

# MOBILE AGENT BASED NETWORK MANAGEMENT

## A DISSERTATION

*Submitted in partial fulfillment of the  
requirements for the award of the degree*

*of*

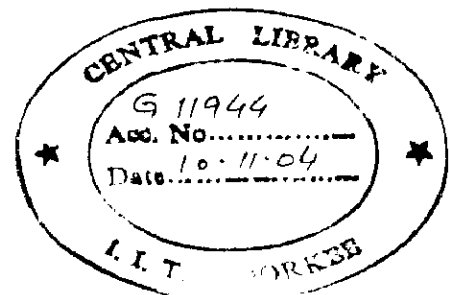
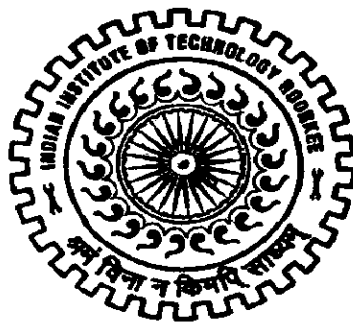
MASTER OF TECHNOLOGY

*in*

INFORMATION TECHNOLOGY

*By*

**MAHEEDHAR VALASA**



**IIT Roorkee - CDAC, NOIDA,  
c-56/1, "Anusandhan Bhawan"  
Sector 62, Noida-201 307**

**JUNE, 2004**

## **CANDIDATE'S DECLARATION**

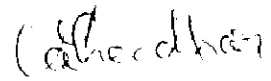
---

I hereby declare that the work presented in this dissertation titled "**MOBILE AGENT BASED NETWORK MANAGEMENT**", in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Information Technology**, submitted in **IIT-Roorkee - CDAC campus, Noida**, is an authentic record of my own work carried out during the period from June 2003 to June 2004 under the guidance of **Dr. Poonam Rani Gupta**, Associate Professor, CDAC, Noida.

I have not submitted the matter embodied in this dissertation for the award of any other degree or diploma.

Date: 28/06/2004

Place: Noida

  
(**MAHEEDHAR VALASA**)

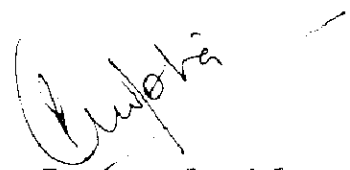
---

## **CERTIFICATE**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 28/06/2004

Place: Noida

  
(**Dr. Poonam Rani Gupta**)  
**Associate Professor,**  
**CDAC, Noida.**

## ACKNOWLEDGEMENTS

---

I hereby take the privilege to express my deep sense of gratitude to **Prof. PREM VRAT**, Director, Indian Institute of Technology, Roorkee, and **Mr. R.K.VERMA**, Executive Director, CDAC, Noida for providing me with the valuable opportunity to carry out this work. I am very grateful to **Prof. A.K.AWASTI**, Programme Director, **Prof. R.P. AGARWAL**, course coordinator, M.Tech (IT), IIT, Roorkee and **Mr. V.N.SHUKLA**, course coordinator, M.Tech (IT), CDAC, NOIDA for providing the best of the facilities for the completion of this work and constant encouragement towards the goal.

I express my sincere thanks and gratitude to my Guide **Dr. POONAM RANI GUPTA**, Associate Professor, CDAC, Noida, for her inspiring guidance and sustained interest throughout the progress of this dissertation.

I am thankful to **Mr. R.K.SINGH** and **Mr. MUNISH KUMAR**, project engineer, CDAC Noida, for providing necessary infrastructure to complete the dissertation in time.

I owe special thanks to my friends, all of my classmates and other friends who have helped me formulate my ideas and have been a constant support. I also thank my parents and other family members for their moral support.

  
(**MAHEEDHAR VALASA**)

Enroll. No. 029011

# CONTENTS

---

<b>Candidate's Declaration</b>	<b>i</b>
<b>Acknowledgements</b>	<b>ii</b>
<b>Abstract</b>	<b>vi</b>
<b>1. INTRODUCTION</b>	<b>1</b>
1.1 Limitations of SNMP	2
1.2 Distributed Network Management	3
1.3 Dissertation Objective	3
1.4 Report Organization	4
<b>2. LITERATURE SURVEY</b>	<b>5</b>
2.1 Mobile Agent - Definition	5
2.2 Background on Mobility	6
2.3 Advantages of Mobile Agents	9
2.4 Java and Mobile Agents	10
2.5 Mobile Agents and Security	11
2.6 A Survey of Mobile Agent Systems	15
<b>3. ANALYSIS</b>	<b>17</b>
3.1 Integration of Mobile Agents and SNMP	17
3.2 MANM Hybrid Model	18
3.2.1 Management Application	20
3.2.2 Mobile Agent Execution Environment	20
3.2.3 Mobile Agent Producer	20
3.2.4 Mobile Agents	20
3.3 Mobile Agent Infrastructure	20
3.4 AgentSpace – Mobile Agent System	22
3.4.1 Mobile Agent Platform	22
3.4.2 Agent Migration	22
3.4.3 Agent Cloning	23
3.4.4 Agent Caching	23
3.4.5 AgentSpace Client-Server Architecture	23

3.4.6 Object Model	26
3.4.7 The Agent Pattern in AgentSpace	27
3.5 Simple Network Management Protocol	28
3.5.1 SNMP Management Information	29
3.5.2 Lexicographical Ordering	37
3.5.3 Protocol Specification	37
3.5.4 Get Request PDU	41
3.5.5 GetNext Request PDU	42
3.5.6 Set Request PDU	42
3.5.7 Trap PDU	43
<b>4. DESIGN</b>	<b>45</b>
4.1 Mobile Agent Life Cycle	45
4.2 Mobile Agent Traveling Patterns	46
4.2.1 Itinerary Model	47
4.2.2 Broadcast Model	47
4.3 AgentSpace Design Issues	47
<b>5. IMPLEMENTATION</b>	<b>53</b>
5.1 Mobile Agents – Programmers Perspective	53
5.2 Network Management Features	56
5.2.1 Itinerary Model	57
5.2.2 Broadcast Model	57
5.2.3 Active Network-Management	58
5.2.4 Trap Generation	59
5.2.5 Runtime Decision-Making	60
5.2.6 Dynamic Service Provisioning	61
<b>6. RESULTS AND DISCUSSIONS</b>	<b>63</b>
<b>7. CONCLUSIONS</b>	<b>69</b>
<b>REFERENCES</b>	

## LIST OF FIGURES

<b>Figure No.</b>	<b>Name</b>	<b>Page No.</b>
Figure 3.1	MANM Architecture	19
Figure 3.2	Mobile Agent Infrastructure	21
Figure 3.3	AgentSpace – Mobile Agent System	22
Figure 3.4	AgentSpace Client-Server Architecture	23
Figure 3.5	Structure of AgentSpace’s Agent pattern – Collaboration diagram	27
Figure 3.6	SNMPv1 Architecture	29
Figure 3.7	MIB-2 Object groups	31
Figure 3.8	SNMP Message Formats	38
Figure 4.1	Mobile Agent Life Cycle	45
Figure 4.2	Interactions between Mobile Agent and SNMP Agent at Device	46
Figure 4.3	Itinerary Model	47
Figure 4.4	Broadcast Model	47
Figure 4.5	Generic Structure of Agent pattern – Class diagram	48
Figure 4.6	Agent’s main groups of methods	49
Figure 4.7	Generic interaction of the Agent pattern – Scenario diagram	51
Figure 6.1	AgentSpace Server	63
Figure 6.2	AS-Client Applet	64
Figure 6.3	AS-Client’s Features	64
Figure 6.4	SNMP Manager GUI	65
Figure 6.5	MANM Manager GUI	65
Figure 6.6	Itinerary Model Results	66
Figure 6.7	Broadcast Model Results	66
Figure 6.8	Performance Management GUI	66
Figure 6.9	Interface Utilization Management GUI	67
Figure 6.10	Results of Active Network-Management	67
Figure 6.11	Trap Generator GUI	68
Figure 6.12	Dynamic FTP Client Service Provisioning	68

## ABSTRACT

---

Network management fundamentally involves monitoring and controlling the devices connected in a network by collecting and analyzing data from them. Majority of present day network management systems operate SNMP (Simple Network Management Protocol). Although the centralized management approach gives network administrators a flexibility of managing the whole network from a single place, it is prone to information bottleneck and excessive processing load at the manager and heavy usage of network bandwidth. Mobile agent based network management overcomes these limitations by distributing the network management functionality. Mobile agents refer plainly to self-contained and identifiable computer programs that can move within the network and act on behalf of the user or another entity. These agents move to the place where data are stored and collect the information the manager wants. The idea of a self-controlled program execution near the data source has been proposed as the next wave to replace the client-server paradigm as a better, more efficient and flexible mode of communication. The mobile agent paradigm proposes to treat the network as multiple agent friendly environments and the agents as programmatic entities that perform tasks for users. Agents can function independent of each other or cooperate to solve problems. Mobile agent based network management is to equip agents with network management capabilities and allow them to issue requests to managed devices after migrating to their place. This work investigates the advantages of using mobile agent (distributed) as opposed to conventional (client-server or SNMP) approach to network management. To exploit the potential of mobile agent technology it also demonstrates the application of two traveling patterns for the mobile agent and their use. Further it incorporates new features like trap generation, active network-management and runtime decision-making to considerably increase the performance of mobile agent based approach to network management.

# INTRODUCTION

---

Managing large networks with hundreds of computers has become a challenging and tedious task for today's network administrators. A typical computing infrastructure in a medium to large-scale organization contains many nodes, possibly of different kinds, organized into multiple local-area networks and administrative domains. So the need for efficient tools, techniques and technologies for managing IP networks appears more substantive than ever before and increased intelligence in management solutions is becoming a major requirement for the communications networks in the Internet era. Today's Network Management Systems (NMS) are mostly based on centralized manager-agent model, where a host who acts as the manager is taking care of a set of nodes by collecting, analyzing and configuring the appropriate management parameters.

Following some early protocols such as the Simple Gateway Monitoring Protocol (SGMP), the Simple Network Management Protocol (SNMP) [13,30] gained acceptance as the management protocol of choice for IP networks. SNMP provides a set of services that allows SNMP managers interact with SNMP agents that provide access to management information of the managed equipment (and/or software). Through this information, a central manager, or managers provide detailed and/or enterprise wide views of a network. This management information is referenced through a construct called a Management Information Base (MIB) [14].

SNMP was designed to be an application-level protocol that is part of the TCP/IP protocol suite. It is intended to operate over the User Datagram Protocol (UDP). For a stand-alone management station, a manager process controls access to a central MIB at the management station and provides an interface to the network manager. The manager process achieves network management by using SNMP, which is implemented on top of UDP, IP, and the relevant network-dependent protocols (e.g., Ethernet, FDDI, and X.25). SNMP supports the operations: *Get-Request*, *Get Next-Request*, *Get-Response*, *Set-Request* and *Trap*. The first four operations are used to obtain/set object values. For each of the *Get-Request*, *Get Next-Request*, *Set-Request* operations there will be a *Get-*



*Response* operation. A *Trap* generated by an SNMP Agent is an asynchronous operation that enables the management station of a significant event in a device. These operations allow SNMP managers access to management information of the managed equipment.

From its original beginnings as a "simple" solution to the challenges involved in managing the operation of networking equipment, SNMP continues to evolve in support of increasingly sophisticated requirements. Through several steps, SNMP has evolved into three major versions, the original Simple Network Management Protocol, enhanced SNMPv2 and SNMPv3.

## **1.1 Limitations of SNMP**

While extremely useful as a tool in network management, concerns have been expressed in the use of SNMP for managing large networks. These deficiencies include:

1. SNMP may not be suitable for the management of truly large networks because of the performance limitations of polling. With SNMP, you must send one packet out to get back one packet of information. This type of polling results in large volumes of routine messages and yields problem response times that may be unacceptable.
2. SNMP is not well suited for retrieving large volumes of data, such as an entire routing table.
3. SNMP traps are unacknowledged. In the typical case where UDP/IP is used to deliver trap messages, the agent cannot be sure that a critical message has reached the management station.
4. The basic SNMP standard provides only trivial authentication. Thus, basic SNMP is better suited for monitoring than control.
5. SNMP does not directly support imperative commands. The only way to trigger an event at an agent is indirectly, by setting an object value. This is a less flexible and powerful scheme than one that would allow some sort of remote procedure call, with parameters, conditions, status, and results to be reported.
6. The SNMP MIB model is limited and does not readily support applications that makes sophisticated management queries based on object values or types.

7. SNMP does not support manager-to-manager communications. For example, there is no mechanism that allows a management system to learn about the devices and networks managed by another management system.

## **1.2 Distributed Network Management**

The most suitable solution for the above-mentioned limitations is to distribute [7] the management intelligence by bringing it as close as possible to the managed network elements. This approach has appeared firstly in the so-called Management by Delegation (Mbd) [15] research initiative. Other related attempts are Remote Monitoring (RMON) [30], and the proxy agent paradigm specified in the SNMPv2 [30]. Recently the Mobile Agent technology [2,11] has emerged as a promising solution towards implementing strategies that distribute and automate management tasks. The Mobile Agent technology provides an innovative software interaction paradigm that allows code migration between hosts for remote execution and is a rapidly developing area of research in the fields of distributed network management. This does not imply that the large system is merely divided into smaller pieces. Instead several centralized applications each capable of addressing a certain aspect of a problem are tied together with a communication system. It would allow for exchange of viewpoints and coming up with strategies to make progress or to combine the results into a solution. Each type of cooperating systems may be considered as an agent. The mobile agent paradigm proposes to treat the network as multiple agent friendly environments and the agents as programmatic entities that perform tasks for users. Agents can function independent of each other or cooperate to solve problems.

## **1.3 Dissertation Objective**

The purpose of this study is to investigate the use of mobile agent based approach to network management in overcoming the limitations of conventional network management protocols. Further to provide a solution for generating traps in a mobile agent based network management environment and introduce new features like active network-management, runtime decision-making to make the approach more efficient.

## **1.4 Report Organization**

In chapter 2 definition of mobile agent is given with background on mobility and the advantages of using java language for developing mobile agent applications. It also discusses different security issues to be considered and a survey of existing mobile agent systems. Chapter 3 covers the details of integrating mobile agents with SNMP Agents and explains the infrastructure required to execute mobile agents with an e.g. AgentSpace system used in the work. It also describes the MANM model and gives the details of SNMP protocol. The fourth chapter discusses how the mobile agent life cycle is designed and the execution process of the mobile agent at the managed device. It also explains the AgentSpace pattern and design issues. Fifth chapter explains how to create mobile agent based applications with AgentSpace system and the different features implemented in this work. Chapter 6 of the report includes the GUIs built and results obtained. Chapter 7 concludes the report mentioning how MANM approach has overcome the limitations mentioned in chapter 1 and gives further areas of improvement.

# LITERATURE SURVEY

---

---

## 2.1 Mobile Agent – Definition

A Mobile Agent (MA) [1,2,6,21] can be defined as a software program capable of migrating autonomously from node to node and traverse the network carrying logic and data, performing specialized tasks on behalf of their creators working independently or in conjunction with other software components.

The term *mobile agent* contains two separate and distinct concepts: mobility and agency. A software agent is a computational entity, which acts on behalf of others, is autonomous, proactive, reactive, and exhibits capabilities to learn, cooperate, and move. This is called as a *basic agent model*. A *mobile agent* is a software agent (which is characterized by *basic agent model*) that can move between locations. In addition to the basic model, any software agent defines a *life-cycle model*, a *computational model*, a *security model*, and a *communication model*. A navigation model additionally characterizes a MA. Most research examples of the MA paradigm as reported in the literatures currently have two general goals: reduction of network traffic and asynchronous interaction.

To make use of MAs, a system has to incorporate a mobility framework. The framework has to provide facilities that support all of the agent models, including the navigation model. For the life-cycle model, we need services to create, destroy, start, suspend, stop, etc. agents. The computational model refers to the computational capabilities of an agent, which include data manipulation and thread control primitives. The security model describes the ways in which agents can access network resources, as well as the ways of accessing the internals of the agents from the network. The communication model defines the communication between agents and between an agent and other entities (e.g. the network). Navigation model handles all issues referring to transporting an agent (with or without its state) between two computational entities residing in different locations.

