNMP UDP packets to a SNMP agent that responds to the manager. The manager sends requests to all elements to be managed, one after the other, that is, a new request is started after receiving the response from the previous one, until the last NE receives a

request and sends the response to the manager. It has been implemented in Java directly over the Java Virtual Machine.

The response time of MA is measured as the meantime of the MA launching time and returning time. The centralized SNMP approach is measured as the meantime of the first GET message was sent out and the last result fetched back. Table 4 below lists the testing result:

Table 4: Response Time of SNMP and MA

	Centralized SNMP Approach	**Mobile Agent Approach**
1 host	0.7 Second	0.9 Second
2 host	0.91 Second	1.447 Second
4 host	1.3 Second	1.81 Second
10 host	1.68 Second	3.51 Second
20 host	3.01 Second	6.62 Second
50 host	8.01 Second	16.62 Second

As can be seen from table 4 above, for the listed number of managed elements, the SNMP takes less time than the mobile agent to perform the task. This is due to the fact that the MA is built above JADE which is not efficient for handling mobility.

Java as the implementation language had an influence on the speed. The mobile agent infrastructure makes the execution of Java code slower, mainly because of serialization/deserialization, threads creation, handling security and internal message transmission. If the agent platform was written in C code, the speed may not be a problem.

The SNMP get request/response message is 90 bytes long, on average (at MAC layer), while every extra value included in the SNMP packet's list represents an additional overhead of 17 bytes *(*Gavalas *et al.,* 2000).

Regarding the health function computation, the daemon agent transfer less number of messages comparing to the SNMP method as shown in Table5. Thus, the total message size is reduced and the bandwidth is saved.

Table 5: Communication Overhead of SNMP and Daemon Agent

	SNMP		Daemon Agent	
	Number of Messages	**Total Message size**	**Number of Messages**	**Total Message size**
Interface Utilization	4	364Byte	1	36Byte
Interface Accuracy	3	275Byte	1	35Byte
IP Discard Rate	5	458Byte	1	40Byte

CHAPTER 5

SUMMARY, CONCLUSION AND RECOMMENDATION

5.1 SUMMARY

Traditional network management based on centralized architecture was observed to have some infeasibility and inefficiency problem when the managed networks are large in scale. MA based approaches have been widely investigated to solve those problems. It has several benefits compared to the centralized system:

a. Distribution of Management Workload

   One of the problems of the centralized system is that most of the work needs to be done at the central manager. It needs to fetch and analyze a lot of the management data. If a large number of NEs are involved, the manager becomes the system bottleneck. MAs could be launched with the task delegated to it, which will automatically finish its job and only return those results to the manager. In this way, the manager is not as loaded and can do more things simultaneously. The other good effect is that many management decision could be taken locally, thus avoiding the transfer of large amount of data back to the manager. As a result, the network bandwidth usage may also be reduced.

b. Adaptability and Flexibility

   Due to its intrinsic mobility and no need for pre-installation, mobile agents represent a promising technology to cope with changing environment and user mobility.

c. Programmability and Customization

Mobile agent could be created and customized according to the user's request, enabling dynamic programmable functionality to be provided. According to, Bohoris *et al.,* (2000), the customization of mobile agent behavior can provide a powerful mechanism for "intelligence on demand". In this thesis, it was outlined how users could use class inheritance to customize basic performance monitoring functions to their needs.

5.2 CONCLUSION

While mobile agent has many superior features over centralized network management approach, it has some performance overhead and other related problems. Due to the agent migration, normally Remote Method Invocation (RMI), serialization and de-serialization, these create some latency, which might be a problem. Security is also a crucial issue of mobile agent, especially in the network management domain where more secure environment needs to be provided.

In conclusion, it is believed that mobile agent, as a promising approach to network management, could provide a new range of opportunities in the future. But a lot of research is still needed to reach an acceptable usage of MA technology in network management domain.

5.3 RECOMMENDATION AND FUTURE WORK

As described in chapter 4, some efforts have been made to make the system more secure. But given the extensive scope of security as it relates to network management as

a whole, a complete software implementation of the system is beyond the scope of this work. A detailed design and implementation of the whole secure framework should be considered as a future work.

The current system implemented is only used for network status monitoring and some performance management applications. Enormous number of potential services are possible in the area of network fault and configuration management. More researchers and manufacturers are interested in combining intelligent agents and mobile agents in the area of proactive fault management. To, allow mobile agent to have some properties of intelligence and self-healing might help the automation of network management.

5.4 REFERENCES

Baldi M., Gai, S. and Picco, G. P. (1997). Exploiting Code Mobility in Decentralized and Flexible Network Management. *Lecture Notes in Computer Science*, volume 1219; PP. 13-26.

