NETWORK MANAGEMENT BY MOBILE AGENTS

By

HAIQING MA

A Thesis

Submitted to the Faculty of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of

Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

NETWORK MANAGEMENT BY MOBILE AGENTS

BY

HAIQING MA

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Manitoba requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

ACKNOWLEDGMENTS

While I was finishing this thesis, I received a great deal of help, for which I am most appreciative. This thesis would not have been possible without that support. I would like to take this opportunity to thank all the people who have contributed towards this thesis.

First of all, I would like to express my thanks to my supervisor R. D. McLeod for his encouragement and sound judgement, and for providing his thoughtful guidance. On a technical note, this thesis benefited enormously from contributions made by Dr. David C. Blight over the past two years. I would like to thank Dave for introducing me to this very interesting research world and for making key contributions to the project.

Next, a special thanks to Dr. Jose Rueda, for spending his considerable time on discussions and recommendations for my research. My thanks also go to Wenbo Sheng, for his discussion and suggestions on this thesis.

In addition, I would like to thank Dr. Randal J. Peters, for taking the time to be on my thesis committee, to read my thesis, and all his effort during this thesis process.

Furthermore I would like to acknowledge the Telecommunications Research Laboratories (TR*Labs*) for their financial support and providing facilities for this research. TR*Labs* provided good opportunities for studying and an excellent research environment.

A special thanks is due to my family – my mother Yinfo Kang, my father Zengrong Ma, my younger sister Yiqing Ma and her smart baby boy, Jeff Ma Sheng – for their encouragement and care.

Finally, my deepest thanks to my wife Liping Cheng for her great support and motivation, and our amazing daughter, Cindy.

ABSTRACT

Recently a new technique for distributed computing software on networks referred to as "mobile agents" has attracted much research attention. A mobile agent is an independent software entity which exists in a software environment. It is an autonomous, intelligent program that moves through a network, searching for and interacting with services on the user's behalf. In this thesis, applications of mobile agents in network management is investigated. Two research areas are addressed, one is network monitoring and the other is network signaling.

A network management system (NMS) was developed combining mobile agents, SNMP, Java and web browser technologies. The mobile agent's attributes, such as mobility, delegation, communication, persistence, fault tolerance and independence of platform, are realized in the system. A working demonstration of the system has illustrated a strong variety of capabilities in monitoring network elements.

Also implemented were two resource negotiation models for network signaling by mobile agents in a simulated environment. In model 1, all agents involved in establishing a connection will go to one central place for network resource negotiation, but in model 2, only the call initiating agent will go to each intermediate node to negotiate resources with a local agent. Two message-based systems were also implemented for comparison with their corresponding mobile agent models. The resource negotiation uses concepts of economic consumer surplus and burstiness model. The test results show that the mobile agent methods are better than the message-based methods in terms of negotiation time under normal conditions. In addition, the mobile agent of model 1 is better than that of model 2.

ABBREVIATIONS

ASN.1	Abstract Syntax Notation 1
ATM	Asynchronous Transfer Mode
BER	Basic Encoding Rules
B-ISDN	Broadband ISDN
CAC	Connection Admission Control
CCITT	Comité Consultative Internationale de Telegraphique et Telephonique
CLNS	Connectionless Network Service
CMIT	Common Management Information Protocol
CORBA	Common Object Request Broker Architecture
DPE	Distributed Processing Environment
GC	Garbage Collection
IAB	Internet Architecture Board
IPX	Internet Packet Exchange
ISDN	Integrated Service Digital Network
ITU	International Telecommunication Union
IETF	Internet Engineering Task Force
JDK	Java Development Kit
KQML	Knowledge Query and Manipulation Language
LAN	Local Area Network
MIA	Mobile Intelligent Agent
MIB	Management information base

MTBF	Mean Time Between Failure
MTTR	Mean Time To Repair
NMF	Network Management Framework
NMS	Network Management System
OSI	Open Systems Interconnect
OWRC	Optimized Weighted References Counting
QoS	Quality of Service
PSM	Persistent Storage Manager
PVC	Permanent Virtual Circuit
RFC	Requests for Comment
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SVC	Switch Virtual Circuit
TCP	Transmission Control Protocol
TINA	Telecommunications Information Networking Architecture
TMN	Telecommunications Management Network
UDP	User Datagram Protocol
VC	Virtual Connection
VPN	Virtual Private Networks
WAN	Wide Area Network
WRC	Weighted Reference Counting

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

1.1 Mobile Agents

1.1.1 Definition of Mobile Agents

In general, an agent means a person or business authorized to act on another's behalf. Software agents acting on behalf of another entity is argued to be the fundamental property of software agents [Shaw 97]. The software agents discussed here also exhibit a second fundamental agent characteristic, namely, they both enjoy at least a variable degree of autonomy. A third important aspect of an agent's behavior is the degree of proactivity and reactivity present in their behavior. The agent can display high amounts of both proactivity and reactivity at different times. Finally, agents also exhibit some level of a number of attributes, the key ones of which are learning and co-operation. In essence, therefore, we believe that the concept of agent can be summed up by the following definition:

An agent is a computational entity which:

- acts on behalf of other entities in an autonomous fashion
- performs its actions with some level of proactivity and/or reactiveness
- exhibits some level of the key attributes of learning and co-operation.

A mobile agent is a software entity which exists in a software environment. It inherits some of the characteristics of a software agent (as defined above). A mobile agent must contain all of the following models: an agent model, a life-cycle model, a computational model, a security model, a communication model and finally a navigation model. Mobile

agents are a special kind of mobile objects. They are autonomous, intelligent programs that move through a network, searching for and interacting with services on the user's behalf. They have behavior, state, and location. Mobile agents are autonomous because they can decide where they will go and what they will do. They can control their lifetimes, decide whether or not to comply with external requests and decide to perform actions, such as travel across a network to a new computer, independent of any external request. The mobile agent can also halt itself, ship itself to another computer on the network, and continue execution at the new computer. It doesn't restart execution from the beginning at the new computer; it continues where it left off. Mobile agents can migrate from space to space carrying their states with them. The space is a server of some kind and is also an object where agents travel to. As opposed to mobile agents, spaces are static objects. Once an agent is accepted at a space, it is loaded into a space where it can execute. Agent execution is subject to resource availability and security constraints that spaces impose on them.

1.1.2 Advantages And Disadvantages of Mobile Agents

According to some recent research reports, mobile agents have several advantages over the traditional client/server models or message-based models.

1. Efficiency:

   In some certain situations, mobile agents consume fewer network resources since they move the computation to the data rather than the data to the computation.

2. Resource Utilization:

   If an agent requires facilities such as persistence or a fast processor, it can move to a machine that has these facilities.

3. Fault tolerance:

   Increasing the fault tolerance and robustness of the management solutions by reducing the dependency of the management operations on the remote network connections while performing analysis tasks. Mobile agents do not require a continuous connection between machines when they are working. If the machine that contains an agent is about to become disconnected from the network, the agent can move to another machine to continue its execution.

4. Convenient paradigm:

   Mobile agents hide the communication channels but not the location of the computation.