The size of MAs depends on what they do. In swarm intelligence the agents are very small. On the other hand, configuration or diagnostic agents might get quite big, because they need to encode complex algorithms or reasoning engines. However, agents can extend their capabilities on the fly, on-site by downloading required code off the network. They can carry only the minimum functionality, which can grow depending on the local environment and needs. This capability is facilitated by code mobility.

## 2.2 Background on Mobility [32]

MAs can be implemented using one of the two fundamental technologies mobile code or remote objects. The ability to start a computation on a site, suspend the execution of the computation at some point, migrate the computation to a remote site and resume its execution there is called *mobile communications*. Sometimes called *MAs* it is a far more complex problem than simple moving objects around as done in RMI or CORBA, or moving code alone as with java applets. When an agent migrates, its state is extracted from the source agent system and transferred to its destination where it is restored into a new instance of an agent object. During transfer, only site-independent information is transferred. In the case of communicating channels, this information consists of the agent names with which the migrating agent had opened channels as well as their current location. The state relevant to each particular node is transient, i.e., it is discarded. For example the sockets maintained in the agent control object are closed and then reopened in the remote agent control object.

A distinction is usually made between *strong migration* and *weak migration*. Strong migration means migrating a process by sending its memory image to a remote site: the current state of the stack, the value of the program counter, and of course all the objects reachable from the process. Moreover, the migration may occur preemptively; the process does not need to know it has migrated. On the other hand, weak migration usually only involves the objects reachable from the process, and requires that the process has agreed to migrate (it is non preemptive). But there exists no implementation of strong migration in Java that does not break the Java model or require user instrumentation of the code.

The best-known Java libraries for MAs are Aglets [16,26] and Voyager [17]. Both implement a form of weak migration, in the sense that an object must explicitly invoke a primitive in order to migrate, and this can only take place at points in the execution where a check pointed state (either implicit or explicit) is reached. A *checkpoint* is a place in the code where the current state of the computation can be safely saved and restored.

For example, the *MoveTo* primitive of Voyager waits until all threads have completed. On the other hand, in Ajents [18], any object can be migrated while executing by interrupting its execution, moving the most recently check-pointed state of the object to a remote site and re-executing the method call using the check-pointed object state. Apart from the fact that whether check pointing occurs lies in the hands of the user program, general and well-known issues related to checkpoint inconsistency due to rollback are kept unresolved.

Systems also differ in the way mobile objects interact: either by remote method call (Ajents, Voyager), or using a message-centric approach (Aglets). The default interaction mode is usually synchronous, even if some form of asynchronous communication is sometimes provided (in Aglets or Ajents for instance). Interacting synchronously simplifies the overall management of migration: while a remote method or message is handled, nothing else can happen to the two partners (and especially no migration), but this of course incurs a performance penalty.

Remote interaction is achieved transparently by a proxy, which hides the effective location of the destination object to the caller. The proxy also often acts as a forwarder for locating the mobile object and is a convenient place for performing security-related actions.

In this section two of the most popular solutions to the *location problem* are given. The first solution is based on a *location server*, chain of *forwarders*. Other solutions exist as the one described in [19] that uses a distributed two-phase transaction and a sophisticated reference-tracking mechanism.

**Location Server.** The location server responsibility is to track the location of each mobile active object: Every time an active object migrates, it sends its new location to the location server it belongs to. As a result of the active object leaving a host, all the references

pointing to its previous location become invalid. There exists different strategies for updating those dangling references with the new location, one of them being

**Lazy:** when an object tries to send a message to a mobile active object using a reference that is no longer valid, the call fails and a mechanism transparently queries the location server for the new location of the active object, updates the reference accordingly and re-issues the call.

Being a centralized solution, it is very sensitive to network or hardware failures. Standard techniques for fault-tolerance in distributed systems could be put to use here, such as using a hierarchy of possibly replicated location servers instead of a single server. Nevertheless, this solution is costly, difficult to administer and would certainly not scale well.

**Forwarders:** An alternative solution is to use *forwarders*: knowing the actual location of a mobile object is not needed in order to communicate with it; rather, what really matters is to make sure that the mobile object will receive the message we send to it.

To do so, a chain of references is built, each element of the chain being a *forwarder* object left by the mobile object when it leaves a host and that points to the next location of the mobile object [17]. When a message is sent, it follows the chain until it reaches the actual mobile object. This appears to be the solution of choice for several systems [20].

Forwarders can be considered as a distributed solution to the location problem, as opposed to the previous centralized solution. Moreover, contrary to the location server, in the absence of network or host failure, a message will finally reach any mobile object even if it never stops migrating. However, this solution also suffers from a number of drawbacks.

First, some elements of the chain may become temporarily or permanently unreachable because of a network partition isolating some elements from the rest of the chain or just because a single machine in the chain goes down; it would destroy all the forwarder located on it. The chain is then broken, and it becomes impossible to communicate with the mobile object at the end of the chain, although it is still well and alive.

## 2.3 Advantages of Mobile Agents

There are many advantages with the use of MAs in network management. They are

1. Efficiency savings - CPU consumption is limited, because a MA executes only on one node at a time. Other nodes do not run an agent until needed.
2. Space savings - Resource consumption is limited, because a MA resides only on one node at a time. In contrast, static multiple servers require duplication of functionality at every location. MAs carry the functionality with them, so it does not have to be duplicated. Remote objects provide similar benefits, but the costs of the middleware might be high.
3. Reduction in network traffic - Code is very often smaller than data that it processes, so the transfer of MAs to the sources of data creates less traffic than transferring the data. Remote objects can help in some cases, but they also involve marshalling of parameters, which may be large.
4. Asynchronous autonomous interaction - MAs can be delegated to perform certain tasks even if the delegating entity does not remain active.
5. Interaction with real-time systems - Installing a MA close to a real-time system may prevent delays caused by network congestion. In Network Management systems NM agents usually reside close to the hardware, so this advantage might not be as clear as others.
6. Robustness and fault tolerance - If a distributed system starts to malfunction, then MAs can be used to increase availability of certain services in the concerned areas. For example, the density of fault detecting or repairing agents can be increased. Some kind of meta-level management of agents is required to ensure that the agent-based system fulfills its purpose.
7. Supports for heterogeneous environments - MAs are separated from the hosts by the mobility framework. If the framework is in place, agents can target any system. The costs of running a Java Virtual Machine (JVM) on a device are decreasing. Java chips will probably dominate in the future, but the underlying technology is also evolving in the direction of ever-smaller footprints (e.g. Jini).



8. On-line extensibility of services - MAs can be used to extend capabilities of applications, for example, providing services. This allows for building systems that are extremely flexible.
9. Convenient development paradigm - Creating distributed systems based on MAs is relatively easy. The difficult part is the mobility framework, but when it is in place, then creating applications is facilitated. High-level, rapid application development (RAD) environments for agents will be needed when the field matures. It is quite probable that the flourishing tools for object-oriented programming will evolve into agent-oriented development environments, which will include some functionality to facilitate agent mobility.
10. Easy software upgrades - A MA can be exchanged virtually at will. In contrast, swapping functionality of servers is complicated; especially, if we want to maintain the appropriate level of quality of service (QoS).

Although this new management approach seems to solve the problems of centralized management architectures, one cannot neglect the strengths of existing management solutions based on "legacy protocols" such as SNMP. SNMP is an accepted standard, and is likely to be the network management protocol of choice for the foreseeable future and many network nodes will be equipped with an SNMP agent. Even if MAs were being employed for new tasks, it would be reasonable to take advantage of the existing capabilities of the resident SNMP agent.

## **2.4 Java and Mobile Agents [31]**

Java is the predominant language for MA systems, both for implementing MA execution environments and for writing MA applications. The proliferation of the Java programming language led to the development of numerous MA platforms. Actually, Java seems perfect for developing an execution environment for MAs, because Java offers many features that ease its implementation and deployment. Java runtime systems are available for, most hardware platforms and operating systems. Therefore, MA platforms that are built on Java are highly portable and run seamlessly on heterogeneous systems. Furthermore, MAs profit from continuous performance and scalability enhancements, such

as increasingly sophisticated compilation techniques and other optimizations, which are provided by the underlying Java Virtual Machine (JVM).

In addition to portable code, Java offers a *serialization* mechanism allowing to capture a MA's object instance graph before it migrates to a different host, and to resurrect the agent in the new environment. Java also supports dynamic loading and linking of code by means of a hierarchy of *class loaders*. A class loader constitutes a separate *name space* that can be used to isolate classes of the AS and of different agents from each other.

In general, MA platforms execute multiple agents and service components concurrently in a time-sharing fashion. Java caters for this need by means of multi-threading. Java is also a *safe* language, which means that the execution of programs proceeds strictly according to the language semantics. For instance, types are not misinterpreted and data is not mistaken for executable code. The safety properties of Java depend on techniques such as *bytecode verification*, *strong typing*, *automatic memory management*, *dynamic bound checks*, and *exception handlers*. On top of that, the Java platform includes a sophisticated security model with flexible access control based on dynamic stack introspection.

In summary, Java is highly portable and provides easy code mobility. This caused numerous MA systems based on Java being developed and experimented with.

## **2.5 Mobile Agents and Security [10]**

Although MAs have many benefits for distributed computing they introduce a new dimension of security issues. Automatically executing arbitrary code any host can be dangerous. The same care is necessary as if manually starting programs from unknown sources. In order to protect hosts from malicious code, agent systems usually provide a virtual machine or interpreter to run MAs in a separate and locked environment. Any action or communication of MAs is then only possible through the means of agent system (AS).

### 2.5.1 Analysis of Threats

Various kinds of attacks and threats could compromise the security of MA based management systems. An attack is an attempt to illegally access a system, a resource or information or to execute malicious code. Attacks are classified as active and passive attacks. The ability of an attacker to change something is characteristic for an active attack. In a passive attack he/she only collects information but does not to actively manipulate an object. A target of an attack can be any entity (AS, manager, MA, managed resource) in a MA based management system as well as any relation (information channel between two entities, a *communication relation* can be considered as transport of messages, *execution relation* between AS and agent when an AS is executing an agent, *calling relation* when an agent requests for services at AS). Therefore threats are distinguished from **attacking an entity** from **attacking a relation**.

The three kinds of relations communication, execution and calling must be distinguished between attacks which are generally possible for all kinds of relations and those which are special to a particular kind of attack.

#### 2.5.1.1 Entity Attacks

As pointed above there are four main entities, which can act as subjects as well as objects: managers, MAs, AS, managed resources. Each of these entities can be attacked. The attacker tries to become a 'valid identity' by faking an identity or entity in the management system. This attack is called masquerade, e.g. if the attacker can act as a manager of if he/she can launch a MA or AS under the name of a legitimate subject he is able to gain illegal access to the system.

#### 2.5.1.2 Relation Attacks

In addition to entity attacks, there are also attacks to relations between two legitimate entities. Relations between more than two entities can be split into several two-entity relations. Some of them apply to relations in general and some of them are specific to a particular kind of relation. Eavesdropping of messages can enable the attacker to gain information paving the way for further attacks or to steal confidential data. This is a passive attack and very hard to detect.

It is necessary to identify the user, which is responsible for the message or action. For example, it must be impossible to launch a MA doing malicious actions and, afterwards, repudiate everything. This relation attack is called repudiation. Another attack in this regard is the unauthorized replication of MAs. A malicious MA, AS or manager may replicate MAs. Besides, an intruder in a relation may duplicate a MA or message during transmission.

If the attacker can actively manipulate the information channel he/she can do alterations to messages. In this case, he/she may change the functionality or data of a migrating agent. The AS is a mediator between MAs and hosting systems. In addition, it provides a runtime environment for MAs. Therefore, a malicious AS can read, alter or delete data of a local MA (alteration of code and data).

An attacker can do a denial-of-service attack against communication relations, an AS or a hosting system, e.g. a hostile MA overloads the attacked resource and thus it is impossible for other legitimate subjects to use the resource. This scenario is even more complicated if the denial-of-service attack is not done by a single MA by a distributed group of malicious MAs.

Another attack is resource misuse. As an MA implements management functionality and must therefore have administrator rights. The MA can abuse communication resources, resources of the underlying host system or of the AS.

Despite of these general attacks there is one, which only affects the *calling relation*. The attacker can try to circumvent the dedicated calling interfaces to directly access other methods not intended to be used. Also the *communication relation* is security sensitive. An attacker may store a message or an MA and send it once more at a later time to a destination. This is called replay attack. Moreover, an attacker can also redirect agents and messages or delay them.

The last kind of relation attack is that against *execution relations*. As an MA can only live with the aid of an AS it is even feasible for a hostile agent system to manipulate the execution trace of an MA. For example, the AS can manipulate the runtime stack of the MA, prevent execution of a certain function or force execution of additional functionality. Another possible attack is to prevent execution of MAs (denial-of-

execution). As an AS has to execute the MA and thus has complete control of the agent, these attacks are almost impossible to prevent. For this reason, we either assume a relationship of trust between delegator/MA can attack the AS, the underlying hosting system or other MAs in various ways (e.g. denial-of-service, resource misuse).

### 2.5.2 Security Requirements

Regarding the various attacks it is possible to develop a defense strategy for each kind of attack. But this approach has the drawback that any new attack requires a new defense strategy and the security system always 'lags behind' the attacker. The more promising approach is to develop a security architecture, which implements a more abstract view on attacks. The OSI security architecture may be regarded as a basis, but it must be adapted to particular characteristics of agent systems. The first step towards such architecture is to deduce a conceptual view on counteractions against classes of attacks: security requirements. In order to satisfy these requirements several components and services have to be identified and integrated in security architecture for a mobile agent based management system. Such architecture is able to prevent complete classes of attacks and even future attacks belonging to one of these classes.

The security requirement is authentication. MAs are new kind of access to systems that need closer attention. Authentication is very fundamental, because most of the following security requirements presuppose the ability to identify subjects and objects unambiguously;

Authentication is necessary to bind rights to subjects. For that purpose rights and permissions must be described. Access control must then enforce rights and restrictions at runtime. Each object in the system offers interfaces, which can be used by subjects. Access control prevents illegal access of objects. Certain management tasks require that a mobile agents is able to delegate rights and permissions to other entities, a concept for delegation of these rights is necessary. Security management with the aid of mobile agents can be carried out if such a concept is available.

Each information channel representing a relation between entities may need protection. The security requirement confidentiality is satisfied if such a channel is only accessible by authorized participants.

The aim of a lot of attacks is to alter code, data or messages or to replay/replicate messages or MAs. Detecting such alterations, manipulations, replays and misordering can assure the integrity of objects. Being able to establish and enforce resource constraints can prevent another big group of attacks: resource abuse and denial-of-service. The security requirement non-repudiation means that it is possible to prove that a certain subject has done a critical or sensitive action. Even a third party can prove who caused this action.

To prevent the circumvention of legal interfaces and to restrict rights the sandboxing concept is used. A sandbox is a very restricted environment for code execution, which only can be left in a controlled manner.

Some attacks (e.g. manipulating an MA by an AS) seem very hard or even impossible to be prevented. If it is not possible to restrain these attacks technically an organizational solution is necessary, e.g. a trust relation between two entities that a particular kind of attack will not happen.

## **2.6 A Survey of Mobile Agent Systems**

There are several research activities that are related to mobile agent (MA) technology and many more centering on MA issues. In general, there are three targets for MA system design and implementation: using or creating a specialized language, as operating system (OS) services or extensions, or as application software. In the first approach, language features provide the requirements of MA systems. The second approach implements MA system requirements as OS extensions to take advantage of existing OS features. Lastly, the third approach builds MA systems as specialized application software that runs on top of an OS to provide MA functionalities. For comparative purposes, nine current projects [21] are chosen.

- Aglet™ from IBM
- Agent Tcl from Dartmouth College
- Agents for Remote Access (ARA) from the University of Kaiserslautern
- Concordia™ from Horizon Systems Laboratory, Mitsubishi Company
- Mole from the Institute for Parallel and Distributed Computer Systems (IPVR)
- Odyssey™ from General Magic

- TACOMA from Cornell University
- Voyager™ from ObjectSpace
- Secure and High Performance Mobile Agent Infrastructure (SHIP-MAI) from the Multimedia and Mobile Agent Research Laboratory, University of Ottawa

Table 2.1 gives the list of major mobile agent systems and the way their features are incorporated.