Bellifemine, F., Caire, G., Trucco, T., and Rimassa, G. (2010). *JADE Programmer's Guide.* Retrieved January 24, 2011, from http://sharon.cselt.it/projects/jade.

Bellifemine, F., Poggi, A. and Rimassa, G. (2001). Developing multi agent systems with a FIPA-compliant agent framework. *In Software - Practice Experience,* volume 31, PP. 103-128.

Bellifemine, F., Rimassa, G. and Poggi, A. (1999). JADE, A FIPA-Compliant Agent Framework. In: *Proceedings of the 4th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents Technology, United Kingdom.* PP. 1-12

Bieszczad, A., Pagurek, B., and White, T. (1998). Mobile Agents for Network Management. *Communications Surveys & Tutorials, IEEE* Vol 1(1), PP. 2-9

Bivens, A., Fry, P. H., Gao, L., Hulber, M. F. and Szymanski, B.K. (1999). Agent-Based Network Monitoring. In: *Proceedings of the Agent based High Performance Computing Workshop*, *Agent '99 Conference, Seattle; Washington, United State.* PP. 41-53.

Bohoris, C., Pavlou, G., and Cruickshank, H. (2000). Using mobile agents for network performance management. *IEEE/IFIP Network Operations and Management Symposium (NOMS'00), Honolulu, Hawaii, United States*, PP. 637-652.

Caire, G., Bellifemine, F., Trucco, T., and Rimassa, G. (2010). *JADE Administrator's Guide.* Retrieved January 24, 2011, from http://sharon.cselt.it/projects/jade.

Case J., Fedor, M., Schoffstall, M., and Davin, J. (1990) *A Simple Network Management Protocol (SNMP) RFC 1157*. PP. 1-20

Fu, P. (2001). *A Security Architecture for Mobile Agent Systems* (Master of Science Theses, Tsinghua University, China). Retrieved on May 23, 2013 from https://circle.ubc.ca/bitstream/handle/2429/11716/ubc_2001-0381.pdf?sequence=1

Gavalas, D., Greenwood, D., Ghanbari, M. and O'Mahony, M. (1999). An Infrastructure for Distributed and Dynamic Network Management based on Mobile Agent Technology. In: *Proceedings of the IEEE International Conference on Communications (ICC'99),* PP. 1362-1366.

Gavalas, D., Greenwood, D., Ghanbari, M. and O'Mahony M. (2000). Advanced Network Monitoring Applications Based on Mobile/Intelligent Agent Technology. *Computer Communications Journal,* volume 23(8) PP. 720-730.

Griffin D., Pavlou G. and Georgatsos P. (2001). Providing Customisable Network Management Services through Mobile Agents. In: *Proceedings of the 7th International Conference on Intelligence in Services and Networks (IS&N'00), Athens, Greece*. PP. 1-19.

Hia, H.E. (2001). *Secure SNMP-Based Network Management in Low Bandwidth Network*. (Master of Science Theses, Virginia Polytechnic Institute and State University, United States). Retrieved on May 20, 2013 from http://scholar.lib.vt.edu/theses/available/etd-05032001-113550/unrestricted/Erik_Hia_Thesis.PDF.

Paliafito, A. and Tomarchio O. (2000). Using Mobile Agents to implement flexible Network Management Strategies. *Computer Communication Journal*, volume 23(8): PP. 708-719.

Picco, G. P. (1998). *Understanding, Evaluating, Formalizing, and Exploiting Code Mobility*. Doctorate Dissertation, "Politecnico di Torino", Italy). Retrieved on May 20, 2013 from https://cs.ucsb.edu/~vigna/publications/1998_fuggetta _picco_vigna_understanding.pdf

Pinheiro, R., Poylisher, A. and Caldwell, H. (1999). Mobile Agents for Aggregation of Network Management Data. In: *Proceedings of the First International Symposium on Agent Systems*. PP. 130-140.

Poggi, A., Rimassa, G. and Tomaiuolo, M. (2001). Multi-User and Security Support for Multi-Agent Systems. In: *Proceedings of Workshop from Objects to Agents (WOA) 2001 Workshop, Modena*. PP. 1-7

Reiser, H. and Vogt, G. (2000), Security Requirements for Management Systems using Mobile Agents. In: *Proceedings of the Fifth IEEE Symposium on Computers and Communications: (ISCC 2000), Antibes, France*. PP. 3-6.

Rubinstein, M. G., Duarte, M. B. and Pujolle, G. (2002*) Scalability of a Network Management Application Based on Mobile Agents*. Retrieved on February 20, 2011, from http://citeseer.nj.nec.com/ 503561.html.