5. Customization:

   Mobile agents allow clients and servers to extend each other's functionality by programming each other.

6. Reduced network load:

   Mobile agents have the ability to reduce network load by migrating to the most appropriate host on which to perform its processing. By performing all data transactions at the server, the mobile agent paradigm can substantially decrease network load compared to traditional implementations, especially in a large distributed data oriented system.

7. Adaptive way to update software:

   Each piece of management software in a managed environment can frequently become outdated, e.g., because a bug was discovered and fixed or because a new environment requires different or extended functionality. Using mobile agents the functionality within an agent can be easily replaced by a new version of the software from a remote site.

The reduced network load may not be true under certain conditions. It has been argued that remote programming (including migration) achieves a better performance than client/server interaction. However, this claim is highly questionable as the migration of agents also causes a communication overhead. Which of the two approaches provides better performance strongly depends on the interaction patterns and the "size" (code plus state information) of the agent.

There are several drawbacks associated with mobile agents as follows:

1. At present, there are not too many operating companies that would allow autonomous mobile agents to float inside their networks. The lack of network devices supporting direct execution of mobile agents, which require dedicated and often heavy run-time support, is today's major obstacle to the exploitation of code mobility.
2. How do agents represent their knowledge and communicate their needs to other agents? The "meeting" and "collaborate" mechanisms available in some of today's systems are extremely limited. Likewise, the static interfaces supported by today's agent communication mechanisms limit the flexibility and possibilities for agent

interaction. Some of the most promising work in this domain is the Knowledge Query and Manipulation Language (KQML), and its related efforts, languages, and systems (like KIF, Ontolinguia, and others) from the Knowledge Sharing Effort Consortium. KQML is both a message format and message-handling protocol designed to support run-time knowledge sharing between agents. Only with the use of technologies like KQML can mobile agent systems hope to reach the complexity level that will be demanded from them in the very near future - however, the level of complexity involved in KQML implementations makes the integration of KQML and agent systems problematic.

3. Since commercial agent systems have only recently begun to support fully autonomous multi-hop agents, little work has been done on issues of security in such systems. While the Java sandbox and security models go a long way toward solving the rogue agent problem, (i.e. an agent damaging a system or network), very little work has been done to solve the inverse problem, that of an agent being attacked by the host computer.
4. Since agents represent actions of real human beings, the standard computer security problems of authentication, trust, and culpability, especially in commercial environments, are tremendous problems which these agent systems will need to solve to the satisfaction of both their customers and their lawyers.

1.1.3 The Usage of Mobile Agents

Mobile agents could be used in following fields:

1. Users could use them to find resources and complete tasks for them while they are off-line.
2. They could be used in network management to notify appropriate people when something breaks down.
3. Mobile agents could be used for data collection from many places. One natural application of mobile agents is collecting information spread across many computers connected to a network.
4. Mobile agents could be used for searching and filtering. On behalf of a user, a mobile agent could visit many sites, search through the information available at each site, and build an index of links to pieces of information that match a search criterion.
5. Mobile agents could be used for monitoring application. This kind of application highlights the asynchronous nature of mobile agents. When an agent is sent, it is not necessary to wait for the results of its information gathering. An agent can be programmed to wait as long as it takes for certain information to become available. Also, you needn't stay connected to the network until an agent returns. An agent can wait until reconnection to the network before making its report.
6. Mobile agents could be used for targeted information dissemination. Another potential use of mobile agents is distributing interactive news or advertising to interested parties.
7. Mobile agents could be used for negotiating. Besides searching databases and files, agents can gain information by interacting with other agents. For example, to schedule a meeting with several other people, one mobile agent could be sent to interact with the representative agents of each of the people invited to the meeting.

The agents could negotiate and establish a meeting time. In this case, each agent contains information about its user's schedule. To agree upon a meeting time, the agents exchange information.

8. Mobile agents could be used for bartering. Electronic commerce is another good fit for mobile agent technology. A mobile agent could do your shopping, including making orders and potentially payment. Electronic commerce also can take place between agents.
9. Mobile agents could be used for parallel processing. Given that mobile agents can move from node to node and can spawn subagents, one potential use of mobile agent technology is as a way to administer a parallel processing job. If a computation requires so much CPU time as to require breaking up across multiple processors, an infrastructure of mobile agent hosts could be an easy way to get the processes distributed.
10. Mobile agents could be used for entertainment. In this scenario, agents represent game players. The agents compete with one another on behalf of the players.

As such, mobile agent systems may become popular in several practical applications. In my assessment however, mobile agents will not replace message-based systems, but could improve message-based methods when they are combined with mobile agent methods.

1.2 Network Management

The most common framework depicted in network management design is centered around the Open Systems Interconnect (OSI) "FCAPS" model [Douglas 95]. However, most network management implementations do not really cover all of these areas. They are listed and explained as follows:

Fault Management

Configuration Management

Accounting Management

Performance Management

Security Management

Chargeback Management

Systems Management

Cost Management

- Fault Management

Fault management is the detection of a problem, fault isolation and correction to normal operation. Most systems poll the managed objects to search for error conditions and illustrate the problem in either a graphic format or a textual message. Most of these types of messages are setup by the person configuring the polling on the Element Management System. Some Element Management Systems collect data directly from a log printer type output receiving the alarm as it occurs. Fault management deals most commonly with events and traps as they occur on the network.

- Configuration Management

Configuration management is probably the most important part of network management in that a network can not be accurately managed unless the configuration of the network can be managed. Changes, additions and deletions from the network need to be coordinated with the network management system's personnel. Dynamic updating of the configuration needs to be accomplished periodically to ensure the configuration is known.

- Accounting Management

The accounting function is usually left out of most implementations in that LAN based systems are said not to promote accounting type functions until one gets into mainframe hosts such as an IBM mainframe or Digital VAX. Others rationalize that accounting is a server specific function and should be managed by the system administrators.

- Performance Management

Performance is a key concern to most network support people. Although it is high on the list of management function, it is considered difficult to be factual about some LAN performance issues unless they employ RMON technology. Although RMON Pods are very useful, one should carefully weigh what is pertinent against what can be accomplished in other ways without having to incur the cost of RMON technology. Performance of Wide Area Network (WAN) links, telephone trunk utilization, etc., are areas that must be revisited on a continuing basis as these are some of the areas easiest to optimize and realize savings. System or application performance is another area in which

optimization can be accomplished but most network management applications don't address this in a functional manner.

- Security Management

Most network management applications only address security applicable to network hardware such as someone logging into a router or bridge. Some network management systems have alarm detection and reporting capabilities as part of physical security (contact closure, fire alarm interface, etc). None really deal with system security as this is a function of system administration.

- Chargeback Management

Chargeback has been done for years in the large mainframe environments and will continue to be accomplished as it is a way to charge the end user for only the specific portion of the service that he or she uses. Chargeback on Local Area Networks presents new challenges in that so many services are provided. In many implementations, chargeback is accomplished on the individual Server providing the service. While chargeback is very difficult on broadcast based networks such as Ethernet, it is realizable on networks that dynamically allocate bandwidth as the end users' needs dictate (ATM). As technology associated with monitoring LAN and WAN networks evolves, chargeback will be integrated into more and more systems.