Mobile Agent	Security	Portability	Mobility	Communication	Resource management
Aglet	Limited, sandbox model	Java	Aglet transfer protocol	Event, message object	Java
Agent Tcl	Limited, sandbox model	Support multiple language interpreters	Multiple protocol	RFC	Yes
ARA	Limited, sandbox model	Support multiple language interpreters	Multiple protocol	RFC	Yes
Concordia	Limited, sandbox model and secure channel	Java	Socket and java serialization	Event, group	Yes, via the queue server
Mole	Basic java	Java	Enhanced java model with code server	Event	Java
Odyssey	Basic java	Java	Java RMI, CORBA, IIOP DCOM	Event	Java
TACOMA	Limited, User firewall agent	None	TCP	Folder Object	Operating system
SHIP-MAI	Sandbox model, secure channel, policy access control	Java	Java, java object serialization	Event, group room object, java syntax for method call	Planned
Voyager	Limited, sandbox model, secure channel	Java	Java object serialization, reflection	Distributed event (voyager space), java syntax for method call	Java

**Table 2.1 Survey of mobile agent systems**

# ANALYSIS

---

### 3.1 Integration of Mobile Agents and SNMP

As said earlier that the mobile agent based network management must utilize the advantages of the conventional SNMP already in use. While integrating the mobile agent based approach with SNMP the possible interactions [4] between a mobile agent arrived at a device and the SNMP Agent residing at the device will be

- a) A mobile agent may arrive at a node and wish to access data, such as interface statistics, from the resident SNMP agent's MIB. That is, it wants to Get or perhaps even Set data in the SNMP MIB. One could equip the mobile agent with SNMP managerial capabilities and allow the mobile agent, acting as a manager, to issue SNMP request packets to the resident SNMP agent. Of course the mobile agent would not have to be co-resident to be able to do this, but for security reasons alone, it is not a good idea to allow a mobile agent to open a socket to communicate with the SNMP agent. A better way would be to provide a local SNMP service or a carefully controlled secure interface to the nodes resources for this purpose. There would still need to be an application program interface (API) or a mechanism that would give the mobile agent a way to make its wishes known to the intermediary service. Such an approach can be adopted using readily available Java SNMP classes. The advantage of such an approach is that one does not have to modify the SNMP agent to provide access to its MIB. The cost of this is that the mobile agent would need to have full SNMP managerial capabilities and would have to handle such tasks as BER encoding. Another approach is to design a lightweight protocol that would take advantage of the co-residency of the mobile and SNMP agents.
- b) A mobile agent may arrive at a node and have the capability of extending the existing SNMP MIB with certain meta-variables that it brings with it. That is the



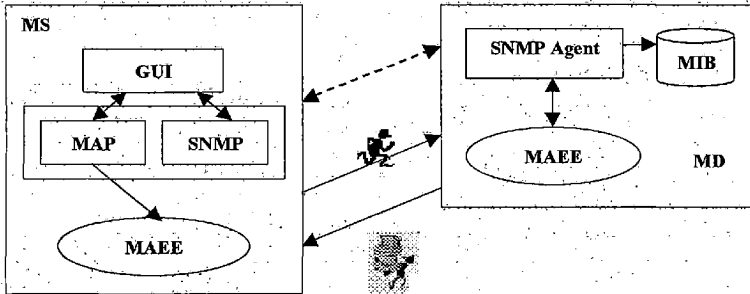
mobile agent may bring with it some state, some information from the outside network at large, that it was charged with to explore and analyze. A manager would have to know of the existence of such additional information to be able to take advantage of it, but presumably it delegated the mobile agent to acquire such information in the first place and would have defined it as a MIB sub-tree. The SNMP DPI protocol [22] and its successor the AgentX protocol [23] were designed specifically to allow a sub-agent process to extend or enhance an existing SNMP agent by first registering with it, and then communicating with it using the specified protocol. Clearly the SNMP agent would have to have the protocol built in, but many agents today are DPI enabled and it appears that many future agents will have the improved AgentX protocol incorporated. Clearly a mobile agent could behave as an AgentX sub-agent, or better, would employ an API of a local service that would securely accomplish the registration and pass queries back and forth. In either case, the manager would need to have predefined the new MIB variable to be dynamically added to the local SNMP MIB.

- c) A mobile agent, having acquired information during its migratory activity, may arrive at a node that consequently implies a fault or degraded performance either at that node or else where in the network, and wish to send off a defined SNMP trap to a remote SNMP manager.
- d) Finally, it is conceivable that some entity would like to send an SNMP trap to a mobile agent, which is acting in the role of an SNMP manager. This assumes that the sender can actually locate the mobile "manager" using the mobile agent environment's agent location service.

### **3.2 MANM Hybrid Model**

To take the advantages of mobile agents for network management, a flexible architecture is designed, in which Mobile Agent based Network management (MANM) forms a layer over conventional SNMP based management. This ensures that the advantages of SNMP are not lost and also serves the purpose of managing legacy SNMP based systems.

As shown in figure 3.1 MANM framework [1,3] is a hybrid model, which has features of mobile agents as well as SNMP. MANM gives the manager the flexibility of using SNMP model or mobile agent based management depending on the management activity that is involved. This architecture has many advantages over the existing architectures.



- MS – Management Station
- MAP – Mobile Agent Producer
- SNMP – Simple Network Management Protocol
- MAEE – Mobile Agent Execution Environment
- MN – Managed Device
- MIB – Management Information Base

**Figure 3.1 MANM Architecture**

In this approach the MANM station assumes responsibilities of a client. All managed nodes are servers, which have mobile agent execution environment and respond to SNMP queries from mobile agents when they visit the servers and manipulate data locally. When the client in the MANM needs access to data on a network-connected device, it does not talk directly to the server over the network. Instead, the client actually dispatches a mobile agent to the server's machine. Once on the server's machine, the MA makes its requests to the server directly. When the entire transaction is complete, the mobile agent returns to the management station with the results.

The MANM provides Java-compliant interfaces to network management services. The MANM framework consists of the following major components:

### **3.2.1 Management Application**

The management application has a Graphical User Interface (GUI), which coordinates with the agent applications underneath it. It interacts with Mobile Agent Producer (MAP) in configuring a MA with details such as the parameters to be evaluated at the managed node's site and health functions.

### **3.2.2 Mobile Agent Execution Environment (MAEE)**

MAEE is an execution environment for the execution of MA's. MAEE could be characterized as home for mobile agents from where they could execute their duties. The agent comes to the managed node from the management station, executes its management task and goes back to the management station. Server acts as a mobile agent execution environment in machines that host mobile agents.

### **3.2.3 Mobile Agent Producer (MAP)**

MAP could be characterized as a tool for generating customized MA's that are equipped according to the requirements of network manager. By using MAP the functional characteristics of MA's, which roam in the network to collect information from managed nodes, can be changed dynamically (i.e. at runtime). Dynamic creation and configuration of MA's is achieved using MAP.

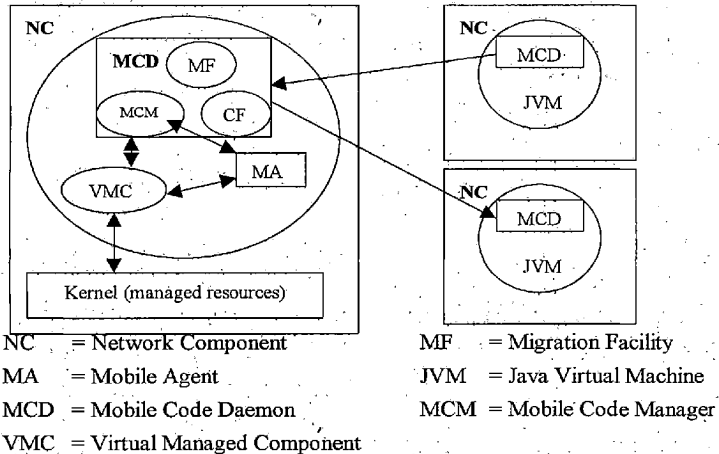
### **3.2.4 Mobile Agents (MA)**

A typical mobile agent is Autonomous, Mobile, Persistent, Communicative/collaborative and active/proactive. The ability to travel allows mobile agents to move to the network element, which is to be managed. In other words, mobility of MA's could be exploited to transfer the MA to managed node and interact locally with the SNMP agent on the managed node.

## **3.3 Mobile Agent Infrastructure**

In order to provide the mobile agents with an execution environment (MAEE) each network component (management station or managed device) should have a mobile agent infrastructure [6,8] that provides the capabilities, which are characterized by the

mobile agents as shown in figure 3.2. Different models like life-cycle model, computational model, security model, communication model and navigation model characterize the definition and behavior [6] of mobile agent. In order to perform network management functionality using mobile agent, it is necessary to have an infrastructure that provides a framework for code mobility and mobile agent execution.



**Figure 3.2 Mobile Agent Infrastructure**

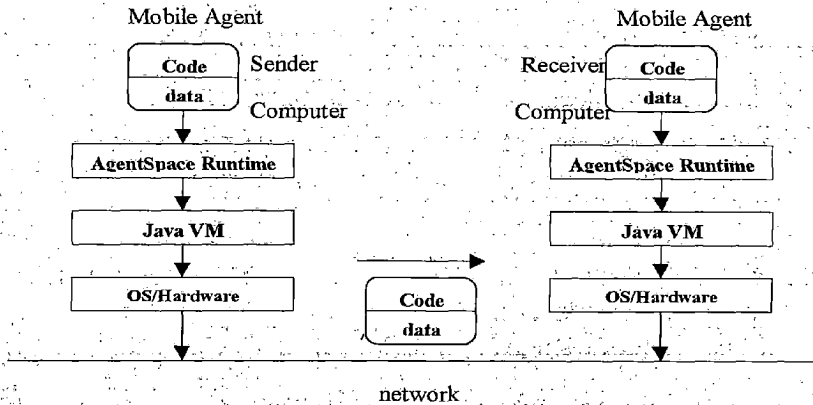
Figure 3.2 explains the development of Mobile Agent Infrastructure. Every network component (NC, either management station or device) contains a Mobile Code Daemon (MCD) running within a Java Virtual Machine (JVM). The MCD provides a number of services that facilitates the execution of mobile agents:

1. A Mobile Code Manager (MCM) that manages the life cycle of a mobile agent from its arrival and authentication at the network component to its migration or perhaps destruction.
2. A Migration Facilitator (MF) to transport mobile agents between NCs.
3. A Communication Facilitator (CF) for collaboration between local and remote mobile agents.

- An interface called the Virtual Managed Component (VMC), which provides for mobile agents accessing the NC's managed objects and resources in a controlled and secure way. The VMC [4] is responsible for management of the mobile agents access rights and the allocation of resources to that agents.

### 3.4 AgentSpace - Mobile Agent System

Figure 3.3 gives the architecture of AgentSpace system and way code and data are moved between devices in the network.



**Figure 3.3 AgentSpace – Mobile Agent System**

#### 3.4.1 Mobile Agent Platform

The runtime system is introduced as a platform for mobile agents. It can create/destroy mobile agents, and send/receive mobile agents to/from a runtime system running on another computer. Also, it is characterized in being designed for a distribution mechanism of network protocols and applications.

#### 3.4.2 Agent Migration

The runtime system permits the migration of not only the code but also the values of the instance variables included in the agent. Hence, after migrating the agent, the values are restored in the agent again, and then its execution starts from a given method.

### 3.4.3 Agent Cloning

The runtime system offers a mechanism to create a copy of an existing agent including all instance variables. The cloned agent has the same state as the original agent has, but its identity is different from that of the original one. If the original agent has a reference to resources, the runtime system protects the resources appropriately.

### 3.4.4 Agent Caching

The runtime system can load the code of each agent on demand and cache it in order to improve performance when particular protocols are sometimes used. The cache is managed in a least used order. When the state of an agent arrives at a remote node, the runtime system on the remote node checks a cache of codes. If the required code is not found at the cache, it sends a load request to the previous node or certain code base nodes. Furthermore, the runtime system ensures that mobile agents can be automatically and dynamically transferred to the nodes, which are needed.

### 3.4.5 AgentSpace Client-Server Architecture

AgentSpace's main goals are the support, development and management of Agent Based Applications (ABA). These goals are provided through three separated but well-integrated components as depicted in figure 3.4.

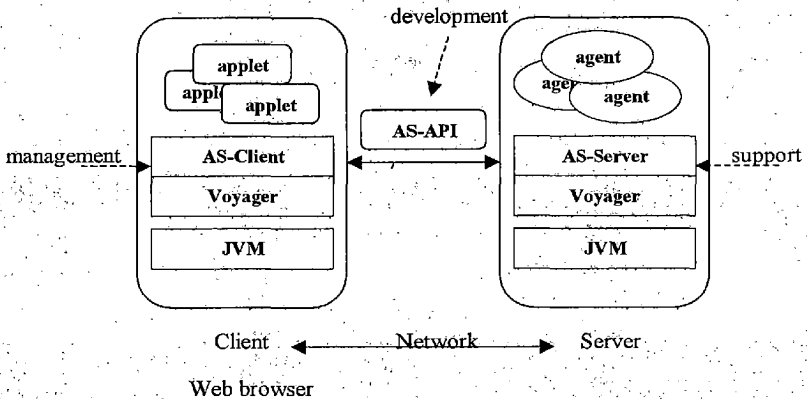


Figure 3.4 AgentSpace Client-Server Architecture

Both server and client AgentSpace [29] components run on top of Voyager and Java Virtual Machine (JVM). They can execute in the same or in different machines. As depicted in figure 3.4, the agents run always on an AS-Server's context. On the other hand, they interact with their end-users through (specific or generic) applets running in some Web browser's context.

#### **3.4.5.1 The AgentSpace Server (AS-Server)**

AS-Server is a Java multithreaded process in which agents can be executed. The AS-Server provides several services, namely:

1. Agent and place creation;
2. Agent execution;
3. Access control;
4. Agent persistency;
5. Agent mobility;
6. Generation of unique identities (UID);
7. Support for agent communication; and
8. Optionally a simple interface to manage/monitor itself.

#### **3.4.5.2 The AgentSpace Client (AS-Client)**

AS-Client supports - depending on the corresponding user access level - the management and monitoring of agents and related resources. The AS-Client is a set of Java applets stored on an AS-Server's machine in order to provide an adequate integration with the Web, offering Internet users the possibility to easily manage their own agents remotely. Furthermore, the AS-Client should be able to access several AS-Servers, providing a convenient trade-off between integration and independence between these two components.

### **3.4.5.3 Voyager**

Voyager [17] is a robust, extensible, scalable solution for enterprise-distributed development. Based on established ORB technology, Voyager's layered and modular architecture transparently supports multiple messaging protocols (IIOP, RMI, SOAP, DCOM), naming protocols (RMI Naming, CORBA Naming, JNDI), communications protocols (TCP/IP, SSL, SOCKS), and messaging patterns (synchronous, one-way, delayed-synchronous, asynchronous). It is equipped with advanced features including:

1. Publish/subscribe
2. Dynamic aggregation
3. Object mobility
4. Federated, distributed naming service
5. Management console
6. Dynamic, automated proxy class generation
7. Remote class loading

### **3.4.5.4 AgentSpace Application Programming Interface (AS-API)**

AS-API is a package of Java interfaces and classes that defines the rules to build agents. In particular, the AS-API supports the programmer when building:

1. Agent classes and their instances (agents) that are created and stored in the AS-Server's database for later use; and
2. Client applets (that are stored in the AS-Server's file system or in the AS-Server's database) in order to provide an interface to agents.

These clients/applets can be either generic mini-applications - such as the AS-Client itself, see above - or specific to some particular agent, for example, to input data or present a report.



### 3.4.6 Object Model

AgentSpace involves the support, development and management of several related objects: contexts, places, agents, users, groups of users, permissions, ACLs (access control lists), security managers, tickets, messages, and identities.

The **context** is the most important and critical object of the AS-Server, as each AS-Server is represented by one context. The context contains the major data structures and code to support the AS-Server, such as lists of places, users, groups of users, meta-agent classes and access control lists.

Each context has a number of places. The **execution place**, or simply **place**, has mainly two objectives. First is to provide a conceptual and programming metaphor where agents are executed and meet other agents. Second, to provide a consistent way to define and control access levels, and to control computational resources.

