Agent-based Network Management System

by

Huawen Luo

B.Sc., YunNan University, 1996
M.Eng., Beijing Institute of Technology, 2000

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
Master of Science
in
THE FACULTY OF GRADUATE STUDIES
(Department of Computer Science)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

August 2002

© Huawen Luo, 2002
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
Vancouver, Canada

Date September 30, 2002
Abstract

Contemporary network management systems as represented by Simple Network Management Protocol (SNMP) are based on the client-server centralized paradigm which may lead to inefficiency when the managed networks are large in scale.

Mobile Agent (MA) as an approach has been investigated by many researchers. MA can be used to alleviate the manager workload and reduce the bandwidth usage by delegation of authority from manager to MA. MA could also be instantly customized by user’s requirement. It can be launched from the manager, visiting each network element according to the itinerary table, computing and compressing the management data locally, only returning the result back to the network manager. There are several benefits: distribution of management workload; adaptability and flexibility; programmability and customization.

A mobile agent based network management system has been implemented using the JADE agent platform. The network element’s status monitoring platform is supported and several performance management applications are implemented using mobile agent, such as SNMP table filtering, global filtering. Issues related to security and scalability are also discussed.
Contents

Abstract .......................................................................................................................... ii
Contents ......................................................................................................................... iii
List of Tables .................................................................................................................. v
List of Figures ................................................................................................................ vi
Acknowledgements ....................................................................................................... vii
Dedication ....................................................................................................................... viii

1 Introduction ................................................................................................................... 1
  1.1 Motivation ................................................................................................................. 1
  1.2 Goal .......................................................................................................................... 2
  1.3 Thesis Outline ......................................................................................................... 2

2 Background ................................................................................................................... 4
  2.1 JADE ......................................................................................................................... 4
  2.2 Network Management .............................................................................................. 6
    2.2.1 SNMP ............................................................................................................... 6
    2.2.2 MbD .................................................................................................................. 8
    2.2.3 RMON .............................................................................................................. 9
    2.2.3 MA .................................................................................................................... 9

3 System Design and Implementation ............................................................................ 11
  3.1 System Architecture ............................................................................................... 11
    3.1.1 Network Manager ........................................................................................... 11
    3.1.2 Network Element ............................................................................................ 13
  3.2 Manager Agent ....................................................................................................... 13
4 Related Problems ................................................................. 33
  4.1 Security Problem .......................................................... 33
    4.1.1 Protecting NE from Malicious MA ......................... 34
    4.1.2 Protecting the Important Data in MA ...................... 36
    4.1.3 Secure the Communication Channel ......................... 37
  4.2 Scalability ....................................................................... 38

5 Conclusions and Future Work ............................................. 41

Bibliography .......................................................................... 44
## List of Tables

Table 3.1  MIB II Variables for Computing Interface Utilization ........................................ 21
Table 3.2  MIB II Variables for Computing Interface Error Rate and Accuracy ...................... 22
Table 3.3  Definition of \textit{MIBTreeObjectGet} ACL Message ................................................. 24
Table 4.1  Response Time of SNMP and MA ........................................................................ 39
Table 4.2  Communication Overhead of SNMP and Daemon Agent ....................................... 40
List of Figures

Figure 2.1  Software Architecture of One JADE Agent Platform ........................................... 5
Figure 2.2  The SNMP Architecture ....................................................................................... 7
Figure 3.1  System Architecture ............................................................................................ 12
Figure 3.2  Class Components of Manager Agent ................................................................. 14
Figure 3.3  Registration Example ............................................................................................ 15
Figure 3.4  Behaviours’ scheduling of the Manager Agent ...................................................... 17
Figure 3.5  Code Segment of ManagerNEInMsgBehaviour Class ........................................ 18
Figure 3.6  Communication between Manager and Daemon Agent ....................................... 20
Figure 3.7  MainGui of Manager Agent .................................................................................. 25
Figure 3.8  SNMP Filtering MA Generator GUI ..................................................................... 28
Figure 3.9  SNMP Table Filtering Flow Diagram .................................................................... 29
Figure 3.10 Results Table of SNMP Table Filtering MA ....................................................... 30
Figure 3.11 Example of Global MA Generator GUI .............................................................. 31
Figure 3.12 Global Filtering Flow Diagram ............................................................................ 32
Figure 3.13 Results Table of Global Filtering MA ................................................................. 32
Figure 4.1  Authentication of MA ............................................................................................ 36
Figure 4.2  Encryption of Data Using JCE API ...................................................................... 37
Figure 4.3  Decryption of Data Using JCE API ...................................................................... 37
Acknowledgements

I would like to express my gratitude to my supervisor, Professor Alan Wagner, for his guidance, inspiration and supervision of this thesis. I would also express sincere appreciation to Professor Son Vuong for being my second reader and giving me useful comments for improvement.