- Systems Management

Systems Management is the management and administration of services provided on the network. A lot of implementations leave out this very crucial part although it is one of the areas in which network management systems can show significant capabilities, streamline business processes, and save the customer money with little effort. There are many good products available to automate system administration functions and these products can be easily integrated into the overall network management system very easily.

- Cost Management

Cost management is an avenue in which the reliability, operability and maintainability of managed objects are addressed. This one function is an enabler to upgrade equipment, delete unused services and tune the functionality of the Servers to the services provided. By continuously addressing the cost of maintenance, Mean Time Between Failure (MTBF), and Mean Time To Repair (MTTR) statistics, costs associated with maintaining the network as a system can be tuned.

1.2.1 Network Management Protocols

There are several organizations which have developed services, protocols and architectures for network management. The three most important organizations are [Pras 95]:

- The International Organization for Standardization (ISO).

- The Comité Consultative Internationale de Telegraphique et Telephonique (CCITT); this organization is now called the Telecommunication Standardization Sector (T) of the International Telecommunication Union (ITU).
- The Internet Engineering Task Force (IETF).

Of these three ISO was the first who started, as part of its `Open Systems Interconnection' (OSI) program, the development of an architecture for network management. The first proposals for such an architecture appeared during the early 1980; nowadays a large number of standards exist for the architecture as well as for network management services and protocols. Of these standards the "OSI Management Framework", the "OSI Systems Management Overview" and the "Common Management Information Protocol" (CMIP) are probably the best known examples.

Initially the aim of ISO was to define management standards for datacom networks; development of management standards for telecom networks was left to CCITT. In 1985 CCITT started the development of such management standards; these standards have become known as the "Telecommunications Management Network" (TMN) recommendations. Originally these recommendations were self standing, but during the 1988-1992 study period they were rewritten to include the ideas of OSI management. Presently, OSI management and TMN can be seen as complements to each other.

The growth of the Internet has played a decisive role in the development of network management protocols. Initially the Internet Architecture Board (IAB) intended to apply

the OSI management approach, but the size of the Internet had reached a level at which management became indispensable. The IAB requested the IETF (the organization responsible for the development of Internet protocols) to define an ad hoc management protocol. This "Simple Network Management Protocol" (SNMP) was completed within a year and soon many manufacturers started the production of SNMP compliant systems. Although SNMP has several deficiencies, it has become the de facto standard for management of datacom networks. In 1993 an attempt was made to tackle these deficiencies and an improved version of SNMP (SNMPv2) appeared.

With SNMP, a single manager may control many agents. The SNMP protocol is built upon the User Datagram Protocol (UDP), which is a connectionless transport protocol. Since the Internet management information as well as the formats of SNMP PDUs are defined according to the Abstract Syntax Notation 1 (ASN.1) syntax, encoding functions are needed immediately on top of UDP. These functions operate according to the Basic Encoding Rules(BER). Five types of SNMP PDUs are defined: GetRequest, GetNextRequest, SetRequest, Response and Trap. In this thesis, I use the SNMP protocol for network management, so in the following section, a brief explanation of SNMP is given.

1.2.2 Simple Network Management Protocol (SNMP)

There are two versions of SNMP: Version 1 and Version 2. Most of the changes introduced in Version 2 increase SNMP's security capabilities. Other changes increase interoperability by more rigorously defining the specifications for SNMP

implementation. SNMP's creators believe that after a relatively brief period of coexistence, SNMP Version 2 (SNMPv2) will largely replace SNMP Version 1 (SNMPv1). SNMP is part of a larger architecture called the Internet Network Management Framework (NMF), which is defined in Internet documents called requests for comments (RFCs). The SNMPv1 NMF is defined in RFCs 1155, 1157, and 1212, and the SNMPv2 NMF is defined by RFCs 1441 through 1452.

SNMP is part of the Internet network management architecture. This architecture is based on the interaction of many entities, as described in the Internet RFCs and other documents. A network management system is comprised of the following components:

- Network elements -- Sometimes called managed devices, network elements are hardware devices such as computers, routers, and terminal servers that are connected to networks.
- Agents -- Agents are software modules that reside in network elements. They collect and store management information such as the number of error packets received by a network element.
- Managed object -- A managed object is a characteristic of something that can be managed. For example, a list of currently active TCP circuits in a particular host computer is a managed object. Managed objects differ from variables, which are particular object instances. Using our example, an object instance is a single active TCP circuit in a particular host computer. Managed objects can be scalar (defining a single object instance) or tabular (defining multiple, related instances).

- Management information base (MIB) -- A MIB is a collection of managed objects residing in a virtual information store. Collections of related managed objects are defined in specific MIB modules.
- Syntax notation -- A syntax notation is a language used to describe a MIB's managed objects in a machine-independent format. Consistent use of a syntax notation allows different types of computers to share information. Internet management systems use a subset of the International Organization for Standardization's (ISO's) Open System Interconnection (OSI) Abstract Syntax Notation 1 to define both the packets exchanged by the management protocol and the objects that are to be managed.
- Structure of Management Information (SMI) -- The SMI defines the rules for describing management information. The SMI is defined using ASN.1.
- Network Management Stations (NMSs) -- Sometimes called consoles, these devices execute management applications that monitor and control network elements. Physically, NMSs are usually engineering workstation-caliber computers with fast CPUs, megapixel color displays, substantial memory, and abundant disk space. At least one NMS must be present in each managed environment.
- Parties -- Newly defined in SNMPv2, a party is a logical SNMPv2 entity that can initiate or receive SNMPv2 communication. Each SNMPv2 party comprises a single, unique party identity, a logical network location, a single authentication protocol, and a single privacy protocol. SNMPv2 messages are communicated between two parties. An SNMPv2 entity can define multiple parties, each with different parameters. For example, different parties can use different authentication and/or privacy protocols.

- Management protocol -- A management protocol is used to convey management information between agents and NMSs. SNMP is the Internet community's de facto standard management protocol.

SNMP itself is a simple request/response protocol. NMSs can send multiple requests without receiving a response. Six SNMP operations are defined:

- Get -- Allows the NMS to retrieve an object instance from the agent.
- GetNext -- Allows the NMS to retrieve the next object instance from a table or list within an agent. In SNMPv1, when an NMS wants to retrieve all elements of a table from an agent, it initiates a Get operation, followed by a series of GetNext operations.
- GetBulk -- New for SNMPv2. The GetBulk operation was added to make it easier to acquire large amounts of related information without initiating repeated get-next operations. GetBulk was designed to virtually eliminate the need for GetNext operations.
- Set -- Allows the NMS to set values for object instances within an agent.
- Trap -- Used by the agent to asynchronously inform the NMS of some event. The SNMPv2 trap message is designed to replace the SNMPv1 trap message.
- Inform -- New for SNMPv2. The Inform operation was added to allow one NMS to send trap information to another.

SNMP is the de facto standard communications protocol supporting integrated network management in heterogeneous environments. With a wide array of SNMP management features, it provides truly useful management functionality for network management.