The place has a unique global identity and knows the identification of its owner/manager. It also maintains a keyword/value list that allows an informal characterization. Optionally, places can be hierarchically organized. The place can also contain the maximum and current number of agents allowed in order to support some resource management. In order to keep track of its agents, the place keeps a list containing its visitant agents and another with its native agents. The place also knows in which place its native agents are executing at a given point of time.

The **agent** is the basic element of the system. Agents are identified by a unique global identity. Agents have two parts:

1. A visible component, that should be developed, or specialized, by programmers and
2. An invisible component, called "internal-agent", kept by AgentSpace.

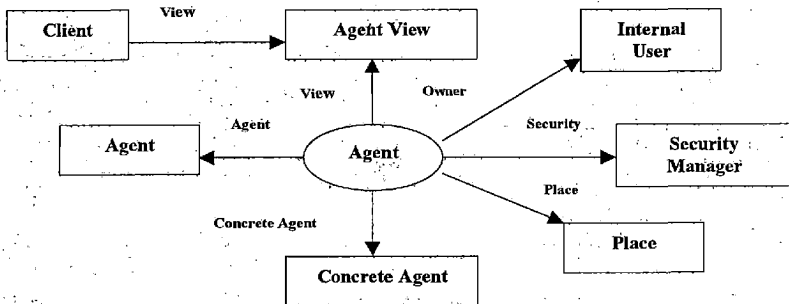
Agents are active objects that execute in some AS-Server, but from a conceptual perspective, they are currently in some place. Agents can navigate to other (local or remote) place if they have permission to do it. Just one **user** owns an agent. Nevertheless,

other users (or even agents from other users) might interact with it, if this is granted by the agent's security policy.

The AS-Server also maintains lists of **users**, **groups of users** and **acl** to implement the permission and access control mechanism. A user may belong to one or more groups. Groups may be hierarchically organized to simplify permission management. This means that all users of some specialized group have implicitly all the permissions they inherit from the more general groups. By default, every AS-Server defines four groups of users and establishes a convenient security access policy, based on them: anonymous group; end-users group; place owners group; and AS-Server's administrators group.

### 3.4.7 The Agent Pattern in AgentSpace

Figure 3.5 shows the specific structure of the Agent pattern [33] related to the AgentSpace framework. Due to the fact that AgentSpace has been developed on top of the Voyager infrastructure, the persistence and distributed details of the Agent pattern are transparently supported by an internal class (i.e., not visible from the programmer's perspective), which is called *InternalAgent*.



**Figure 3.5 Structure of AgentSpace's Agent Pattern – Collaboration Diagram**

It is important to note how suitable, to support dynamic and distributed applications, the process of creating agents (as well as places) can be. Firstly, there is no use or explicit reference to network-enable classes. Secondly, all agents are created

through a factory method (i.e., the `createAgent` method) in a transparent, clean and easy way.

Thirdly, `AgentSpace` provides a very extensible and elegant way to handle security policies/strategies related to the access and interactions between agents and end-users, and between agents themselves. Basically, one security policy/strategy (i.e., a security manager class) is attached to the agent object at its creation. In `AgentSpace` the `SecurityManager` is an abstract class from which other classes should be derived. By default, every agent is attached to the `DefaultAgentSM` class. However, other classes – other agent's security policies – can be defined and used by the system through the class loading and reflection mechanisms of Java.

Another novel aspect of `AgentSpace` is the well-integrated association between users and agents/places. This mechanism, intrinsic by default in `AgentSpace`, provides an easy and clean way to develop and manage this class of applications.

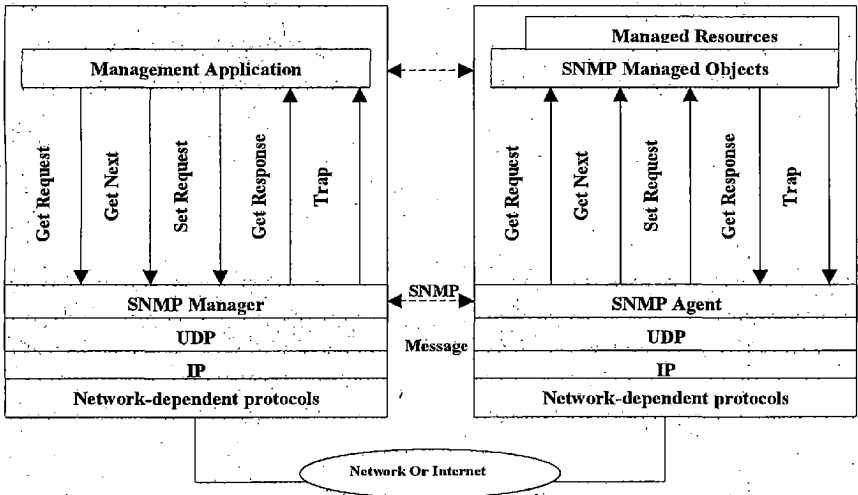
### 3.5 Simple Network Management Protocol

The term Simple Network Management Protocol is actually used to refer to a collection of specifications for network management that include the protocol itself, the definition of data structures, and associated concepts. The three foundation specifications that define SNMP are

- I. Structure and Identification of Management Information for TCP/IP-based networks [24]: describes how managed objects contained in the MIB are defined.
- II. Management Information Base for Network Management of TCP/IP-based Internets: MIB-2 [14]: describes the managed objects contained in the MIB.
- III. Simple Network Management Protocol [13]: defines the protocol used to manage these objects.

Figure 3.6 provides a closer look at the protocol context of SNMP. From a management station, three types of SNMP messages are issued on behalf of a management application: *GetRequest*, *GetNextRequest*, and *SetRequest*. The first two are variations of the *get* function. All three messages are acknowledged by the agent in the form of a

*GetResponse* message, which is passed up to the management application. In addition, an agent may issue a trap message in response to an event that affects the MIB and the underlying managed resources.



**Figure 3.6 SNMPv1 Architecture**

Because SNMP relies on UDP, a connectionless protocol, SNMP itself is connectionless. No ongoing connections are maintained between a management station and its agents. Instead, each exchange is a separate transaction between a management station and an agent.

**3.5.1 SNMP Management Information**

As with any network management system, the foundation of a TCP/IP-based network management system is a database containing information about the elements to be managed. In both the TCP/IP and the OSI environments, the database is referred to as a management information base (MIB). Each resource to be managed is represented by an object. The MIB is a structured collection of such objects. For SNMP, the MIB is, in essence, a database structure in the form of a tree. Each system (workstation, server,

router, bridge, etc.) in a network or internetwork maintains a MIB that reflects the status of the managed resources at that system. A network management entity can monitor the resources at that system by reading the values of objects in the MIB and may control the system resources at that system by modifying those values.

In order for the MIB to serve the needs of a network management system, it must meet certain objectives.

- *The object or objects used to represent a particular resource must be the same at each system.*
- *A common scheme for representation must be used to support interoperability.*

The second point is addressed by defining a structure of management information (SMI).

### **3.5.1.1 Structure of Management Information**

The structure of management information (SMI), which is specified in RFC 1155, defines the general framework within which a MIB can be defined and constructed. The SMI identifies the data types that can be used in the MIB and specifies how resources within the MIB are represented and named. The philosophy behind SMI is to encourage simplicity and extensibility within the MIB. Thus, the MIB can store only simple data types: scalars and two-dimensional arrays of scalars. SNMP can retrieve only scalars, including individual entries in a table. The SMI does not support the creation or retrieval of complex data structures.

To provide a standardized way of representing management information, the SMI must do the following.

- ✓ Provide a standardized technique for defining the structure of a particular MIB.
- ✓ Provide a standardized technique for defining individual objects, including the syntax and value of each object.
- ✓ Provide a standardized technique for encoding object values.

### 3.5.1.2 MIB Structure

All managed objects in the SNMP environment are arranged in a hierarchical or tree structure. The leaf objects of the tree are the actual managed objects, each of which represents some resource, activity, or related information that is to be managed. The tree structure itself defines a grouping of objects into logically related sets.

Associated with each type of object in a MIB is an identifier of the ASN.1 type OBJECT IDENTIFIER. The identifier serves to name the object. In addition, because the value associated with the type OBJECT IDENTIFIER is hierarchical, the naming convention also serves to identify the structure of object types.

The Object identifier is a unique identifier for a particular object type. Its value consists of a sequence of integers. The set of defined objects has a tree structure, with the root of the tree being the object referring to the ASN.1 standard.

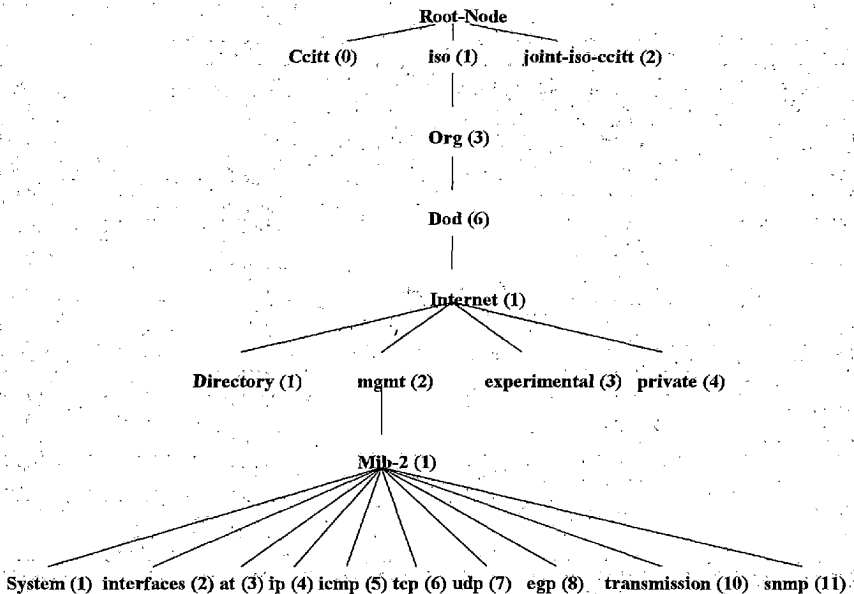


Figure 3.7 MIB-2 Object groups

Beginning with the root of the object identifier tree, each object identifier component value identifies an arc in the tree. Starting from the root, there are three nodes at the first level: *iso*, *ccitt*, and *joint-iso-ccitt*. Under the *iso* node, one subtree is for the use of other organizations, one of which is the U.S. Department of Defense (*dod*). RFC 1155 makes the assumption that one subtree under *dod* will be allocated for administration by the Internet Activities Board as follows:

Internet OBJECT IDENTIFIER: = {iso (1) org (3) dod (6) 1}

This is illustrated in Figure 3.7. Thus, the *internet* node has the object identifier value of 1.3.6.1. This value serves as the prefix for the nodes at the next lower level of the tree.

As shown in figure 3.7, the SMI document defines four nodes under the *internet* node:

1. *directory*: reserved for future use with the OSI directory
2. *mgmt*: used for objects defined in IAB-approved documents
3. *experimental*: used to identify objects used in Internet experiments
4. *private*: used to identify objects defined unilaterally

The *mgmt* subtree contains the definitions of management information bases that have been approved by the IAB. At present, two versions of the MIB have been developed, *mib-1* and *mib-2*. The second MIB is an extension of the first. Both are provided with the same object identifier in the subtree since only one of the MIBs would be present in any configuration.

Additional objects can be defined for a MIB in one of three ways:

1. The *mib-2* subtree can be expanded or replaced by a completely new revision (presumably *mib-3*). To expand *mib-2*, a new subtree is defined.
2. An experimental MIB can be constructed for a particular application. Such objects may subsequently be moved to the *mgmt* subtree.
3. Private extensions can be added to the *private* subtree. One that is documented an RFC is the MUX MIB (RFC 1227).

### 3.5.1.3 Object Syntax

Every object within an SNMP MIB is defined in a formal way; the definition specifies the data type of the object, its allowable forms and value ranges, and its relationship to other objects within the MIB. The ASN.1 notation is used to define each individual object and also to define the entire MIB structure. In keeping with the objective of simplicity, only a restricted subset of the elements and features of ASN.1 are used.

The object identifier is a unique identifier of an object, consisting of a sequence of integers, known as subidentifiers. The sequence, read from left to right, defines the location of the object in the MIB tree structure. For example object ID for tcpConnTable is shown below.

iso	org	dod	internet	mgmt	mib-2	tcp	tcpConnTable
1	3	6	1	2	1	6	13

This identifier would normally be written as 1.3.6.1.2.1.6.13.

### 3.5.1.4 Encoding

Objects in the MIB are encoded using the basic encoding rules (BER) associated with ASN.1. While not the most compact or efficient form of encoding, BER is a widely used, standardized encoding scheme.

It is not possible to change the structure of a MIB by adding or deleting object instances (e.g., adding or deleting a row of a table). Nor is it possible to issue commands for an action to be performed. Further, access is provided only to *leaf* objects in the object identifier tree. That is, it is not possible to access an entire table or a row of a table with one atomic action. These restrictions greatly simplify the implementation of SNMP. On the other hand, they limit the capability of the network management system.

### 3.5.1.5 Communities and Community Names

Network management can be viewed as a distributed application. Like other distributed applications, network management involves the interaction of a number of application entities supported by an application protocol. In the case of SNMP network



management, the application entities are the management station applications and the managed station (agent) applications that use SNMP, which is the supporting protocol.

SNMP network management has several characteristics not typical of all distributed applications. The application involves a one-to-many relationship between a management station and a set of managed stations: The management station is able to get and set objects in the managed stations' and is able to receive traps from the managed stations. Thus, from an operational or control point of view, the management station "manages" a number of management stations. There may be a number of management stations, each of which manages all or a subset of the managed stations in the configuration. These subsets may overlap.

Interestingly, we also need to be able to view SNMP network management as a one-to-many relationship between a managed station and a set of management stations. Each managed station controls its own local MIB and must be able to control the use of that MIB by a number of management stations. There are three aspects to this control:

1. *Authentication service*: The managed station may wish to limit access to the MIB to authorized management stations.
2. *Access policy*: The managed station may wish to give different access privileges to different management stations.
3. *Proxy service*: A managed station may act as a proxy to other managed stations. This may involve implementing the authentication service and/or access policy for the other managed systems on the proxy system.

All of these aspects relate to security concerns. In an environment in which responsibility for network components is split, such as among a number of administrative entities, managed systems need to protect themselves and their MIBs from unwanted and unauthorized access. SNMP, as defined in RFC 1157, provides only a primitive and limited capability for such security, namely the concept of a community.

An *SNMP community* is a relationship between an SNMP agent and a set of SNMP managers that defines authentication, access control, and proxy characteristics. The community concept is a local one, defined at the managed system. The managed

system establishes one community for each desired combination of authentication, access control, and proxy characteristics. Each community is given a unique (within this agent) community name, and the management stations within that community are provided with and must employ the community name in all get and set operations. The agent may establish a number of communities, with overlapping management station membership.

Since communities are defined locally at the agent, different agents may use the same name. This identity of names is irrelevant and does not indicate any similarity between the defined communities. Thus, a management station must keep track of the community name or names associated with each of the agents that it wishes to access.

#### *Authentication Service:*

An authentication service is concerned with ensuring that a communication is authentic. In the case of an SNMP message, the function of an authentication service would be to assure the recipient that the message is from the source from which it claims to be. As defined in RFC 1157, SNMP provides for only a trivial scheme for authentication. Every message (get or put, request) from a management station to an agent includes a community name. This name functions as a password, and the message is assumed to be authentic if the sender knows the password.

With this limited form of authentication, many network managers will be reluctant to allow anything other than network monitoring; that is, *Get* and *Trap* operations. Network control, via a *Set* operation, is clearly a more sensitive area. The community name could be used to trigger an authentication procedure, with the name functioning simply as an initial password-screening device. The authentication procedure could involve the use of encryption/decryption for more secure authentication functions.

#### *Access Policy:*

By defining a community, an agent limits access to its MIB to a selected set of management stations. By the use of more than one community, the agent can provide different categories of MIB access to different management stations. There are two aspects to this access control:

1. **SNMP MIB view:** a subset of the objects within a MIB. Different MIB views may be defined for each community. The set of objects in a view need not belong to a single subtree of the MIB.
2. **SNMP access mode:** an element of the set {READ-ONLY, READ-WRITE}. An access mode is defined for each community.