Many thanks to all my friends at UBC. Without their generous help, continuous encouragement and moral support, this work could not have been completed.

HUAWEN LUO

The Department of Computer Science
The University of British Columbia
August 2002
To my dearest parents,
I'm nothing without them.
Chapter 1

Introduction

1.1 Motivation

Contemporary network management systems as represented by Simple Network Management Protocol(SNMP) are based on the client-server centralized paradigm, where a central station collects and analyses data retrieved from physically distributed network elements. In those systems, management data are stored in a standard structure maintained on the elements to be managed, such as a Management Information Base(MIB) Objects Tree in SNMP. There is a daemon agent at each network element, such as snmpd running on Linux, which periodically fetches and returns management data in response to inquiries from the manager.

Network management system based on Client/Server paradigm normally requires transferring large amount of management data between the manager and agents. The large amount of data not only requires considerable bandwidth, but also can cause a processing bottleneck at the manager. As current networks grow larger and more complicated, the problem becomes more severe.

The solution to such problem is straightforward; distributing the management mechanism to overcome the limitations of the centralized Client/Server architecture. There are several solutions which have already been put forward, such as Remote Monitoring(RMON) and Management by Delegation(MbD), which introduce some
degree of decentralization. A third solution, the use of Mobile Agent (MA) technology to distribute and delegate management tasks has also been investigated [1][9][10][11][12].

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 that MA have some superior features over SNMP, MbD and RMON [1][13][10]. There is a general agreement that MA can be used to alleviate the manager workload and reduce the bandwidth usage by delegation of authority from the manager to MA. 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, compute and compress 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 transferring of large amounts of data from the remote nodes to the central manager [10].

1.2 Goal

In order to demonstrate the applicability of MA to network management, we investigated the related problems and implemented a network management system based on mobile agents using the JADE agent platform.

Applications implemented in this system are network element’s status monitoring, SNMP table filtering, and global filtering. In this thesis, I will describe the architecture, design and implementation details of those applications. I also investigated the security problem of mobile agent. Security is a crucial problem to the feasibility of mobile agent, especially to the network management domain which has stringent security requirements. Finally, we tested the performance of the system and analyzed potential problems.

1.3 Thesis Outline
In Chapter 2 we give an introduction to some background knowledge related to the thesis, including a brief overview of JADE, SNMP; a brief description of the RMON, MbD and MA. In Chapter 3, we give an overview of the architecture of our system and the detailed design and implementation issues are discussed. The related problems, such as securing the whole system and scalability test is explained in Chapter 4. Finally, the conclusions and possible future work are given in Chapter 5.
Chapter 2

Background

2.1 JADE

Java Agent Development Environment (JADE) is a software framework to aid the development of agent applications in compliance with the Foundation for Intelligent Physical Agents (FIPA) for inter-operable intelligent multi-agent systems [2]. 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 [5]. A JADE platform comprises one or more agent containers, each living in one Java virtual machine (JVM) and providing the execution environment for the agents. There is one Main Container acting as the front end. It has the supervisory control over the JADE platform. The AMS, the DF and the ACC are in the Main Container. The general architecture of one JADE platform is shown in Figure 2.1.
Message passing is used to communicate between agents. The FIPA 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:

1. **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.
2. **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.
3. **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.

### 2.1 Network Management

Network management has been the subject of intense research over the last decade, with the relevant progress being twofold: on the one hand, architectures and algorithms for solving management problems have been devised; on the other hand, different management technologies have been proposed and standardized [11].

Currently, network management systems adopt a centralized paradigm represented by SNMP. Although it is a standard protocol, it has many limitations and inefficiency. Most reasons are rooted in the centralized architecture. The rational approach to overcome it is to distribute the network management operations. There are several ways which have been under investigation, Management by Delegation (MbD), Remote Monitoring (RMON), and Mobile Agent (MA). The following will give brief overviews of those approaches.