1.3 Network Management by Mobile Agents

In centralized network management, some disadvantages are as follows [Baldi 97]:

- Limitation of the scalability

The IETF and ISO approaches are characterized by high centralization, since their architecture puts almost all the computational burden on the management station. Centralization has proven to seriously limit the scalability of network management. As the dimension of the network grows, the management station has to communicate with a larger number of devices, and to store and process an ever increasing amount of data. This leads to the need for high cost hardware dedicated to the management station, poor performance, or even to the impossibility to cope with the dimension of the network. In some cases, a separated network is employed to carry management data, but in most cases in-band management is employed, that is, the very network that is being managed is used to carry management information. Consequently, the area of the network around the management station experiences heavy traffic due to the combination of messages sent around by the management station and those containing data from the devices. The worst shortcomings of the centralized in-band approach show up during periods of heavy congestion, when management intervention is particularly important. In fact, during these periods:

i) The management station increases its interactions with the devices and possibly downloads configuration changes, thus increasing congestion

ii) Access to devices in the congested area becomes difficult and slow (sometimes even impossible), and

iii) Congestion, as an abnormal status, is likely to trigger notifications to the management station which worses congestion.

- Server flexibility

In Client/Server applications, the set of services offered by the server is fixed, defined a priori by the application designer, and accessible through interfaces defined statically. The services provided or the particular interface may not be suitable for different unforeseen user needs. In addition, evolution of the application usually involves high-cost activities and may limit availability. For instance, in the network management approaches described so far, if the protocol, the information base structure, or the functionality of the agent is changed, the static agent has to be extended and rebuilt. In other words, there is no way to dynamically extend the capabilities of the server side, in terms of code describing its actual behavior.

- Bandwidth Usage

Particular tasks involving continuous and intensive Client/Server interaction through the network may result in bandwidth waste. For instance, monitoring stock information maintained by a server may require continuous polling by the client. This is a major problem of micro management in SNMP where the management station polls the agent continuously, thus wasting bandwidth and, even worse, increasing network load.

- Less Fault Tolerance

Centralized management can be realized in an implicit as well as an explicit way. But the disadvantage of centralized management is that the entire network may get out of control after failure of a single manager.

By using a mobile agent system, several benefits in network management can be identified:

- Asynchronous autonomous interaction:

  Tasks can be encoded into mobile agents and then dispatched. The mobile agent can operate asynchronously and independent of the sending program. An example of this would be a mobile device dispatching an autonomous search agent onto the fixed network, disconnecting, and reconnecting some time later to report the results of the search.

- Interaction with real-time entities:

  Real-time entities such as software controlling an ATM switch or a safety system in a nuclear installation require immediate responses to changes in their environment. Controlling these entities from across a potentially large network will incur significant latencies. For critical situations (nuclear system control) such latencies are intolerable. Mobile agents offer an alternative. They can be dispatched from a central system to control real-time entities at a local level and also process directives from the central controller.

- Asynchronous and cooperative processing of tasks:

  The possibility of delegating specific tasks by means of mobile agents to one specific or even multiple nodes allow for highly dynamic and parallel computations.

- Decentralization of management:

  Mobile agents allow systems to decrease pressure on centralized network management systems and network bandwidth by delegating specific management tasks from the central operations system to dispersed management agents. Mobile agents representing management scripts enable both temporal distribution (i.e. distribution over time) and spatial distribution (i.e. distribution over different network nodes) of management activities.

Network Management Systems (NMS) typically operate on large, heterogeneous, and asynchronous networks, and they must provide features including security, distributed control, portability, and dynamic operation in response to failures of system components. The high level of complexity inherent in an NMS requires a correspondingly high level of abstraction within the development and methodology of the NMS software.

The use of agents is ideally suited for an NMS. In fact SNMP, CMIP, and TNM use the concepts of OO design and agent technology. Moreover, intelligent agents have been demonstrated performing some network management activities already in commercial packages. For example, agent based packages have been implemented to manage and to perform operating system maintenance on networks, and to monitor specific devices in the networks. Generally, mobile agents enable dynamic placement of control and

management software processes at the most appropriate locations within the telecommunications environment. However, agent technology will probably not replace traditional client/server computing entirely. Rather mobile agents should be regarded as an "add on" to existing computing/service platforms, providing more flexibility for the realization of services within next generation telecommunications environments.

The use of mobile agents in the management of large networks is still in its early conceptual stages. Ideally, agents should be created using a language which is portable between network platforms, mobile, and autonomous in their operation. This would allow reliable agents to be created which implement an NMS system without requiring vendor implementation or support. This will greatly increase the flexibility, maintainability, and versatility of the NMS system.

This thesis investigates whether mobile agents can be applied to network management and compares them to traditional methods. The applications of mobile agents in network management, their advantages and disadvantages were investigated. The focus was on the two fields: network element monitoring through SNMP and network signaling by mobile agents. In these two research fields, the mobile agent shows strong capability and advantages. The implementation results also show that mobile agent approach is a better approach than traditional message-based methods.

Following this chapter, chapter two explains the working mechanism of mobile agents. Chapter three is the literature survey, it investigates some applications of mobile agents.

Chapter four shows the work of mobile agents in monitoring network element. Chapter five discusses the network signaling by mobile agents. Finally chapter six summarizes the work and suggests the future work.

CHAPTER 2 WORKING MECHANISM

2.1 Introduction

Many working principles of mobile agent systems are derived from distributed object systems. Mobile agents developed through distributed objects. They are based on distributed objects adding autonomy and intelligence to the object. That is, a mobile agent is a special kind of object. As such, it is necessary to understand a distributed object's working principles and extensions for mobile agents. In this chapter, both of them will be introduced and their main features and differences presented.

A distributed object-oriented system is composed of many objects that can reside anywhere on the network. These objects interact by exchanging messages with each other. Distributed object management is management of "objects" that are distributed across a number of sites. One feature of a distributed object system is that objects can move from time to time among network nodes, which means the objects could be mobile objects. Object migration involves moving an object from one node to another on the network. The original node on which the object was located is called its source and the final node is called its destination. The migrated object then becomes local to the destination and the destination can invoke methods on the object locally without resorting to any remote invocations.

There are several migration policies such as copying, replication and complete movement. Copying an object involves making a copy of the object at the destination. The original object and the copied object do not share any resources in common; the

copied object gets a new identity. Replication is copying an object but in this case: the target and the replicated object share the same identity. Updates to a replicated object should be reflected in the target and vice-versa. Replication requires concurrency control strategies on the object involved. An object can also be completely migrated from the source to the destination. In this case, the target object will disappear on the source and be identified with the same identity on the destination.

2.2 Unit of Migration

In the system where the state is separated from the methods, it is possible to consider moving the object's state without moving the methods. The counterpart of this scenario in purely behavioral systems is the fragmentation of an object according to its behaviors. In either case, the application of methods to an object requires the invocation of remote procedures [Ozsu 94]. This method is not suitable for mobile agents society because of its own characteristics. The mobile agent systems treat the object as the unit of distribution. When moving an object, both state and operations must be moved - this is their smallest unit of migration. But there still exist some problems: Does the system need to migrate the objects which will be referenced by the mobile agents and how many of them should be moved?