The combination of a MIB view and an access mode is referred to as an SNMP *community profile*. Thus, a community profile consists of a defined subset of the MIB at the agent, plus an access mode for those objects. The SNMP access mode is applied uniformly to all objects in the MIB view. Thus, if the access mode READ-ONLY is selected, it applies to all the objects in the view and limits management stations' access to this view to read-only operations.

Within a community profile, two separate access restrictions must be reconciled. Table 3.1 shows the rules for reconciling an object's ACCESS clause with the SNMP access mode imposed for a particular view.

MIB Access Category	SNMP Access Mode	
	READ-ONLY	READ-WRITE
Read-only	Available for get and trap operations	
Read-write	Available for get and trap operations	Available for get, set, and trap operations
	Available for get and trap operations, but the value is implementation specific	Available for get, set, and trap operations but the value is implementation for get and trap operations
Write-only		
Not accessible	Unavailable	

**Table 3.1 Access specifications**

Most of the rules are straightforward. However if an object is declared as write-only, it may be possible with SNMP to read that object. This is an implementation-specific matter.

### **3.5.2 Lexicographical Ordering**

An object identifier is a sequence of integers that reflects a hierarchical or tree structure of the objects in the MIB. Given the tree structure of a MIB, the object identifier for a particular object may be derived by tracing a path from the root to the object.

Because object identifiers are sequences of integers, they exhibit a lexicographical ordering, which can be generated by traversing the tree of object identifiers in the MIB, provided that the child nodes of a parent node are always depicted in ascending numerical order. This ordering extends to object instance identifiers, since an object instance identifier is also a sequence of integers.

An ordering of object and object instance identifiers is important because a network management station may not know the exact makeup of the MIB view that an agent presents to it. The management station therefore needs some means of searching for and accessing objects without specifying them by name. With the use of lexicographical ordering, a management station can in effect traverse the structure of a MIB. At any point in the tree, the management station can supply an object or object instance identifier and ask for the object instance that occurs next in the ordering.

### **3.5.3 Protocol Specification**

The following subsection gives the overall message format for SNMP and then describes each of the protocol data units (PDUs) that can be carried in a message.

#### **SNMP Formats**

##### **3.5.3.1 SNMP Formats**

With SNMP, information is exchanged between a management station and an agent in the form of an SNMP message. Each message includes a version number indicating the version of SNMP, a community name to be used for this exchange, and one of five types of protocol data units. Figure 3.8 formally depicts this structure and Table 3.2 defines constituent fields.

<b>Version</b>	<b>Community</b>	<b>SNMP PDU</b>			
----------------	------------------	-----------------	--	--	--

(a) SNMP message

<b>PDU type</b>	<b>Request - id</b>	<b>0</b>	<b>0</b>	<b>Variable bindings</b>
-----------------	---------------------	----------	----------	--------------------------

(b) GetRequest PDU, GetNextRequest PDU, and SetRequest PDU

<b>PDU type</b>	<b>Request - id</b>	<b>Error-status</b>	<b>Error-index</b>	<b>Variable bindings</b>
-----------------	---------------------	---------------------	--------------------	--------------------------

(c) GetResponse PDU

<b>PDU type</b>	<b>Enter- prise</b>	<b>Agent- address</b>	<b>Generic trap</b>	<b>Specific trap</b>	<b>Time stamp</b>	<b>Variable bindings</b>
-----------------	-------------------------	---------------------------	-------------------------	--------------------------	-----------------------	------------------------------

(d) Trap PDU

<b>Name 1</b>	<b>Value 1</b>	<b>Name 2</b>	<b>Value 2</b>	<b>...</b>	<b>Name n</b>	<b>Value n</b>
---------------	----------------	---------------	----------------	------------	---------------	----------------

(e) Variable bindings

**Figure 3.8 SNMP message formats**

**3.5.3.2 Transmission of an SNMP Message**

In principle, an SNMP entity performs the following actions to transmit one of the five PDU types to another SNMP entity:

1. The PDU is constructed, using the ASN.1 structure defined in RFC 1157.
2. This PDU is then passed to an authentication and a community name. The authentication service then performs any required transformations for this exchange, such as encryption or the inclusion of an authentication code, and returns the result.
3. The protocol entity then constructs a message, consisting of a version field, the community name, and the result from step 2,
4. This new ASN.1 object is then encoded using the basic encoding rules and passed to the transport service.

In practice, authentication is not typically invoked.

<b>Field</b>	<b>Description</b>
Version	SNMP version
Community	A pairing of an SNMP agent with some arbitrary set of SNMP application entities ( The name of the community acts as a password to authenticate the SNMP message.
Request-id	Used to distinguish among outstanding requests by providing each request with a unique ID
Error-status	Used to indicate that an exception occurred while processing a request; values are noError(0), tooBig(1), noSuchName(2), badValue(3), readOnly(4), genErr(5).
Error-index	When error-status is nonzero, may provide additional information by indicating which variable in a list caused the exception.
Variable bindings	A list of variable names and corresponding values.
Enterprise	Type of object generating trap; based on sysObjectID
Agent-addr	Address of object generating trap
Generic-trap	Generic trap type; values are coldStart(0), warmStart(1), linkDown(2), linkup(3), authentication-Failure(4), egpNeighborLoss(5), enterprise-specific(6)
Specific-trap	Specific trap code
Time-stamp	Time elapsed between the last (re)initialization of the network entity and the generation of the trap; contains the value of sysUpTime

**Table 3.2 SNMP message**

### **3.5.3.3 Receipt of an SNMP Message**

In principle, an SNMP entity performs the following actions upon reception of an SNMP message.

1. It does a basic syntax-check of the message and discards the message if it fails to parse.
2. It verifies the version number and discards the message if there is a mismatch.

3. The protocol entity then passes the user name, the PDU portion of the message, and the source and destination transport addresses (supplied by the transport service that delivered the message) to an authentication service.
  - a. If authentication fails, the authentication service signals the SNMP protocol entity, which generates a trap and discards the message.
  - b. If authentication succeeds, the authentication service returns a PDU in the form of an ASN.1 object that conforms to the structure defined in RFC 1157.
4. The protocol entity does a basic syntax-check of the PDU and discards the PDU if it fails to parse. Otherwise, using the named community, the appropriate SNMP access policy is selected and the PDU is processed accordingly.

In practice, the authentication service serves merely to verify that the community name authorize the receipt of messages from the source SNMP entity.

#### 3.5.3.4 Variable Bindings

All SNMP operations involve access to an object instance. Only *leaf* objects in the object identifier tree may be accessed; that is, only scalar objects. However, it is possible SNMP to group a number of operations of the same type (get, set, trap) into a single message. Thus, if a management station wants to get the values of all the scalar objects in a particular group at a particular agent, it can send a single message, requesting all values, and get a single response, listing all values. This technique can greatly reduce the communication burden of network management.

To implement multiple-object exchanges, all of the SNMP PDUs include a *variablebindings* field. This field consists of a sequence of references to object instances, together with the value of those objects. Some PDUs are concerned only with the name of the object instance (for example, get operations). In this case, the receiving protocol entity ignores the value entries in the *variablebindings* field. RFC 1157 recommends that in such cases the sending protocol entity use the ASN.1 value NULL for the value portion of the *variablebindings* field.

### 3.5.4 *GetRequest PDU*

The *GetRequest* PDU is issued by an SNMP entity on behalf of a network management station application. The sending entity includes the following fields in the PDU:

- **PDU Type:** This indicates that this is a *GetRequest* PDU.
- **Request-id:** The sending entity assigns numbers such that each outstanding request to the same agent is uniquely identified. The request-id enables the SNMP application to correlate incoming responses with outstanding requests. It also enables an SNMP entity to cope with duplicated PDUs generated by an unreliable transport service.
- **Variablebindings:** This lists the object instances whose values are requested.

The receiving SNMP entity responds to a *GetRequest* PDU with a *GetResponse* PDU containing the same request-id. The *GetRequest* operation is atomic. If the responding entity is able to provide values for all of the variables listed in the incoming variablebindings list, then the *GetResponse* PDU includes the variablebindings field, with a value supplied for each variable. If at least one of the variable values cannot be supplied, then no values are returned. The following error conditions can occur.

1. An object named in the variablebindings field may not match any object identifier in the relevant MIB view, or a named object may be of an aggregate type and therefore not have an associated instance value. In either case, the responding entity returns a *GetResponse* PDU with an error-status of *noSuchName* and a value in the error-index field that is the index of the problem object in the variable-bindings field. Thus, if the third variable listed in the incoming variable-bindings field is not available for a get operation, then the error-index field contains a 3.
2. The responding entity may be able to supply values for all variables in the list, but the size of the resulting *GetResponse* PDU may exceed a local limitation. In that case, the responding entity returns a *GetResponse* PDU with an error-status of *tooBig*.



3. The responding entity may not be able to supply a value for atleast one of the objects for some other reason. In that case, the responding entity returns a GetResponse PDU with an error-status of genErr and a value in the error-index field that is the index of the problem object in the variablebindings field.

### **3.5.5 *GetNextRequest PDU***

The GetNextRequest PDU is almost identical to the GetRequest PDU. It has the same PDU exchange pattern and same format as the Get-Request PDU. The only difference is the following: In the GetRequest PDU, each variable in the variablebindings list refers to an object instance whose value is to be returned. In the GetNextRequest PDU, for each variable, the respondant is to return the value of the object instance that is next in lexicographical order. Like GetRequest, GetNextRequest is atomic: Either all requested values are returned or none is.

The apparently minor difference between GetRequest and GetNextRequest has tremendous implications. It allows a network management station to discover the structure of a MIB view dynamically. It allows a network management station to discover the structure of a MIB view dynamically. It also provides an efficient mechanism for searching a table whose entries are unknown.

### **3.5.6 *SetRequest PDU***

The SetRequest PDU is issued by an SNMP entity on behalf of a network management station application. It has the same PDU exchange pattern and the same format as the GetRequest PDU. However, the SetRequest is used to write an object value rather than read one. Thus the variablebindings list in the SetRequest PDU includes both object instance identifiers and a value to be assigned to each object instance listed.

The receiving SNMP entity responds to a SetRequest PDU with a GetResponse PDU containing the same request-id. The SetRequest operation is atomic: Either all of the variables are updated or none is. If the responding entity is able to set values for all of the variables listed in the incoming variablebindings list, then the GetResponse PDU includes the variablebindings field, with a value supplied for each variable. If at least one of the variable values cannot be supplied, then no values are returned, and no values are updated.

The same error conditions used in the case of Get-Request may be returned (noSuchName, tooBig, genErr). One other condition may be reported: badValue. This is returned if the SetRequest contains at least one pairing of variable name and value that is inconsistent. The inconsistency could be in the type, length, or actual value of the supplied value.

### 3.5.7 *Trap PDU*

The Trap PDU is issued by an SNMP entity on behalf of a network management agent application. It is used to provide the management station with an asynchronous notification of some significant event. Its format is quite different from that of the other SNMP PDUs. The fields are

- PDU type: indicating that this is a Trap PDU
- Enterprise: identifies the network management system subsystem that generated the trap (its value is taken from sysObjectID in the System group).
- Agent-addr: the IP address of the object generating the trap.
- Generic-trap: one of the predefined trap types.
- Specific-trap: a code that indicates more specifically the nature of the trap.
- Time-stamp: the time between the last (re)initialization of the network entity that issued the trap and the generation of the trap.
- Variable bindings: additional information relating to the trap (The significance of this field is implementation-specific).

The generic-trap field may take on one of seven values.

1. ColdStart(0): The sending SNMP entity is reinitializing itself such that the agent's configuration or the protocol entity implementation may be altered. Typically, this is an unexpected restart due to a crash or major fault.
2. WarmStart(1): The sending SNMP entity is reinitializing itself such that neither the agent's configuration nor the protocol entity implementation is altered. Typically this is a routine restart.

3. **LinkDown(2)**: Signals a failure in one of the communications links of the agent. The first element in the `variablebindings` field is the name and value of the `ifIndex` instance for the referenced interface.
4. **Linkup(3)**: Signals that one of the communications links of the agent has come up. The first element in the `variablebindings` field is the name and value of the `ifIndex` instance for the referenced interface.
5. **AuthenticationFailure(4)**: This signals that the sending protocol entity has received a protocol message that has failed authentication.
6. **EgpNeighborLoss(5)**: This signals that an EGP neighbor for whom the sending protocol entity was an EGP peer has been marked down and the peer relationship no longer exists.
7. **EnterpriseSpecific(6)**: Signifies that the sending protocol entity recognizes that some enterprise-specific event has occurred. The `specific-trap` field indicates the type of trap.

Unlike the `GetRequest`, `GetNextRequest`, and `SetRequest` PDUs, the `TrapPDU` does not elicit a response from the other side.

## DESIGN

## 4.1 Mobile Agent Life Cycle

Figure 4.1 gives the mobile agent life cycle with respect to network manager, mobile

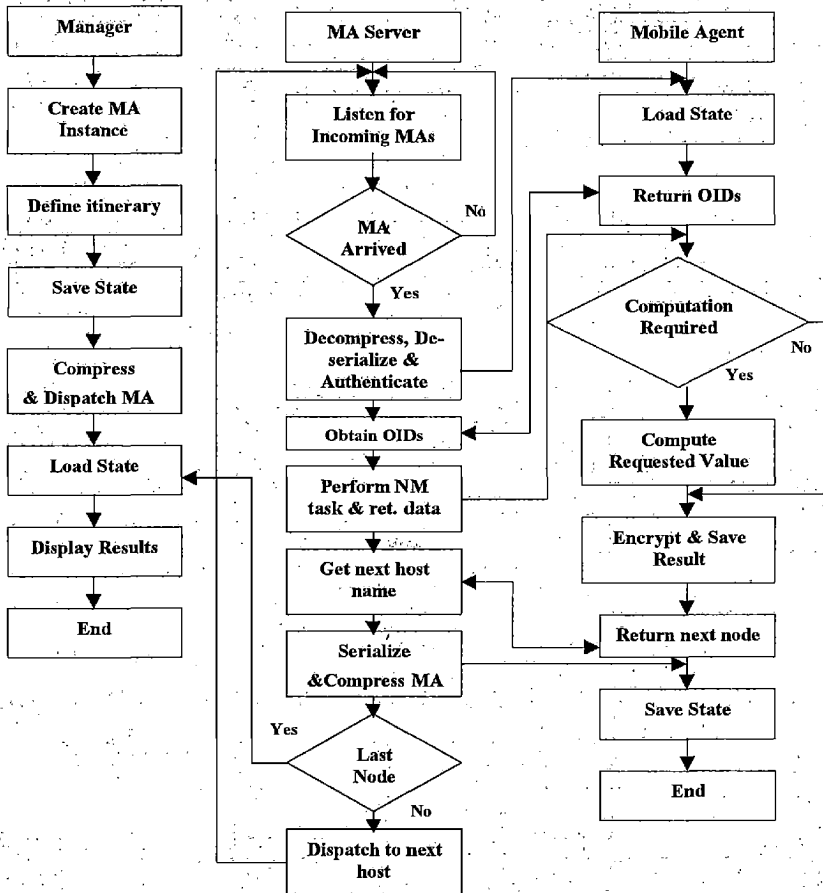
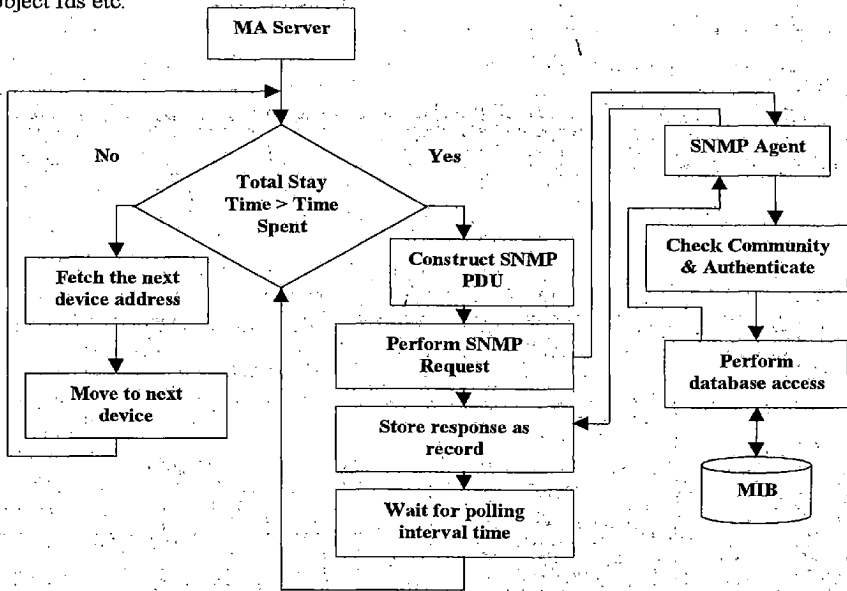


Figure 4.1 Mobile Agent Life Cycle

agent server and mobile agent perspective. The manager specifies all the information related to the operations that should be performed at the device like, type of SNMP operation, object Ids etc.