#### 2.1.1 SNMP
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.

The architecture of SNMP[8] is shown in Figure 2.2. It defines a client/server relationship.

![SNMP Architecture Diagram](image)

**Figure 2.2: 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 status. The database, controlled by a 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 identifier, or SET the identifier to a new value.

Through the use of private MIB variables, SNMP agents can be tailored for a myriad of specific devices, such as network bridges, gateways, and routers. The definitions of MIB variables supported by a particular agent are incorporated in descriptor files, written in Abstract Syntax Notation (ASN.1) format[7].

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 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[8]. Those shortcoming did not stop the widespread use of it.

2.1.2 MdD

The Management by Delegation (MbD) model was proposed in 1991 to address this problem of difficult to manage centralized systems[20]. 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[20].

2.1.3 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. 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[21].

2.1.4 MA

The emergence of mobile agent frameworks has led many researchers to examine their applicability to network management and control environments. It is believed 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 respective solutions based on traditional approaches[1].
Mobile agent frameworks are currently addressed by two standards bodies. The Federation of Intelligent Physical Agent (FIPA) [2] 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). OMG looks mostly at the issue of mobility according to a standard interoperable framework through its Mobile Agent System Interoperability Facility (MASIF) [5]. In the latter, agent systems model the execution environment able to host mobile agents.
This chapter describes the design and implementation of the agent-based network management system. First, I will describe the system architecture and design. Then, I will present the implementation details. This system is created on top of the JADE agent platform. All the code is written in Java.

3.1 System Architecture

As shown in Figure 3.1, the system consists of two main parts: Network Manager(NM), 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 will show 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.

Figure 3.1: System Architecture
At the NM, users can also use 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 rest of the network to do some domain or global management.

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 dispatchs a daemon agent to the NE which resides at the NE for getting and setting the MIB values.

For the device to be managed, it needs to support SNMP. At its polling station, we also need to deploy the AdventNet package[24] as the SNMP API to connect with the 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 behaviour classes as shown in Figure 3.2.

GUls include *PasswordGUI, RegisterGUI, MainGUI, MobileGeneratorGUI1* and *MobileGeneratorGUI2* which are used by NM to view the network. The behaviour classes include *GetavailableLocationBehaviour, ManagerNEInMsgBehaviour, PingBehaviour*, and *AgentRegisterBehaviour.*
3.2.1 Communication of those components

Manager Agent implements its GUIs by extending the 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()[4].

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.
• Receiving a Register Message

RegisterBehaviour is responsible for waiting for the coming register message sent by the NE. After receiving a message, it fetchs 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 `moretti.cs.ubc.ca`) 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. The Manager Agent sends daemon agent NEAgent2(there is a NE `lindemans.cs.ubc.ca` already registered and NEAgent1 has been sent to it) to `moretti.cs.ubc.ca` and updates the list in the RegisterGUI as shown in Figure 3.3. “Ok” means the registered NE is running fine. It will show “DOWN” whenever the manager finds the NE is down.

![Register GUI for NE Agents](image)

**Figure 3.3: Registration Example**
Click on the View Monitor Panel Button on RegisterGui

At the registerGui shown in Figure 3.3, the 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 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 tasks 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 3.4.

\[
\text{RegisterBehaviour} \rightarrow \text{ManagerNEInMsgBehaviour} \rightarrow \text{PingBehaviour 1} \rightarrow \text{GetAvailableLocationBehaviour} \leftarrow \text{PingBehaviour} N \ldots \leftarrow
\]

**Round-robin Non-preemptive Scheduler**

Figure 3.4: Behaviours’ scheduling of the Manager Agent

### 3.2.2.2 Implementing Agent Behaviour 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 the use of endless loops and performing 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 3.4 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 Figure 3.5, the problem can be solved by calling to receive just one kind of message at a time. While in the interval, it gives other behaviours, such as PingBehaviour, the chance to execute.
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 ml = MessageTemplate.MatchPerformative(ACLMessage.INFORM);
        MessageTemplate m2 = MessageTemplate.MatchLanguage("PlainText");
        MessageTemplate m3 = MessageTemplate.MatchOntology("performanceIndicators");
        MessageTemplate m1andm2 = MessageTemplate.and(ml,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 3.5: 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, we could use a mobile agent 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 we need to monitor the status of the NE and obtain feedback in real time. Mobile Agents could be launched anytime as required. However, if it is not sent out in time, we may miss detecting a fault. It may cause more problems if not handled as soon as possible. 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 potential problems that may have occurred or will occur.