2.2.1 Object Graph

Object graphs are introduced which describe the dependencies between objects in the form of containment, where one object is contained within another object, or reference where one object contains a reference to another. For example, a data-viewer object can

contain several window frame objects and the window frame objects could also contain several data vector objects. These dependencies can be modeled using a graph as shown in Figure 1.

Such a graph can be used to propagate the changes made in any of the objects involved to all its dependents recursively. Object graphs introduce special problems when objects are migrated or for implementing object persistence. There are also several problems that need to be addressed while deciding the locations of objects within a graph since they need to interact frequently. In an object graph, we can let some objects reside on node X and some other objects on node Y. This introduces the notion of migration of objects within a graph and exerting control restrictions on these objects. There are no easy mechanisms to decide which group of objects in a graph should be migrated. These decisions are in general dependent on the application semantics.

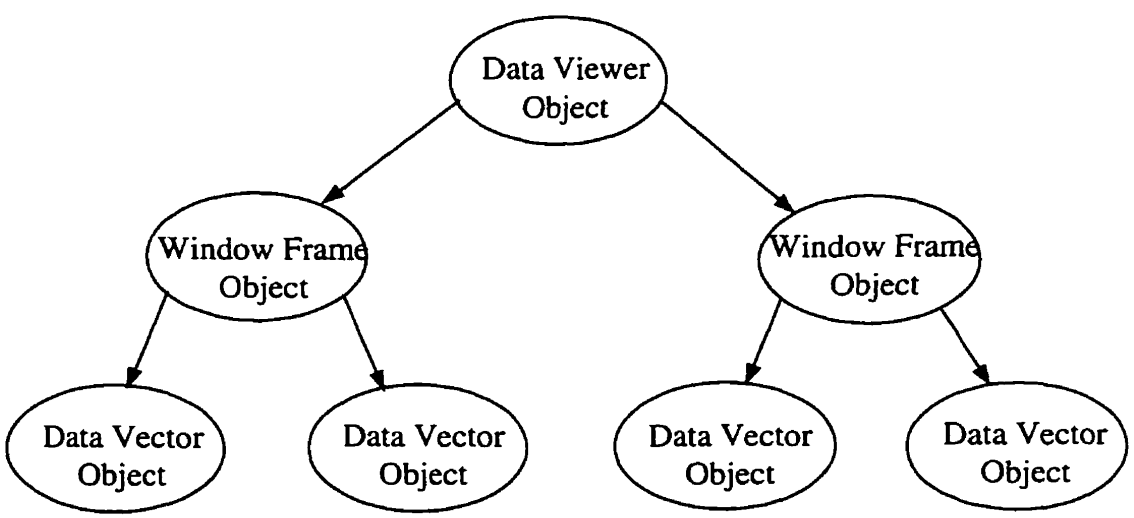


Figure 1: Example of Object Graph

2.2.2 Degree of Migration Control

Objects in a distributed system do not exist in isolation. In general they will be related to other objects resulting in a graph of objects. These graphs introduce special problems for migration. In this section, we consider the object graph model in [Machiraju 97] to explain the policies in the distributed object field and mobile agent field.

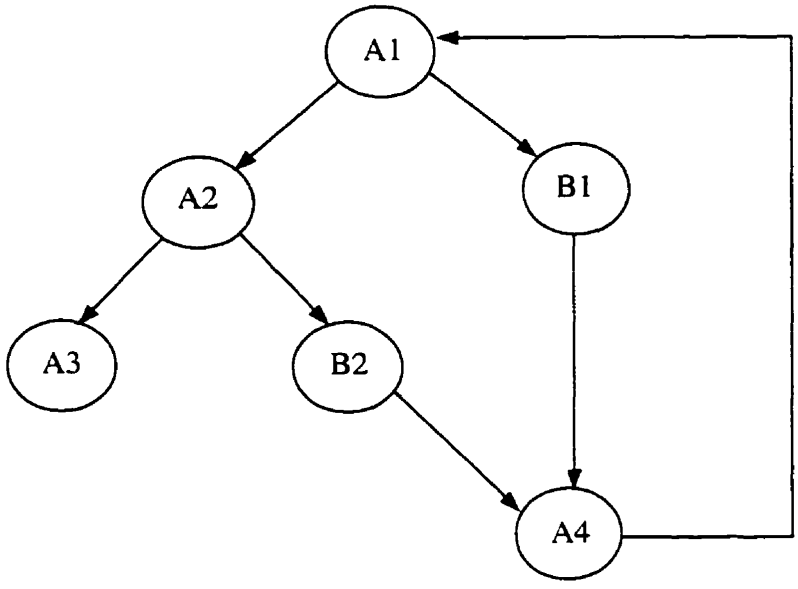


Figure 2: An Object Graph

The graph of Figure 2 is composed of objects of two types - A and B. A1, A2, A3 and A4 are objects of type A while B1 and B2 are objects of type B. Every object points to one or two objects except for the leaves.

There are some migration policies in distributed object management:

1) Shallow migration

Under the shallow migration policy, only one object is migrated at a time. The other objects that it refers to are migrated only when they are required.

2) Deep migration

In deep migration, all the objects that can be reached from the root object are migrated immediately. When an object in a graph is migrated, the objects that it points to should also be migrated. In Figure 2, when object A1 is migrated, object A2 and B1 should also be migrated so that the two pointers of A1 point to valid objects when they are referenced. This means that the entire graph has to be moved.

3) Layer migration

One can migrate all the objects that can be reached from the root within a certain layer. For example, if we migrate A1 on Figure 2 with a layer of 1, the objects A2 and B1 will be migrated.

4) Class boundaries

Sometimes the application demands that objects of a particular type be migrated whereas objects of other types be left at the source. For example, if A1 migrates and we set a class boundary of migration at objects of class B, then objects A1, A2 and A3 will be migrated initially. The rest of the graph will be migrated when it is required.

5) Diffusion model

In this case, class is associated with a migration power between 0 and 1. The migration is initiated with an initial power of 1. When an object of a class is migrated, the initial power decreases by the power of class. When the initial power decreases to a value of 0

the migration stops. For example, if A1 migrates with a power of 0.6 associated with objects of class A and a power of 0.3 associated with objects of class B, then A2, B1, B2 and A4 will migrate with A1.

In a mobile agent system, the situation is different. We can not apply all these methods to this field. In a mobile agent system, we should distinguish "contain" and "reference". We could consider the following cases with Figure 2 as an example:

1) If agent A1 contains all other objects (agents can not contain other agents, because an agent is an independent program process), when A1 migrates, all other objects should be migrated with A1 such as in deep migration. Because all other objects contained by A1 belongs to agent A1, we have to treat the agent as a smallest entity and can not separate it otherwise the agent could not work as an independent process (for example, working off-line).