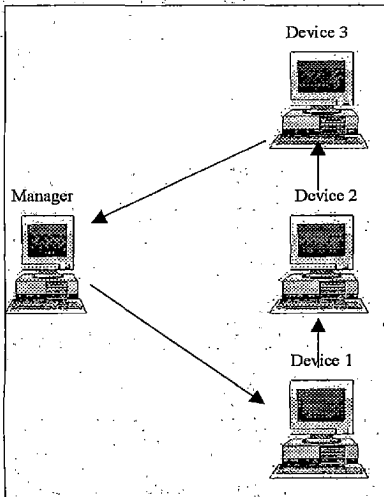


**Figure 4.2 Interactions between Mobile Agent and SNMP Agent at Device**

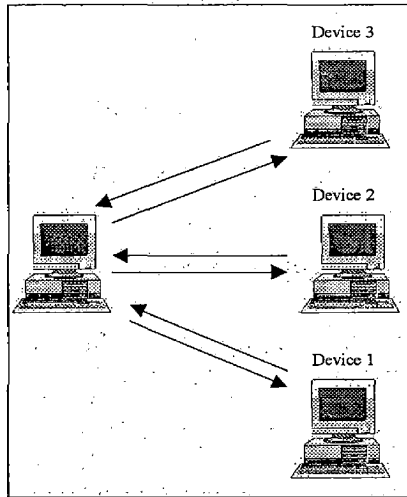
Figure 4.2 shows the interaction between a mobile agent arrived at a device and the SNMP agent residing at that device. Each mobile agent carries with it the total time it should stay at the device and the polling interval between the SNMP requests that should be queried to the local SNMP Agent.

#### **4.2 Mobile Agent Traveling Patterns**

In this work two traveling patterns [5] are followed by the mobile agents for performing the management tasks, itinerary model and broadcast model. The following figures represent the way the mobile agents move in the network.



**Figure 4.3 Itinerary Model**



**Figure 4.4 Broadcast Model**

#### 4.2.1 Itinerary Model

In this model the mobile agent is equipped with the list of devices to be traversed and the operations to be performed at each of the devices. The mobile agent traverses through the list in the order specified by the manager and collects the results at each of the device and carries back to the manager after all the devices are traversed.

#### 4.2.2 Broadcast Model

In this model manager creates as many mobile agents as there are devices to be managed each of them equipped with its own properties for performing the operations at the device. After the task is completed the mobile agent will be back to the manager with the results obtained.

### 4.3 AgentSpace Design Issues

The main participants in the pattern are:

#### 4.3.1 Client

- Manipulates agents through the AgentView reference.

- Clients can be other agents or other objects (for instance Java applets).

#### 4.3.2 AgentView

- The AgentView is an adaptation of the *Proxy* [25] and *Remote Proxy* patterns. This pattern is very suitable to support transparent and secure access to these different types of objects.
- The aim of AgentView is to provide transparent access to agents. This access is done indirectly through proxies in order to protect them, and to hide transparently their current localization (this is important due to the mobility characteristic).
- Additionally, AgentView avoids the need to create and manage remote/virtual classes (e.g., stubs and skeletons in RMI and CORBA implementations). Usual examples of operations provided/protected through agent proxies are: `sendMessage`, `getCurrentPlace`, `start`, `moveTo`, `getClassName`, etc.

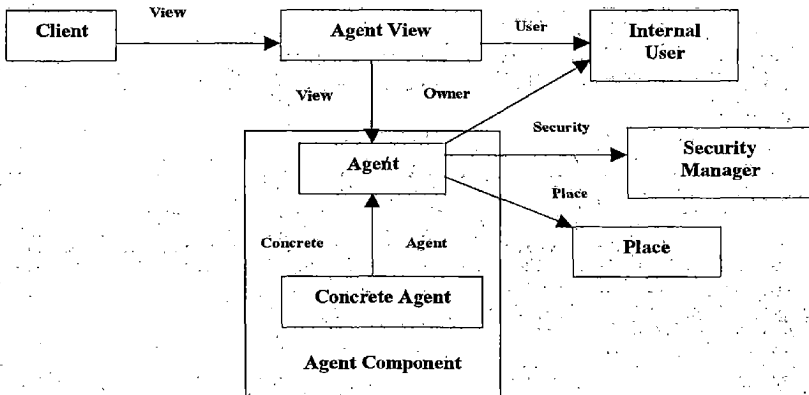


Figure 4.5 Generic Structure of Agent Pattern – class diagram

#### 4.3.3 User

- The user is identified by a unique identity, which may contain for instance: his/her name; a public key; a set of certificates; the organization and country he/she belongs to; and his/her e-mail. Moreover, the user can have different identifiers depending on the context he/she belongs to. This specific identity, managed in every Agent

Server's context, is represented by the User class, which may contain, in addition to all fields mentioned above, the authentication attributes (e.g., login and password).

- The agent's owner has necessarily an associated user identity, represented always by a User instance.
- Different users can access the same agent however, only through the corresponding AgentView instance. Depending on the agent's security manager, each access is allowed or not (see Figure 3.7).

#### 4.3.4 Agent

- The Agent abstract class is the visible and extensible part of the Agent pattern.
- Basically, programmers should derive the Agent abstract class in order to build their own concrete classes. The agent class has three main groups of methods as depicted in Figure 4.6: (1) public final; (2) callbacks; and (3) helper methods.

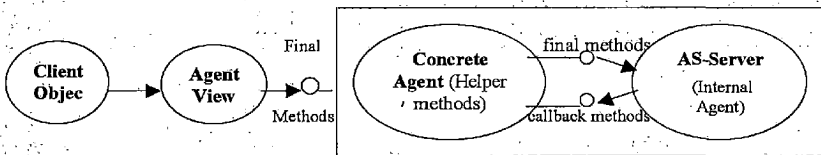


Figure 4.6 Agent's main groups of methods

- *Final methods* are pre-defined operations provided by all agent classes that cannot be changed by the programmer. Examples of these final methods are: moveTo, save, die, backHome, clone, getId, sendMessage, etc.
- On the other hand, *callbacks* are methods customized by specific agent classes, and are usually invoked transparently as the result of some event. Events are triggered by some action started by the agent itself or by other related entity, such as another agent, an end-user (via same applet), a time service, etc. The callback mechanism provides the desired extensibility of the Agent pattern. Usual examples of callbacks are: run, onCreate, beforeDie, handleMessage, etc.



- Finally, agent classes also have *helper methods*, generally with private or protected access modifiers, in order to support specific functions of that class/object. These methods are used internally by callback methods.
- The Agent instance provides transparently several services, such as: persistence, communication, mobility, naming and access control. Additionally, the Agent instance may keep related information, such as: the current and native place identities, security policy object, a reference to the concrete agent itself, its own identity, its owner identity, a reference to the involved security manager, and the group of threads involved.

#### 4.3.5 ConcreteAgent

- Concrete agent classes are Agent subclasses.
- Basically they define helper methods and specialized callbacks that, as a whole, implement the agent's specific functionality.

#### 4.3.6 SecurityManager

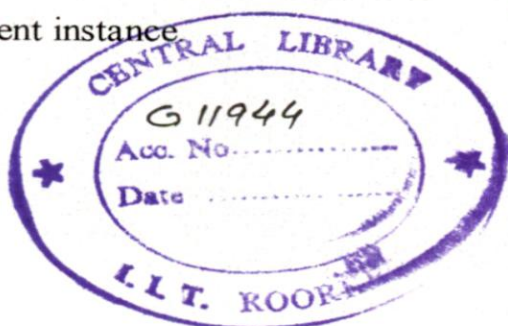
- This class specifies the agent access security policy.
- The agent's SecurityManager instance controls all the operations made available on the agent component through each AgentView instance.

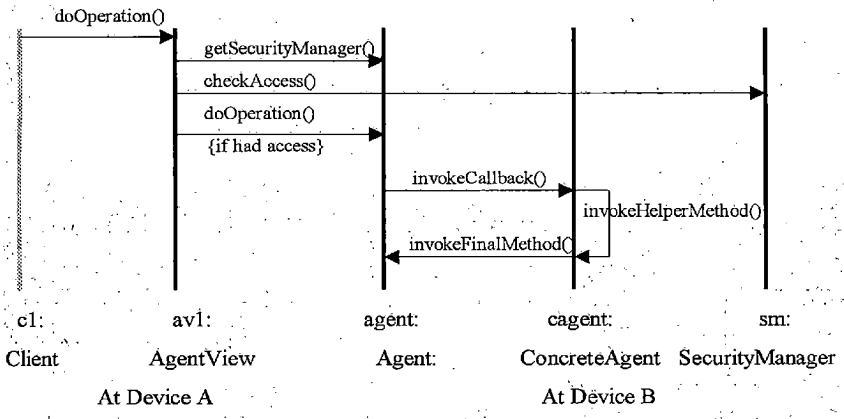
#### 4.3.7 ExecutionPlace

- This class specifies the agent's computational environment, which corresponds to the place where it was created as well as where it is currently resident.
- This class offers specific functions provided by the involved agent support system.
- The notion of execution places is a crucial component supported by mobile agent frameworks due to the need of handling conveniently agent's mobility operations.

#### 4.3.8 Collaborations

Clients call standard agent operations through an AgentView instance. Depending on the agent's security policy and on the involved user, the operation is executed, or not, on the involved agent instance.





**Figure 4.7 Generic interaction of the Agent Pattern – scenario diagram**

Final methods are basically executed by the Agent instance. On the other hand, Callbacks, resulting from the execution of final methods (e.g., `moveTo`, `die`, `sendMessage`), are executed by the concrete agent instance. Lastly, some helper methods may be invoked by the execution of some callbacks, and this process might be repeated several times.

Figure 3.7 shows a UML collaboration scenario between an abstract client (i.e., the `c1` object), located in some device, and an abstract agent (i.e., the agent instance) located in another device.

## IMPLEMENTATION

---

The Mobile Agent based Network Management is implemented using

Programming Language:	Java (JDK1.1.8)
	K-Prolog
Platform:	Windows98
Tools/Packages:	AgentSpace (Mobile Agent System)
	Snm4_13 (Westhawk's java based SNMP Package)
	NetComponents (FTP, ICMP etc protocols package)
	Jftp (java based FTP server)
	JJPL (Java - Prolog Interface)
Database:	MS-Access

### 5.1 Mobile Agents – Programmers Perspective

In order to invoke a mobile agent first a meta agent class object should be written (i.e a file with .mac extension) which describes the respective agent class. It contains the details like the class name, description, version, author, initialization, properties etc. A mobile agent class can be invoked from the agentspace system after this meta agent class is loaded in the system which invokes the class written as follows.

#### 5.1.1 Creating agents

Agents are instances of some Agent derived class. They are created either:

- interactively through the AS-Client (or a similar tool), or
- by any previous created agent.

##### 5.1.1.1 Agent creation using AS-Client tools

In the first case, the AS-Client should perform basically the following algorithm (e.g., in some Applet extended class):

```
InternalUser user=AgentSpace.getUserByLogin(asid, "user-login", "user-
pwd");
ContextView cv= AgentSpace.getContextView(asid, user);
PlaceView pv= cv.getPlaceOf(getCurrentPlace());
```

```
AgentView av= pv.createAgent(user, "examples.MANM.MANMItinerary");
....
```

Firstly, one needs to get the `ContextView` and the `InternalUser` objects, respectively `cv` and `user`. The user has to specify login information, so that the `cv` object can check permissions.

Then, a reference is needed (eventually a remote reference) to the place where the agent should be created. Eventually an exception may be raised, in case the place doesn't exist, or if the user hasn't access to it. The security strategy may vary between places. For instance, one place may adopt a security strategy based on users' ACL, while another may adopt a security strategy based on a previous agent classes record.

Lastly, the agent is created by specifying user information and the agent class name. For security reasons, it is not allowed the agent creation from remote agent classes.

#### **5.1.1.2 Agent creation from another agent**

Let's suppose that the `MANMManager` agent class creates an instance of `examples.MANM.MANMItinerary` class in the same place as it is currently running at, and additionally creates a clone of itself.

The algorithm below shows the two ways to create agents: by explicit agent class specification, or by cloning. Note that the created agents are attached to the current place and have the same owner (`InternalUser`) as the corresponding agent creator's owner. Still, the same security issues should be posed as referred to above.

```
class MANMManager extends inesc.agentspace.Agent {
    ...
    void run() {
        ...
        PlaceView pv= getCurrentPlace();
        AgentView av1= pv.createAgent(getOwner(),
"examples.MANM.MANMItinerary");
        av1.start();
        ...
        AgentView av2= clone();
    }
}
```

#### **5.1.2 Obtaining references to agents**

An `AgentView` object, as seen above, is an agent reference. There are several ways to get an agent view:

- as the result of agent creation methods (createAgent and clone, as above), as well as
- through an AgentView factory method.

To obtain a reference to any agent, through an AgentView factory, it is only need to know its corresponding identity (aid). With this aid the factory method getAgentOf can be invoked from the ContextView object.

```
AgentId aid1= new AgentId("90.0.11.29:8888/PID_1|AID_7");
AgentView av1= cv.getAgentOf(aid1.toString());
AgentView av2= cv.getAgentOf("90.0.17.61:8888/PID_2|AID_17");
....
```

### 5.1.3 Agents Navigation

In general, Java agents are not able, like other MASs (e.g., Telescript or Agent-Tcl), to keep their execution state after a navigation operation. This limitation is due to the fact that it is not possible (in the current Java version) to access a thread's execution stack. AgentSpace uses Java's reflection capabilities to give programmers the possibility to specify the callback they want to be invoked after that operation, instead of always calling the same callback after the move/dispatch operation (as the run callback in Aglets system). This approach reduces significantly the current Java limitation, offering a more elegant and simple way to program agent classes as it (avoiding the "spaghetti code" of long switch instructions found in Aglets agents).

Before a move operation, every agent needs to have a Ticket object in order to be accepted in the target place. A Ticket is a certificated object that keeps the information required by the target place security policy in order the agent may be accepted. There are two final methods concerning agent mobility: moveTo, and backHome. The second method gives the agent the possibility to go back home, and after that, in its native place, to have its afterBackHome callback invoked.

```
class MANMItinerary extends inesc.agentspace.Agent {
    ...
    void run() {
        ...
        Ticket tck= new Ticket(this);
        PlaceId pid= new PlaceId(getCurrentPlaceId().toString());
        moveTo(pid, tck, "atDevice")
    }
    void atDevice () {
        processSNMPRequest();
        backHome();
    }
}
```

```

    }
    void afterBackHome () {
        System.out.println("Results:");
        ...
    }
    ...
}

```

#### 5.1.4 Agents' communication

AgentSpace provides two basic and complementary ways to support inter-agent communication, namely: i) the AgentView' sendMessage method; and ii) the RemoteInvocation instance.

The AgentView' sendMessage mechanism permits to send a Message object and provides a simple but effective communication.

A Message class basically keeps: a key-tag attribute indicating the meaning of its content, in some application-dependent context;

1. a content object;
2. a message creation timestamp; and
3. the agent/object message' sender.'

```

Message msg1= new Message(aid, "trap", trapPDU);
av1= cv.getAgentOf(aid1.toString());
av1.sendMessage(msg1);
...

```

As a side effect of invoking sendMessage, its corresponding agent's handleMessage callback is invoked by the AS-Server. So, the agent class programmer is responsible by this callback, in order to handle conveniently the behavior related to the received messages.

The RemoteInvocation is a class that allows encapsulating the dynamically created Voyager's messenger. It provides a powerful way to invoke the agent's remote static methods. Due to Voyager capabilities, AgentSpace supports consequently OneWay, Synchronous, and Future message types.