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

![Diagram of communication between Manager and Daemon Agent]

**Figure 3.6: 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)[14]. There are several important health functions computed by the daemon agent.

- **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[14].

The primary measure used for network utilization is interface utilization. The following formulas is 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 listen 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[14].

For half duplex media, use the following formula for interface utilization, the meaning of each parameter is shown in Table 3.1.

\[
\frac{(ifInOctets + ifOutOctets) \times 8 \times 100}{ifSpeed \times SysUpTime}
\]

<table>
<thead>
<tr>
<th>NOTATION</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>IfInOctets</td>
<td>Represents the count of inbound octects of traffic</td>
</tr>
<tr>
<td>IfOutOctets</td>
<td>Represents the count of outbound octects of traffic</td>
</tr>
<tr>
<td>IfSpeed</td>
<td>The speed of the interface</td>
</tr>
<tr>
<td>SysUpTime</td>
<td>The time duration after the system start up</td>
</tr>
</tbody>
</table>

Table 3.1: MIB II Variables for Computing Interface Utilization
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.

\[
\frac{\text{max}(\text{ifInOctets}, \text{ifOutOctets}) \times 8 \times 100}{\text{ifSpeed} \times \text{SysUpTime}}
\]

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.

- **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[14]. 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[14]:

- Out-of-specification wiring
- Electrical interface
- Faulty hardware or software

The following variables are used in accuracy and error rate formulas:

<table>
<thead>
<tr>
<th>NOTATION</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>IfInErrors</td>
<td>Represents the count of inbound packets with an error</td>
</tr>
<tr>
<td>IfInUcastPkts</td>
<td>Represents the count of inbound unicast packets</td>
</tr>
<tr>
<td>IfInNUcastPkts</td>
<td>Represents the count of inbound non-unicast packets</td>
</tr>
<tr>
<td>(multicast and broadcast)</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: MIB II Variables for Computing Interface Error Rate and Accuracy
The formula for error rate is usually expressed as a percentage:

\[
\text{Error Rate} = \frac{\Delta \text{ifInErrors} \times 100}{(\Delta \text{ifInUcastPkts} + \Delta \text{ifInNUcastPkts})}
\]

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

\[
\text{Accuracy} = \frac{100 - \Delta \text{ifInErrors} \times 100}{(\Delta \text{ifInUcastPkts} + \Delta \text{ifInNUcastPkts})}
\]

- **IP output datagrams discard rate**

The discard rate defines the percentage \( E(t) \) 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:

\[
\frac{(\text{ipOutDiscards} + \text{ipOutNoRoutes} + \text{ipFragFails}) \times 100}{\text{IpOutRequests} + \text{ipForwDatagrams}}
\]

As described above, the health functions are computed by the 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 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:

- Register and Unregister
For example, the ACL Message for MIBTreeObjectGet is defined in Table 3.3:

<table>
<thead>
<tr>
<th>Language</th>
<th>PlainText</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ontology</strong></td>
<td>MIBTreeObjectGet</td>
</tr>
<tr>
<td>Content (Manager→NEAgent)</td>
<td>.1.3.6.1.2.1.2.1.0</td>
</tr>
<tr>
<td>Content (NEAgent→Manager)</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table 3.3: Definition of MIBTreeObjectGet ACL Message**

### 3.6 Example Communication of Manager Agent with Daemon Agent

As an example, Figure 3.7 shows the Monitor Panel (MainGui) of host moretti.cs.ubc.ca. 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 3.7 shows.

![Figure 3.7: MainGui of Manager Agent](image)

### 3.7 Mobile Agent (MA)

We have implemented two type of mobile agents in the system to do some intelligent filtering applications. The MA can be created through the 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* [1]. The design and implementation is based on the JADE Agent Enviroment.
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 `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, e.g. 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, we need to wait for its response before we can go on to the next GET operation. But for each GET, we can only receive one table row. 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 acquisition of each individual object value leads to potential inconsistencies[1].