2) In Figure 2, if all objects are agents and agent A1 references them, when agent A1 migrates, it could not order other agents to the destination because mobile agents are autonomous, they can decide where they will go and what they will do. They can decide whether or not to comply with external requests and decide to perform actions. So agent A1 can only negotiate with other agents and let them decide by themselves.

3) In a "pure" mobile agent system, there doesn't exist any object which doesn't belong to any agent. That means every object is contained by one agent, when this agent

moves, all objects belonging to that agent should migrate with it. In this situation, the management of objects migration is easy to implement and there is no degree of migration control issue.

4) In a "hybrid" mobile agent system, there exist some independent objects which don't belong to any agent, but are referenced by an agent or object. In this case, the policies taken by distributed object management could be taken into account. However it seems to in efficient because of the complexity of the situation. The method used by Emerald [Jul 88] may be better. Emerald allows the programmer to specify explicitly which objects move together. For this purpose, one could attach objects to an agent or other objects. When an object is declared, the programmer can specify the object to be an "attached object". When an agent or object migrates, the attached object will go with it together.

So in mobile agent systems, the agent migration control is not difficult but object migration in a hybrid environment is complex and needs to be considered in detail.

2.3 When to Migrate

In a distributed object system, the migration of objects should depend on what state they are in during the time of migration [Dollimore 94]. Objects can be in one of four states: ready, active, waiting, or suspended. Ready objects are those that are not currently invoked or have not received a message but are ready to be invoked or to receive a message. Active objects are those which are currently involved in an activity in response to an invocation or a message. Waiting objects are those which have been invoked or

have sent a message to another object and are waiting for a response and suspended objects are temporarily unavailable for invocation. Objects in active or waiting states are not allowed to migrate since the activity they currently involved in would be broken.

In mobile agent systems, the situation become more complex. The mobile agents are autonomous and they can decide when they move. An object can halt itself, ship itself to another computer on the network, and continue execution at the new computer. It doesn't restart execution from the beginning at the new computer; it continues where it left off. But in practical implementations, this goal is very difficult to reach. The main problem is how to checkpoint and transfer the intermediate state accumulated on previous nodes. There are several solutions [Milojicic 96] with various degrees of difficulty. The simple method is to allow an agent to start movement only after its execution is completed and then re-initialize it on the new node. The second method could allow an agent to migrate by invoking a dedicated method on behalf of itself. The most difficult method would allow agent migration at any point of time.

The simple migration only allows a "quiescent" agent to move between nodes such in the ready state of a distributed system. A "quiescent" agent is not communicating with anyone and not being accessed by other agents. The method has three steps: (1) checkpointing the agent state, realized by providing a "move (destination-node)" method, (2) transferring the state to another node, and (3) continuing executing at the point just after the location where the checkpointing method was invoked on the source node. There have been two solutions to implement this method as illustrated in Figure 3. The complex

way (A) is to allow an agent to invoke a "move" method at any place within the code. The simple way (B) is that "move" method can be invoked at the end of the code and the agent restarts at the beginning of another code.

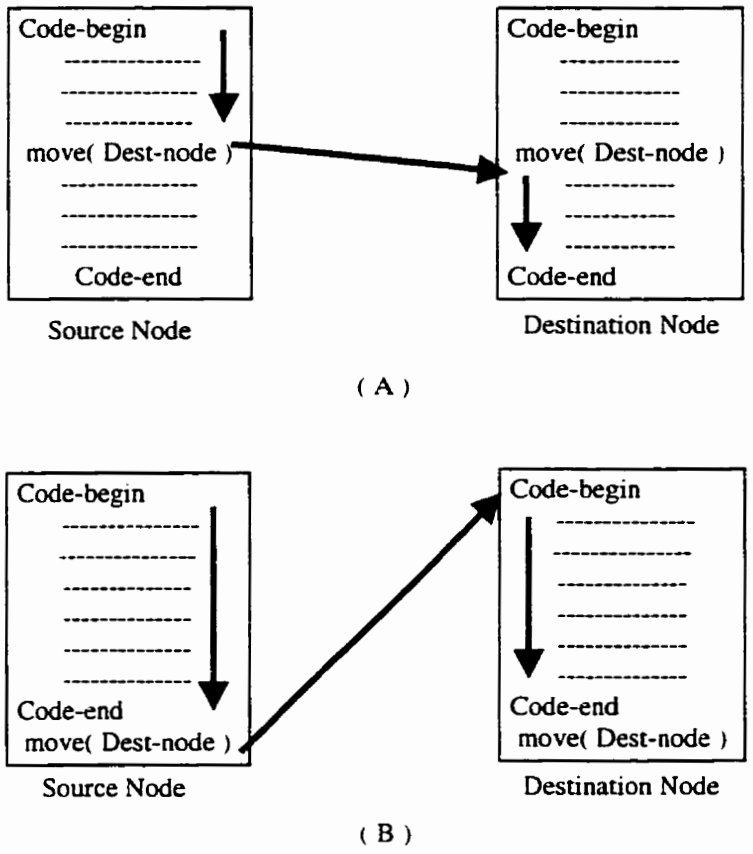


Figure 3: Two Solutions for Migration

Supporting open channels will greatly improve the practical performance of mobile agents. Open channels allow agents to continue communication with the rest of the world even after they move to another node. Prior to migration, an agent could have open channels with other services or agents. After migrating to another node, the agent could still continue the communicating activity already underway. Communication channels that an agent has opened on a node A (see Figure 4), have to be reopened after it is

moved to node B. The servers and agents on other nodes should not recognize the act of migration. There are a couple of approaches to preserve open channels, each with varying degrees of difficulty. The first possibility is that prior to migration an agent closes all open connections and re-opens them after migration. In order to prevent the loss of messages during migration, a proxy agent is left on the original node that accepts incoming messages and then forwards them to the new location of the migrated agent. The second method is to refuse messages during migration and require senders after some period of time to retransmit. In this case besides transferring state, migration must also update communication links between the mobile agent and other agents or servers.

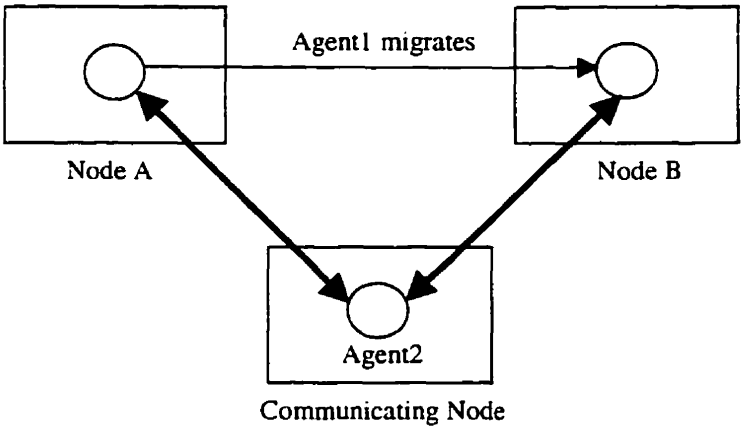


Figure 4: Supporting Open Channel During Migration

2.4 Tracking Management

Various mechanisms are used to track object migration in a distributed system [Milojicic 96] and [Ingham 96]. The object migration diagram of Figure 5 is used to explain the schemes.