## 5.2 Network Management Features

This work introduces different network management features based on mobile agents beyond basic itinerary model, like trap generation, active network-management, functionality based approach to reduce the amount of data carried by mobile agent, runtime

decision-making and dynamic service provisioning, which considerably increase the efficiency of network management.

### 5.2.1 Itinerary Model

Itinerary model is described as roaming management model. In this scheme a mobile agent visits the set of devices to be managed sequentially. The mobile agent is configured with the list of devices to be visited during its itinerary and also the SNMP statistics to be analyzed. Configuration of agents is done while AgentSpace server creates the agents at network management station. A mobile agent sequentially visits all devices to be managed. At each managed device it obtains required information, performs necessary calculations in analyzing data to reduce its size before it visits the next managed device. E.g.: - If a user wants to calculate the percentages of input and output errors at an interface he has to extract 8 MIB variables and then calculate the percentage, instead the mobile agent may be equipped with those calculation formulas so that it can directly return the percentage errors instead of all the objects' values like:

$$\text{Percent input errors} = ((\text{ifInErrors})/(\text{total packets received})) * 100$$

$$\text{Percent output errors} = ((\text{ifOutErrors})/(\text{total packets sent})) * 100,$$

Where

$$\text{Total packets received} = (\text{ifInUcastPkts} + \text{ifInBroadcasts} + \text{ifInMulticasts}),$$

$$\text{Total packets sent} = (\text{ifOutUcastPkts} + \text{ifOutBroadcasts} + \text{ifOutMulticasts}),$$

This function is often used in fault monitoring. If the interface error rate is more than 1%, then there is a problem with the interface of the machine. If the error rate is less than 1% and network shows poor performance, then it could be deduced that there is a problem with the media.

### 5.2.2 Broadcast Model

In this model a MA is dispatched to each managed device. All the dispatched MA's stay at their respective device and analyze it for amount of time specified by the user. MA's poll the managed devices after each polling interval. Each MA agent stays there for an amount of time equal to the total number of polling intervals. It executes its task after each polling interval, performing necessary calculations on obtained management statistics,

analyzing them by using some functions equipped into the MA as in the above model and get back to the manager. E.g. - A performance management application can use *ifInOctets* and *ifOutOctets* of the interfaces group in MIB to compute the utilization of an interface over an interval of time. To perform this computation, two different polling intervals are required: one to find total bytes per second at time *x* and another to find total bytes per second at time *y*. The following equation computes utilization, *U(t)* for the polling interval (*x-y*) seconds:

$$U(t) = ((ifInOctets_y - ifInOctets_x) + (ifOutOctets_y - ifOutOctets_x)*8) / ((y-x)*ifSpeed)$$

Where, *ifSpeed* is the bandwidth of the interface, *inOctetsx* is the bytes received by the interface at time *x*, *ifOutOctetsx* is the bytes sent by the interface at time *x*, (*y-x*) is the polling interval.

### 5.2.3 Active Network-Management

Present work exploits the persistency feature of mobile agent further to achieve active network-management where the management functionality will be active all the time at the device. This is possible by making the mobile agent stay at the device forever and return the results as and when required by the user or if there is any change in the device objects' values. This considerably reduces the amount of data being transferred as well as network traffic. This will be more advantageous when the user can change the configuration of the mobile agent from the remote station as one wants different types of monitoring operations (E.g. performance, efficiency etc) to be carried by the mobile agent with time. This is achieved in the system with the AgentView class, which provides an interface to the remote mobile agent so that its configuration can be changed at runtime by sending a message (Message object) containing the action to be taken. Here to return the object values from the device, the mobile agent connects with a process running at AS-Client, via socket communication. Whenever user wants to obtain values he sends a message object using AgentView class's sendMessage() method to the mobile agent at the device as a signal to retrieve the results the mobile agent has obtained so far.



## 5.2.4 Trap Generation

A trap enables the management station of a significant event in a device. Irrespective of the type of network management approach being used (mobile agent or SNMP), traps play vital role. The important question in case of mobile agent based approach is, how to make a remote mobile agent receive a trap from a device in the network, which doesn't have any information about the present location of the mobile manager (agent). This is important because of two reasons:

1. If a device which is included in the deployed mobile agent's itinerary, has generated a trap specifying a significant event before the mobile agent reaches that device, then the mobile agent has to respond to the trap accordingly (changing the task to be done at that device like bypassing it from itinerary etc) instead of going ahead with its task.
2. If the trap generating device is not in the itinerary or the trap was generated after it has moved away from the device then it has to take the decision of whether to log, or forward to other network management system (in case of multiple managers) or taking any other action.

The AgentSpace mobile agent system provides an interface to the mobile agent generated and makes this trap generation to a mobile manager possible. In order to receive the traps from the devices a thread process running parallel to agent configuration class is created which will be executing at the AS-Client. It is important to note here that a process running outside the AgentSpace system can't send a message to the mobile manager since it can't create an AgentView object. When a trap is received the process will send a message object containing the trap related information like the source, type of the trap to the manager in the network with the help of AgentView class's sendMessage() method. Whenever the mobile manager receives a message it will take the corresponding actions by calling the appropriate methods for different conditions as specified above.

The mobile agent can perform the predefined tasks for the different conditions specified above in two ways.

1. It can create another mobile agent and will be sent to the trap-generated device with predefined task according to the type of trap received.

2. It can modify its itinerary so that the next immediate device to be visited being the trap generated device. After performing the task there it will follow the rest of the itinerary as specified.

This system defines tasks that a mobile agent should perform at the trap-generated device in each of the cases specified above for the different type of traps that could be received.

### 5.2.5 Runtime Decision-Making

Java Interface for ProLog [27] (JIPL) provides an interface between java and prolog, which is used in this work to make the mobile agent to take decisions at runtime. One of the main drawbacks of SNMP being management intelligence too centralized, the use of prolog for distributing the decision-making capability makes this system more efficient. So that most of the decisions can be taken at the device itself. This not only reduces the task of centralized manager but also makes the responses to the events instantaneous. As explained in the previous section there are instances that require different actions to be taken for different events makes the use of prolog more useful. E.g. - Consider the case in trap generation discussed above. For each type of trap received by the mobile agent, it has to take different actions defined earlier by the manager for all the cases discussed in the previous section.

```
:-module decision.
:-public decision/2.
decision(TrapType,Ob):-
    decide("0",TrapType),
    javaMethod(Ob,displayTrap("display"),_),
    javaMethod(Ob,logTrap("log"),_),
    purge(decision.pl).
decision(TrapType,Ob):-
    decide("1",TrapType),
    javaMethod(Ob,forwardTrap("forward"),_),
    purge(decision.pl).
decide(X,X).
```

so that the corresponding method is invoked from prolog. Though this example looks simple, these types of decisions will be more efficient in case a mobile manager want to select the type of service it can temporarily provide at the remote site or in deciding the type of operation to be performed because of specific events (E.g. -attacks to Intrusion detection systems, service failures at web servers etc) [28] occurring at the device.

### **5.2.6 Dynamic Service Provisioning**

This feature greatly reduces the amount of time the system should be idle when an error occurred in the system stops a service it is providing. One can temporarily provide a service using mobile agent at the remote site before the administrator rectifies the system. This feature is also useful in restoring a system to its normal position. E.g. suppose a file is corrupted in a system, which is to be restored to continue its service and that there is no ftp client application available even if the administrator wants to restore the file. Then the SNMPPAgent can generate a trap to a mobile agent (manager) with the information containing the file corrupted, location of the file and the system address. Then the mobile agent will move to the system and perform file transfer from a system containing the required files to the system to be rectified. This is possible by temporarily transferring ftp client application to the remote system and connecting to an ftp server residing at remote site. This feature is implemented in the work using jftp, a java based ftp client application.

## RESULTS AND DISCUSSIONS

### 6.1 Starting AgentSpace Server

Following console image shows how to start a console based AgentSpace server. It asks for login name and password. At the beginning there will be two users i.e admin and anonymous. According to manager requirements one can create users and assign passwords for them. When an AgentSpace is started its own console will be opened with `as>` command prompt. It provides different commands to create agents, places etc and to display them in its context.

```

C:\agentspace>java -nojit -classpath BSServer.zip;aplic\ag_natives\;c:\p\c\j\j\j\
_jar;c:\netcomponents;c:\agentspace\aplic\ag_natives\examples\M88M;c:\jdk1.1.8\l
ib\classes.zip;c:\noyager1.0.1\lib\oyager1.0.1.jar inesc.as.server.BSDaemon
-root config -port 8888 -db 8888
AgentSpace Daemon Alfa 1.2.0, Copyright 1998
Dtg: la vez -- Contexto ainda nao criado -- ou criar!!
as> Login: admin
as> Password: adminpud
as> help
Options:
where options include:
help          show this message
?             show context
zp           show Places
za           show Acl context
zg           show Groups
zu           show Users
zmp         show My Places
zma         show My Agents
zmpy        show My Permissions
zmac        show MetaAgentClasses
umac <metaclass-name> show MetaAgentClass details
amac <file-name> <metaclass-name> add MetaAgentClass
rmac <metaclass-name> remove MetaAgentClass
cp <place-alias> [<owner>] !ss! create place
rp <pid> remove place
ca <pid> <alias> <ame> [<ini>] [<ss>] create agent
ra <aid> remove agent
ia <aid> agent interface
s save all <context>
q <quit - server shutdown
  
```

Figure 6.1 AgentSpace Server

### 6.2 Starting AgentSpace Client

Figure 6.2 shows the AS-Client applet, which provides a convenient way to create mobile agents, and provides many features to manage places, agents, groups, security classes, meta agent classes etc as shown in figure 6.3.

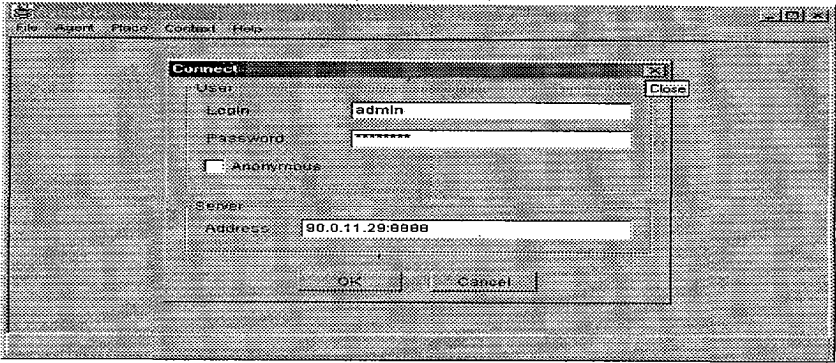


Figure 6.2 AS-Client Applet

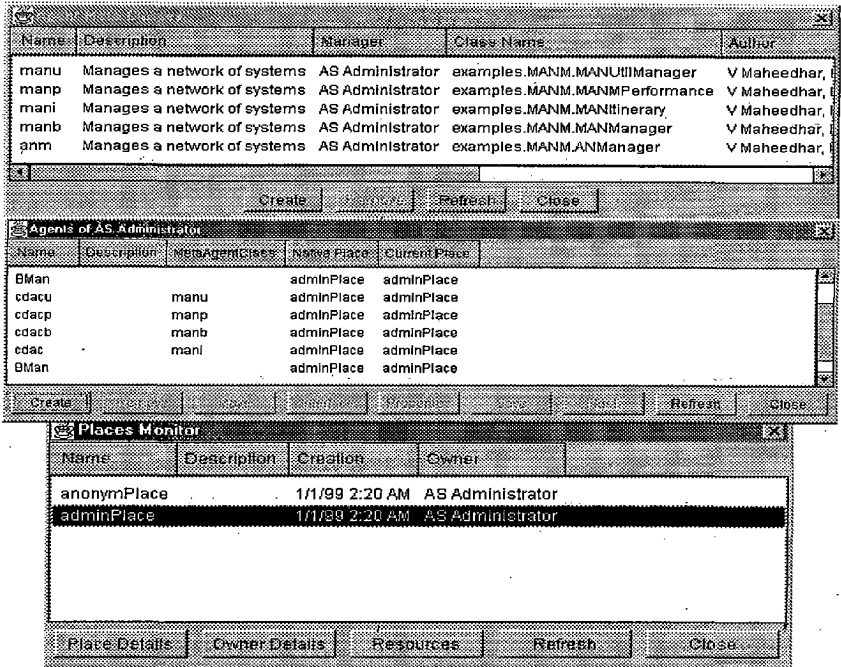


Figure 6.3 AS-Client's Features

### 6.3 SNMP Manager GUI

Figure 6.4 gives the SNMP based network management GUI for the manager who selects the SNMP operation (request type), objects to be retrieved or set, the agent address and sends request for the SNMP agent. The results obtained are viewed in a table, which contains the time of retrieval, device name, and results.

The screenshot shows the SNMP Manager GUI with the following configuration:

- Request Type:** GetObject
- Select the Object:** sysContact, sysName, sysObjectID
- Value:** intech-IT lab
- Selected Objects:** sysDescr, sysContact
- Agent Address:** 90.0.10.23
- Selected Objects (List):** intel-p3, V.Maheedhar-0077-22

The **VIEW DATABASE** table contains the following data:

Time of Retrieval	System/Device	sysDescr	sysObjectID	sysUpTime	Value	Request/Status
Fri Jan 01 01	MAHEEDHAR	intel-p4.40GB	2	10	intel-p4.40GB	GetObject SU
Fri Jan 01 01	MAHEEDHAR	sysDescr	2	10	intel-p4.40GB	GetObject SU
Fri Jan 01 01	MAHEEDHAR	sysObjectID	2	10	intel-p4.40GB	GetObject SU
Fri Jan 01 01	MAHEEDHAR	sysUpTime	2	10	intel-p4.40GB	GetObject SU
Fri Jan 01 01	MAHEEDHAR	sysDescr	2	10	intel-p4.40GB	GetObject SU
Fri Jan 01 01	MAHEEDHAR	sysObjectID	2	10	intel-p4.40GB	GetObject SU
Fri Jan 01 01	MAHEEDHAR	sysUpTime	2	10	intel-p4.40GB	GetObject SU
Fri Jan 01 01	MAHEEDHAR	sysDescr	2	10	intel-p4.40GB	GetObject SU
Fri Jan 01 01	MAHEEDHAR	sysObjectID	2	10	intel-p4.40GB	GetObject SU
Fri Jan 01 01	MAHEEDHAR	sysUpTime	2	10	intel-p4.40GB	GetObject SU
Fri Jan 01 01	MAHEEDHAR	sysDescr	2	10	Maheedhar V	GetObject SU
Fri Jan 01 01	MAHEEDHAR	sysContact	2	10	Maheedhar V	GetObject SU
Fri Jan 01 01	MAHEEDHAR	sysDescr	2	10	intel-p4.40GB	GetObject SU

Figure 6.4 SNMP Manager GUI

### 6.4 MANM – Itinerary/Broadcast Model

The screenshot shows the MANM GUI with the following configuration:

- Device Address:** 90.0.17.61
- Place ID:** PID\_1
- SNMP Request Type:** GetObject, GetNextObject, SetObject (selected)
- Object List:** 1.3.6.1.2.1.1.1, 1.3.6.1.2.1.1.2, 1.3.6.1.2.1.1.3, 1.3.6.1.2.1.1.4, 1.3.6.1.2.1.1.5
- Set Value:** (empty)
- Object ID:** 1.3.6.1.2.1.1.1, 1.3.6.1.2.1.1.4
- Value:** intel-p3, Mahee-9444194280
- Polling Interval:** 500
- Staying Time:** 500
- Device List:** 90.0.11.29:9999/PID\_1, 90.0.17.61:9999/PID\_1

Figure 6.5 MANM Manager GUI

Figure 6.5 shows the MANM Manager GUI for two of the traveling patterns discussed in the previous chapter i.e. itinerary model and broadcast model. The GUI asks for the device address, place id, objects to be retrieved at the device, polling interval between queries and the total time the mobile agent should be stayed at the device.

```
retrieved records from the devices' mib are:
90.0.11.29:8888/PID_1 Fri Jan 01 02:33:56 GMT+05:30 1999 GetObject 1.3.6.1.2.1.1
.4 Reheedhar 0-93441-92260 success
90.0.17.61:8888/PID_1 Wed Jan 02 19:42:27 GMT+05:30 2004 GetObject 1.3.6.1.2.1.1
.1 compaq success
```

Figure 6.6 Itinerary Model Results

```
retrieved records from the devices' mib are:
90.0.11.29:8888/PID_1 Fri Jan 01 02:41:22 GMT+05:30 1999 GetObject 1.3.6.1.2.1.1
.1 celeron success
Agent: 90.0.11.29:8888/PID_1|RID_4 in beforeDie()

retrieved records from the devices' mib are:
90.0.17.61:8888/PID_1 Wed Jan 02 19:49:19 GMT+05:30 2004 GetObject 1.3.6.1.2.1.1
.5 new system success
Agent: 90.0.11.29:8888/PID_1|RID_5 in beforeDie()
```

Figure 6.7 Broadcast Model Results

Figures 6.6 and 6.7 show the outputs obtained for itinerary model and broadcast model. In itinerary model single mobile agent returns with results at all the devices. Where as in broadcast model each mobile agent returns with the results at the corresponding device.