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 get. 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 have the potential to improve the retrieval of SNMP tables in terms of network overhead. MA could be launched to NE where we wish to retrieve SNMP values. Using the successive GET-NEXT request, MA gets the snapshot of the table, encrypts it 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 we can significantly reduce the latency and bandwidth needed to obtain the result. 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 the user through the MA Generator GUI. For example, Figure 3.8 shows one customized MA Generator GUI.
As shown in the Figure 3.8, 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 Figure 3.7 is to fetch the ifTable at moretti.cs.ubc.ca, lindemans.cs.ubc.ca 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. Figure 3.9 shows the flow diagram of SNMP table filtering by MA.

Figure 3.9: SNMP Table filtering Flow Diagram
The results for the example SNMP table filtering by MA is listed in the table GUI shown in Figure 3.10.

![GUI of MA Table](image)

**Figure 3.10: 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 just have been fetched. In this level, MA could bring more benefits. Not only the manager relieved from handling heavy processing tasks, but also the MA's state size is prevented from growing rapidly. Right now, in the implementation, we only support domain/global filtering based on the following health functions: *InputUtilization*, *OutputUtilization*, *IfInErrorRate*, *IfAccuracy*, *IPOutputDiscardRate*. For example, we may want to fetch the two more heavily loaded interface, the results will 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 3.11 shows one example of this type of agent generator.
Figure 3.11: Example of Global MA Generator GUI

As the example shows, we 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 flow diagram for global filtering is shown in Figure 3.12.
Arrive at destination and start execution

Get the SNMP table
Set totalcount to 0

Filter according to the filtering expression

Totalcount < Maxtotalcount?

no

Yes

Sort and compare And keep only the Maxtotalcount line

Encrypt the result

Migrate to the next host

Figure 3.12: Global Filtering Flow Diagram

The results brought back by global filtering MA customized in Figure 3.11 is listed in the table GUI shown in Figure 3.13.

<table>
<thead>
<tr>
<th>Host Name</th>
<th>IfAccuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>moretti.cs.ubc.ca</td>
<td>100.0</td>
</tr>
<tr>
<td>lindemans.cs.ubc.ca</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Figure 3.13: Results Table of Global Filtering MA
Chapter 4

Related Problems

The current prototype has demonstrated the usefulness of the properties provided by MA to help NM to do network management. In this Chapter, we discuss some related problems that have arisen during the work. There are also several interesting future directions that could be explored as an extension of this work. First, we discuss the security problem. Second, we address the scalability problem of this system.

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 evesdropped and improperly set by malicious users. 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[6][8]. In the system
that I developed, 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 secure 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 a malicious MA; Second, security threats by a malicious host; Third, security threats from network communication[22]. Considering these three kinds of threats, we tried 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. In the current version of JADE, there is no security mechanism at this point. The following is the design to secure the MA in this system.

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, are allowed to be instantiated. The RSA algorithm, based on the public-private pair of keys paradigms could provide both authentication and encryption features. We designed a way to authenticate the MA and make sure the data integrity of the MA. It uses certificates and digital signatures. In digital signature, instead of encrypting the data
itself, the signing software creates a one-way hash of the data, then uses the private key to encrypt the hash[17]. 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 associate that identity with a public key[17]. It is like a passport to identify a person. A certificate is normally issued by Certificate authorities(CAs).

The following describes the authentication of MA for the network management system. Here we assume that the communication channel has already been secured through the use of the Secure Socket Layer(SSL) connection as described in the later section. As well we assume the manager has received the certificate from the CAs. Figure 4.1 shows the authentication step:

1. 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.
2. Send the certificate and digital signature along with the MA across the network.
3. After NE container gets the MA, it retrieves the public keys from the certificate and try 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 be instantiated and further executed.
4. Travel to the next NE according to the itinerary table.
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. I choose the Java Cryptography Enviroment (JCE) API[16] 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.

Figure 4.1 Authentication of MA
In the filtering mobile agent, after it fetchs 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 returning back to the manager. At the last destination, the message decrypts and shows the result in the table GUI.