The four schemes are as follows:

1) Forwarding Scheme

- replace migrated object with forward reference
- invocations are redirected transparently to the new location
- references are updated through piggy-backed information

2) Searching Scheme

- do nothing at migration time
- object fault occurs at next invocation through each reference
- object must be located through the search

3) Updating Scheme

- callback to all references at migration time
- no further work necessary at invocation time

4) Registering Scheme

- update name-server at migration time
- object fault occurs at next invocation through each reference
- query name-server to determine new location of object
- reference invokes object at new location

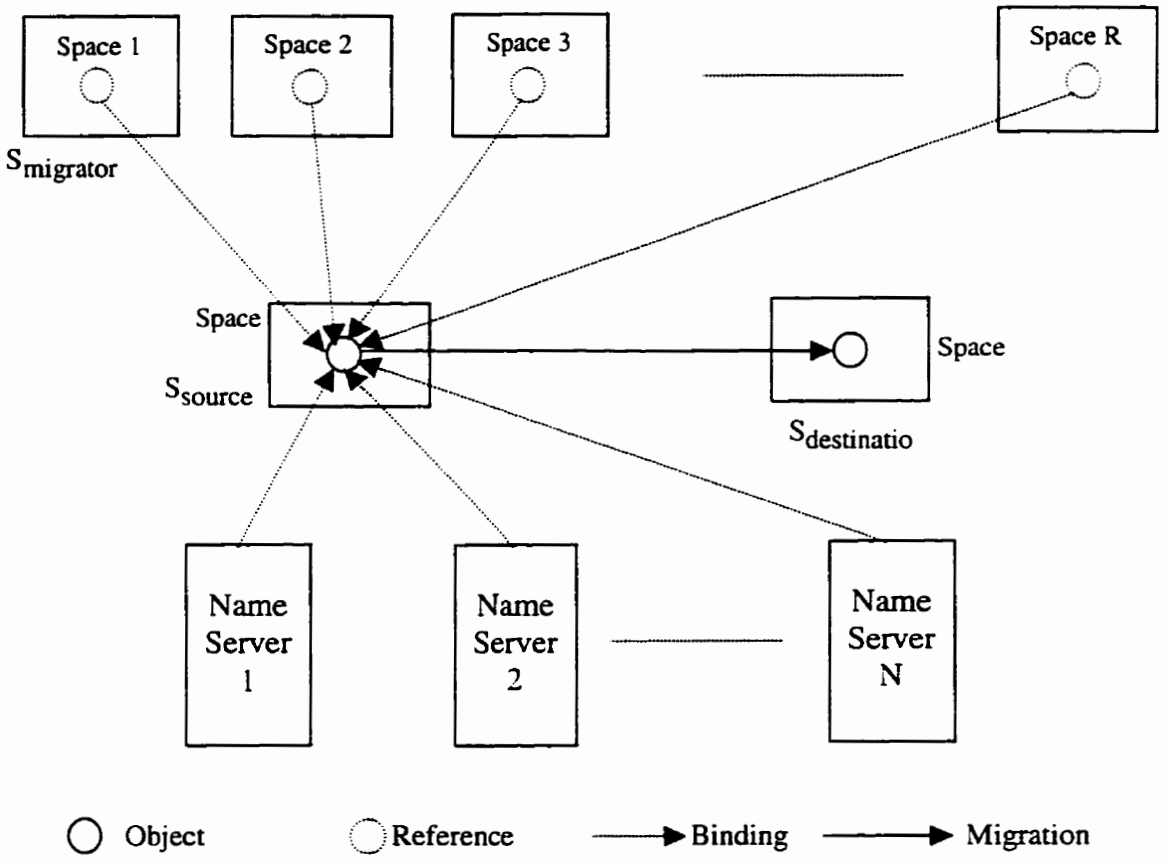


Figure 5: Object Migration Tracking

We can compare the cost for each of these solutions both at migration and invocation time by considering the major messages and assuming a fault-free environment. For simplicity, we have made the following assumptions in Figure 5:

- The system has S spaces.
- An object O, resides at one of the spaces, S_{source}
- R spaces contain a reference to O
- N spaces contain name-servers which refer to O
- Reference holder on space $S_{migrator}$ sends a message to S_{source} instructing O to migrate to $S_{destination}$

- Assumptions R and N are disjoint sets
- $S_{migrator}$ is not equal to S_{source} which is not equal to $S_{destination}$

These messaging costs are summarized in Table 1.

- Forward referencing requires no additional messages at migration time but for the first invocation by each stale reference, an extra message is required to follow the indirection to the new destination. From table 1 we could see the forwarding and updating schemes are least costly, but the updating method is not as efficient as the forwarding method because it will send messages to all references even if some of them may not invoke the migrated object. So forward referencing is used widely in distributed object system.

	Migration Costs	Invocation Costs
Forwarding Scheme	**0**	**R - 1**
Searching Scheme	**0**	**(S - 2) * (R - 1)**
Updating Scheme	**R - 1**	**0**
Registering Scheme	**N**	**(R - 1) * 3**

Table 1: Summary of Message Cost in Distributed Object System

- The search solution also requires no additional messages at migration time, but for the first invocation by each reference from 1 to (R-1), it has to send messages to all other (S-2) spaces except for itself and S_{source} to query the new object location. This method is the most expensive under all circumstances. But in some cases when the system loses its information on the object's current location, a search must be used to find the migrated object.
- In the updating scheme, in order to let the holder of every reference be informed of the new location, additional (R-1) messages are required at migration time,

but no additional messages are required when invocation occur. It has the same cost as the forwarding solution, but it is less efficient than the forwarding reference because of the reasons mentioned above.

- For the registering service solution, N messages are required at migration time to inform the name servers. When the reference invokes the missing object, the fault will occur and it will query the nearest name server to discover the new location, followed by the actual access of the object.

In mobile agent systems, agents can decide to move by themselves, they don't need $S_{migrator}$ to instruct them, so the Table 1 doesn't apply to mobile agents. We recalculate the message cost in Table 2.

	Migration Costs	Invocation Costs
Forwarding Scheme	**0**	**R**
Searching Scheme	**0**	**(S - 2) * R**
Updating Scheme	**R**	**0**
Registering Scheme	**N**	**R * 3**

Table 2: Summary of Message Cost in Mobile agent Systems

From Table 2 we get similar results: the Forwarding solution is the first choice for mobile agents. Most mobile agent systems use this scheme to track mobile agents. However, in mobile agent systems, agents move frequently and as such the forwarding chain becomes too long and is costly to maintain. Many systems [Caughey 95], [Ingham 96] and [ObjectSpace 1997] use short-cut methods to reduce the chain. That is, every possible point along the chain could be "cut" and any unnecessary forward references will be garbage collected. For example in Figure 6, if agent2 accesses agent3, it will invoke

Forward1 and Forward2. Because Forward2 is unnecessary, after invocation it will be cut and the situation will become that shown in Figure 7. This method will reduce the length of the forward chain.