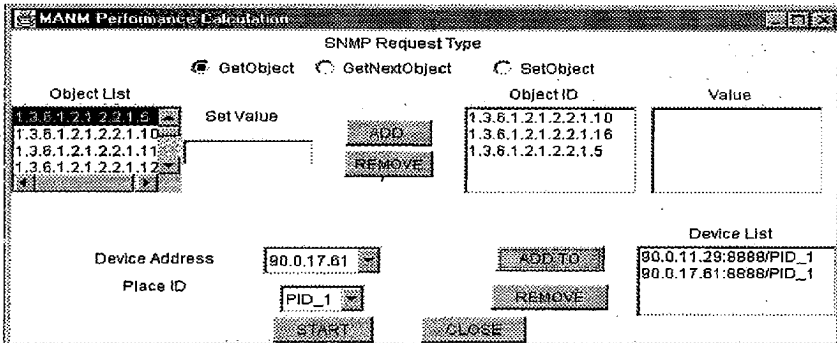
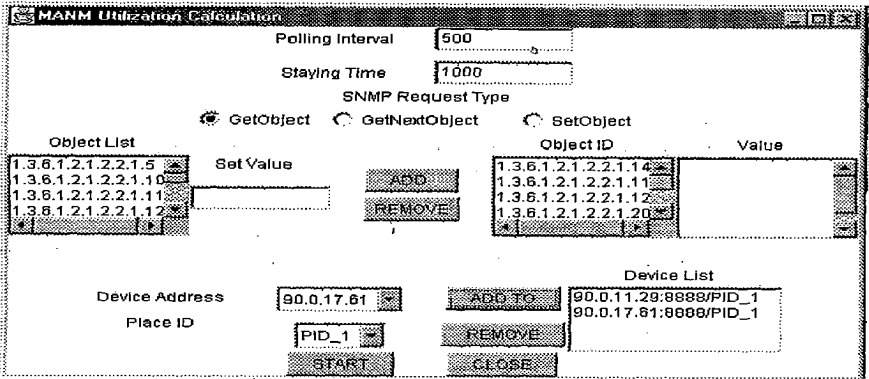


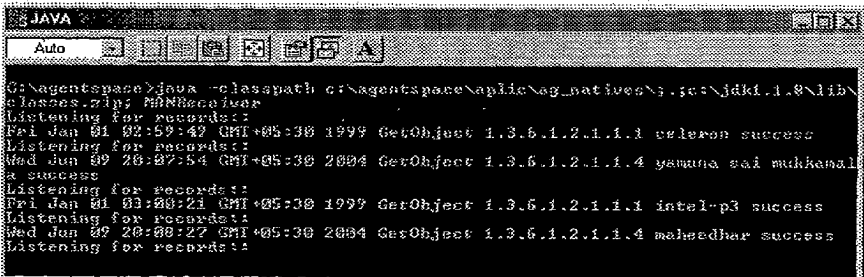
Figure 6.8 Performance Management GUI



**Figure 6.9 Interface Utilization Management GUI**

Figures 6.8. and 6.9 show slightly modified manager GUIs for returning with calculated performance and utilization values instead of all the object values. While deploying MAs for calculating utilization a single mobile agent will be cloned and deployed to as many devices as mentioned because the functionality of each MA is same.

## 6.5 Active Network-Management



**Figure 6.10 Results of Active Network-Management**

Figure 6.10 gives the results for active network-management. Once MAs are deployed they return the results through socket communication and they keep monitoring



the device for changes. Only when there is a change, they return the changed values to the manager.

## 6.6 Trap Generator and Dynamic Service Provisioning example with Runtime Decision-Making

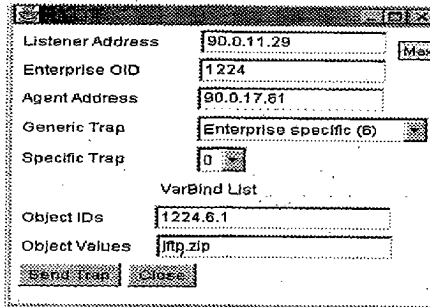


Figure 6.11 Trap Generator GUI

Figure 6.11 shows the GUI of trap generator when an enterprise specific trap has occurred. It specifies the related information in object Ids and Object Values fields that there is an error occurred in file jftp.zip and it has to be replaced. When the mobile agent (manager) receives this trap it takes the decision to provide FTP client service to transfer the specific file from a remote station. Figure 6.12 shows the output where K-prolog is started, ftp connection is established and the file is transferred.

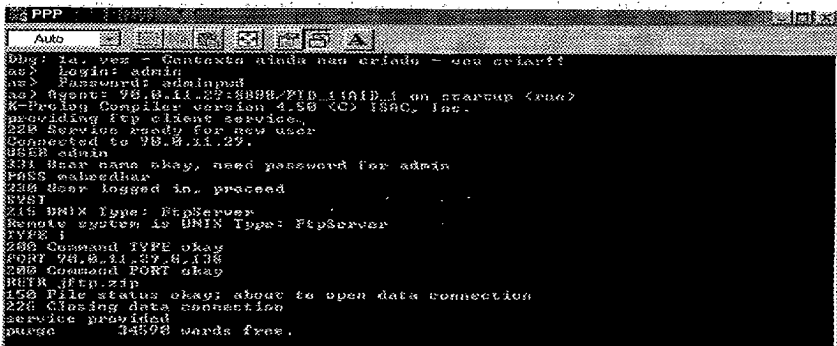


Figure 6.12 Dynamic FTP Client Service Provisioning

# CONCLUSIONS

---

Today's complex and heterogeneous networking environments require flexible and distributed network management solutions. Towards this end the MA based approach presents distinct advantages when compared with the techniques, which are currently in operation. This work implemented a hybrid model based on mobile agent and SNMP strategies for efficient management of heterogeneous networks. To exploit the potential of mobile agent technology the work has incorporated two agent-traveling patterns and introduced active network-management to considerably decrease the load of mobile agents and give instantaneous responses to changes in the network. This work proposes new methods to make the SNMP Agent able to generate traps to a mobile manager. The MAs are provided with runtime decision-making as well as dynamic service provisioning capabilities to make the best use of mobile agent technology in network management. This makes the application very efficient as it can take the managerial decisions by itself and considerably reduce the response time of the manager for error recovery. From this work it is concluded that the mobile agent technology clearly overcomes many of the limitations existing in conventional network management approaches like in reducing network traffic, extracting large amounts of data, scalability etc.

### **Future Scope:**

Few areas of improvement were identified during the dissertation, which include optimization of MA placement and deployment strategies [9,12] so that the amount of time to traverse the network can be reduced. Providing the MA with more intelligence by dynamically extending the decision-making capabilities according to different events occurring in the network. It is also observed that even though java is a suitable language for MA applications, improved bytecode verification is necessary because bytecode verifiers of several current standard java implementations also accept bytecode that does not represent a valid java program. So security measures have to be taken accordingly.



## References:

- [1] Maheedhar Valasa, Dr.P.R.Gupta, "Mobile Agent based Network Management", International Conference/workshop on Mobile Systems, E-Commerce and Agent Technology, (DMS/MSEAT'04) September 8-10, 2004.
- [2] Andrzej Bieszczad, Bernard Pagurek and Tony White "Mobile Agents for Network Management" IEEE Communications Surveys, Fourth Quarter 1998. Vol.1 No.1 [www.scs.carleton.ca/~arpwhite/documents/ieee-cs-sep98.pdf](http://www.scs.carleton.ca/~arpwhite/documents/ieee-cs-sep98.pdf)
- [3] Manoj Kumar Kona and Cheng-Zhong Xu "A Framework for Network Management using Mobile Agents",IEEE/IPDPS-2002 Workshops, April 15-19 2002. [www.ece.eng.wayne.edu/~czxu/paper/iccc01.pdf](http://www.ece.eng.wayne.edu/~czxu/paper/iccc01.pdf)
- [4] B. Pagurek, Y. Wang and T.White "Integration of Mobile Agents with SNMP: Why and How" 2000,IEEE. [www.sce.carleton.ca/netmanage/papers/integration.pdf](http://www.sce.carleton.ca/netmanage/papers/integration.pdf)
- [5] Iwan Adhichandra, Colin Pattinson, Ebrahim Shaghoei, "Using Mobile Agents to Improve Performance of Network Management Operations, PGNet, 16-17 June 2003. <http://www.cms.livjm.ac.uk/pgnet2003/submissions/Paper-12.pdf>
- [6] Tarag Fahad, Sufian Yousef & Caroline Strange "A Study of the Behavior of the Mobile Agent in the Network Management Systems", PGNet 2003. [www.cms.livjm.ac.uk/pgnet2003/submissions/Paper-02.PDF](http://www.cms.livjm.ac.uk/pgnet2003/submissions/Paper-02.PDF)
- [7] Danny B.Lange "Mobile Objects and Mobile Agents: The Future of Distributed Computing", pp 1-12, ECOOP'98. [www.ifs.uni-linz.ac.at/~ecoop/cd/papers/1445/14450001.pdf](http://www.ifs.uni-linz.ac.at/~ecoop/cd/papers/1445/14450001.pdf)
- [8] Damianos Gavalos, Dominic Greenwood Mohammad Ghanbari Mike O' Mahony, "An Infrastructure for Distributed and Dynamic Network Management based on Mobile Agent Technology." [www.simpleweb.org/bibliography/articles/general/gavalas-3.pdf](http://www.simpleweb.org/bibliography/articles/general/gavalas-3.pdf)

- [9] Angelos Michalas, Theodore kotsilieries, Stylianos kalogeropoulos, George Karetzos, Moshe sidi, "Enhancing the Performance of Mobile Agent based Network Management Applications" pp. 432-438 , 2001 IEEE.
- [10] H.Reiser, G.Vogt "Security Requirements for Management Systems using Mobile Agents", pp,160-166, 2000 IEEE.  
[www.mnmteam.informatik.uni-muenchen.de/Literatur/MNMPub/Publikationen/revo00a/PDF](http://www.mnmteam.informatik.uni-muenchen.de/Literatur/MNMPub/Publikationen/revo00a/PDF)
- [11] Paulo Marques, Paulo Simoes, Luis Silva, Fernando Boavida, Joao Silva, "Providing Applications with Mobile Agent Technology", IEEE OPENARCH 2001.  
[http://comet.ctr.columbia.edu/activities/openarch2001/papers2001/OA\\_12.PDF](http://comet.ctr.columbia.edu/activities/openarch2001/papers2001/OA_12.PDF)
- [12] Xiapu Luo, Puliu Yan, Cheng Cheng Guo, Yaguan Tang, "Optimal Placement and Deployment Strategies in Mobile Agent based Network Management" pp, 753-758, 2002 IEEE.
- [13] J.Case, M.Fedor, M.Schoffstall, J.Davin, A Simple Network Management Protocol (SNMP), IETF RFC 1157, 1990. <http://rfc.net/rfc1157.html>
- [14] M. Rose and K. McCloghrie "Management Information Base for Network Management of TCP/IP based Internets MIB-2" RFC 1213, march 1991.<http://rfc.net/rfc1213.html>
- [15] Yemini Y., Goldszmidt G., and Yemini S., "Network Management by Delegation." Proceedings of ISINM '91, Integrated Management II (Krishnan and Zimmer Eds.) pp. 95-97, North Holland Pub., April 1991. [www.simpleweb.org/bibliography/articles/general/gol9110.pdf](http://www.simpleweb.org/bibliography/articles/general/gol9110.pdf)
- [16] Bill Venners. Under the hood: The architecture of aglets. JavaWorld: IDG's magazine for the java community, 2(4), April 1997.

- [17] ObjectSpace, Inc. ObjectSpace Voyager  
<http://www.objectspace.com/developers/voyager/index.html>
- [18] M. Izatt and P. Chan, "Agents: Towards an Environment for Parallel, Distributed and Mobile Java Applications". In Proc. Of the 1999 Java Grande Conference. ACM, 1999.  
[www.cs.ucsb.edu/conferences/java99/papers/13-izatt.pdf](http://www.cs.ucsb.edu/conferences/java99/papers/13-izatt.pdf)
- [19] E Bruneton. Indirection-Free Referencing for Mobile Components. In Proc. Of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Madeira Island, Portugal, April 1999.
- [20] K. Kiniry and D. Zimmerman. "A hands-on look at java mobile agents. IEEE internet computing", 1(4):21-30, July/August 1997.
- [21] Vu Anh Pham and Ahmed Karmouch, University of Ottawa, Ontario "Mobile Software Agents: An Overview", IEEE Communications Magazine, July 1998.  
[www.cs.wpi.edu/~emmanuel/courses/mobile\\_computing/papers/pham\\_mobile\\_agents.pdf](http://www.cs.wpi.edu/~emmanuel/courses/mobile_computing/papers/pham_mobile_agents.pdf)
- [22] Wijnen B., Carpenter G., Curran K., Sehgal A., Waters G. "The SNMP Distributed Protocol Interface", Version 2.0, RFC 1592, March 1994. [www.apps.ietf.org/rfc/rfc1592.html](http://www.apps.ietf.org/rfc/rfc1592.html)
- [23] Daniele M., Wijnen B., and Francisco D., "Agent Extensibility Protocol Version 1", RFC2257, January 1998. [www.apps.ietf.org/rfc/rfc2257.html](http://www.apps.ietf.org/rfc/rfc2257.html)
- [24] M. Rose and K. McCloghrie "Structure and Identification of Management Information for TCP/IP based Internets" RFC 1155, May 1990. [www.ietf.org/rfc/rfc1155.txt](http://www.ietf.org/rfc/rfc1155.txt)
- [25] E. Gamma, R. Helm, R. Johnson, J. Vlissides. "Design Patterns – Elements of Reusable Object-Oriented Software." 1st Edition, Addison-Wesley Longman. 1995.
- [26] IBM Tokyo Research Laboratory. "The Aglets Workbench: Programming Mobile Agents in Java", 1997.

[27] JIPL: A Java Interface for Prolog, K- Prolog.

[http://www.kprolog.com/jipl/index\\_e.html](http://www.kprolog.com/jipl/index_e.html)

[28] Anand Tripathi, Tanvir Ahmed, Sumedh Pathak, and Megan Carney “Design of a Dynamically Extensible System for Network Monitoring using Mobile Agents”

[www.cs.umn.edu/Ajanta/papers/network-monitoring.pdf](http://www.cs.umn.edu/Ajanta/papers/network-monitoring.pdf)

[29] AgentSpace: A Next-Generation Mobile Agent System.

<http://berlin.inesc.pt/agentspace/> -

[30] William Stallings. “SNMP, SNMPv2 and RMON: Practical Network Management” 1st Edition, Addison – Wesley 1999.

[31] Walter Binder, Volker Roth “Secure Mobile Agent Systems Using Java: Where are We Heading?” ACM 2002. [www.igd.fhg.de/~vroth/papers/vroth02b\\_sac.pdf](http://www.igd.fhg.de/~vroth/papers/vroth02b_sac.pdf) -

[32] Françoise Baude, Denis Carómel, Fabrice Huet, and Julien Vayssiére, “Communicating Mobile Active Objects in Java”, pp. 633-643, HPCN 2000.

[33] Alberto Silva, José Delgado ,”The Agent Pattern for Mobile Agent Systems”, 3<sup>rd</sup> European conference on Pattern Languages of Programming and Computing, Euro PloP’98.