In Figure 4.2 and Figure 4.3, the encryption and decryption of a message in the daemon agent are illustrated:

```java
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 4.2: Encryption of Data Using JCE API

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

Figure 4.3: Decryption of Data Using JCE API

4.1.3 Secure the Communication Channel

Although we made an effort 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 although he may not able to read the content because of the encryption. So, we need to secure the communication channel.

In JADE, the communication is normally through RMI. RMI is a distributed object broker above TCP/IP socket. To make it secure, we only need to secure the TCP/IP socket. We could use SSL. The SSL protocol runs above TCP/IP and below higher-level protocols such as HTTP or IMAP. It uses TCP/IP on behalf of the higher-level protocols, and in the process allows a 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. The job is not difficult, we just replace the standard socket with the SSL sockets. The version of RMI included in Java 2 SDK enables the RMI developer to use custom socket types. All an application has to do is to install the RMI socket factory that creates sockets of the desired type (SSL socket)[19].

4.2 Scalability

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

The topology used on this experiment consists of one management station (sleeman.cs.ubc.ca) and two managed NEs (moretti.cs.ubc.ca, salvator.cs.ubc.ca) 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, we alternately repeat the two elements, e.g., if we want to manage 5 elements, we use an itinerary {moretti, salvator, moretti, salvator, moretti, sleeman}. 

38
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, i.e. 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.

The following table listed the testing result:

<table>
<thead>
<tr>
<th></th>
<th>Centralized SNMP Approach</th>
<th>Mobile Agent Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 host</td>
<td>0.7 Second</td>
<td>0.9 Second</td>
</tr>
<tr>
<td>2 host</td>
<td>0.91 Second</td>
<td>1.447 Second</td>
</tr>
<tr>
<td>4 host</td>
<td>1.3 Second</td>
<td>1.81 Second</td>
</tr>
<tr>
<td>10 host</td>
<td>1.68 Second</td>
<td>3.51 Second</td>
</tr>
<tr>
<td>20 host</td>
<td>3.01 Second</td>
<td>6.62 Second</td>
</tr>
<tr>
<td>50 host</td>
<td>8.01 Second</td>
<td>16.62 Second</td>
</tr>
</tbody>
</table>

Table 4.1: Response Time of SNMP and MA

As we can see from the table 4.1, 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. This is demonstrated by other graduate students using the WAVE, a mobile agent platform written in C code, to do some similar migrating and fetching data. The speed is much faster than JADE.

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 varbind list represents an additional overhead of 17 bytes[1].

Regarding the health function computation, the daemon agent transfers less number of messages comparing to the SNMP method as shown in table 4.2. Thus, the total message size is reduced and the bandwidth is saved.

<table>
<thead>
<tr>
<th></th>
<th>SNMP</th>
<th>Daemon Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Messages</td>
<td>Total Message size</td>
<td>Number of Messages</td>
</tr>
<tr>
<td>Interface Utilization</td>
<td>4</td>
<td>364Byte</td>
</tr>
<tr>
<td>Interface Accuracy</td>
<td>3</td>
<td>275Byte</td>
</tr>
<tr>
<td>IP Discard Rate</td>
<td>5</td>
<td>458Byte</td>
</tr>
</tbody>
</table>

Table 4.2: Communication Overhead of SNMP and Daemon Agent
Chapter 5

Conclusions and Future Work

1. Conclusions

Traditional network management based on centralized architecture has the 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 comparing to the centralized system:

- **Distribution of Management Workload**

  One of the problems of the centralized system is 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 could be taken locally, thus avoiding transfer large amount of data back to the manager. As a result, the network bandwidth usage may also be reduced.

- **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.
• **Programmability and Customization**

Mobile agent could be created and customized according to the user's request, enabling dynamic programmable functionality to be provided. The customization of mobile agent behavior can provide a powerful mechanism for "intelligence on demand"[11]. In this thesis, we have shown how users can use class inheritance to customize basic performance monitoring functions to their needs.

On the other hand, while mobile agent has many superior features over centralized one, it has some performance overhead and other related problems. Due to the agent migration, normally remote method invocation, serialization and deserialization, latency might be a problem. Security is also a crucial issue of mobile agent, especially in the network management domain where a more secure environment needs to be provided.

In summary, we believe 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.

### 2. Future Work

As described in chapter 4, we have made some effort to make the system more secure. But given the extensive scope of the topic, a complete software implementation of the system is beyond the scope of this thesis. A detailed design and implementation of the whole secure framework should be considered as future works.

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. Allow mobile agent to have some properties of intelligence and self-healing might help the automation of network management.
Bibliography


[17] Introduction to Public-key Cryptography