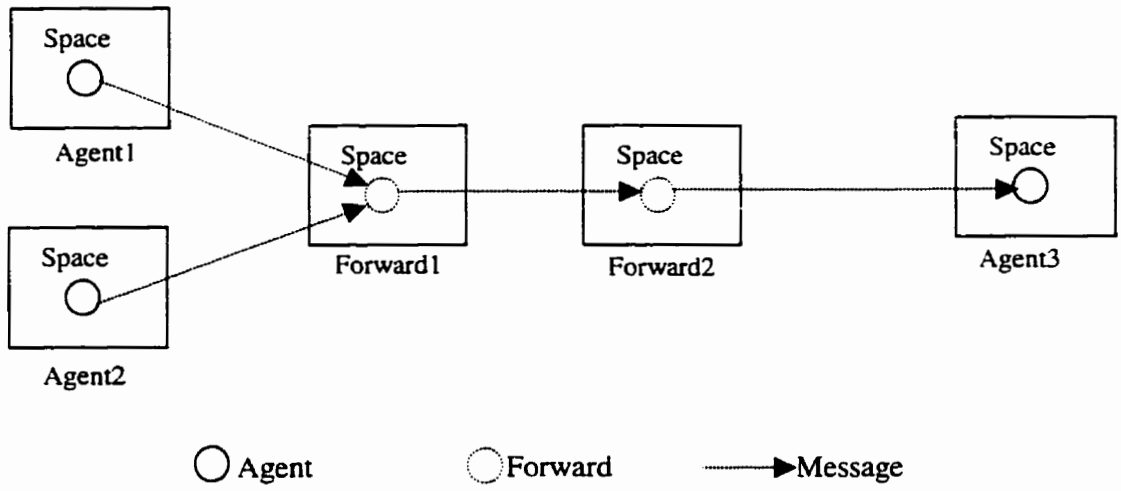


Figure 6 Reference Chain Before short-cut

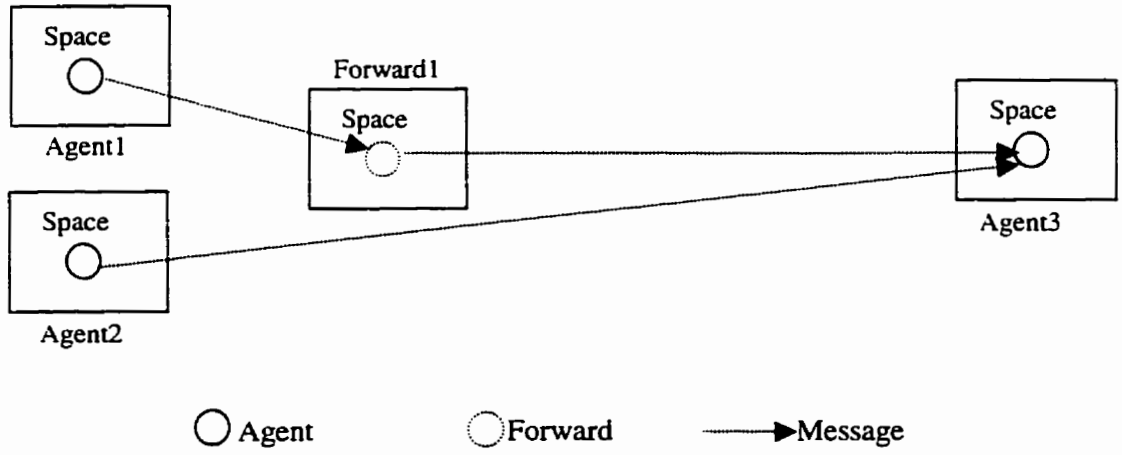


Figure 7 Reference Chain After short-cut

When in an unreliable network, it may be too risky to rely on forwarding. It will aggravate the problem by introducing additional points of failure along the chain of reference to the object. We could use a broadcast protocol [Jul 88] whenever the previous

step has failed to find the agent. The searching space sends a broadcast message to all spaces, only a space that has the specified agent responds to the broadcast. If the searching space receives no response within a time limit, it sends a second broadcast requesting a positive or negative reply from all other spaces. All spaces not responding within a short time are sent a reliable, point-to-point message with the location request. If every space responds negatively, we conclude that the agent is missing.

2.5 Communications

Messages between agents can be exchanged in many ways.

- Synchronous: an object that has invoked a method on another object blocks until it gets the response.
- Asynchronous: the client that has invoked the operation will be interrupted when the response is available. So the client will not be blocked for the response.
- Semisynchronous: the client polls for the response periodically while continuing its normal operation.

Supporting point to point communication is not enough in mobile agent systems. For example, if a group of agents cooperatively perform a user-defined task or if one group member wants to meet another member of this group at a particular place for the purpose of cooperation. So mobile agent systems should also support following types of communications:

- User/Agent communication

Communication with the user is the easiest type of agent communication to support. A user could monitor agents and open a channel to communicate with it and also the agent could talk to the user if permission is granted.

- Agent/Service communication

This style of communication is typically of the client/server form. Agents request services for certain information, and results are reported by responses. For simplicity, many systems use RPC-like communication mechanism.

- Agent/Agent communication

This type of communication differs significantly from the previous one. The role of the communication partners are peer-to-peer rather than client/server. Each mobile agent has its own agenda and hence initiates and controls its interactions according to its needs and goals. The communication patterns that may occur in this type of interaction might not be limited to request/response only. The required degree of flexibility is provided by a message passing scheme.

- Anonymous agent group communication

In the previous types of communication, the communication partners know each other and the sender of a message or RPC is able to identify the recipients. However, there are situations where a sender does not know the identities of the agents that are interested in the sent message. Assume, for example, a given task is performed by a group of agents,

each agent taking over a subtask. In order to perform their subtasks, agents themselves may dynamically create subgroups of agents. In other words, the member set of the agent group responsible for performing the original task is highly dynamic. Of course, the same holds for each of the subgroups involved in this task. Now assume that some agent wants to terminate the entire group or subgroup. In general, the agent that has to send out the terminate request does not know the individual members of the group to be terminated. Therefore, communication has to be anonymous, i.e., the sender does not identify the recipients. This type of communication is supported by group communication protocols [Birman 94]. Another approach ([Baumann 1997] [ObjectSpace 1997]) is the event model in which senders send out event messages anonymously, and receivers explicitly register for those events they are interested in. The event model is particularly well-suited for distributed communication since it abstracts from the receiver's identity. As a consequence, it enables the specification of complex interactions without the need to know the communication partners in advance. With regard to agent systems, the event model simplifies application as well as system level communication. On the application level, events are employed as a general communication means. On the system level, events can be used to design and implement protocols that encompass agent synchronization, termination, and orphan detection.

Mobility of agents increase the complexity of the communications. In some systems [Baumann 1997], while an agent is involved in a session, it is not possible to move to another space. However, if it decides to move anyway, the session is terminated implicitly. The main reason for this property is to simplify the underlying communication

mechanism, i.e. to avoid the need for message forwarding. If one of two sides is static, keeping the connection channel is not difficult. When the agent migrates, the channel is closed prior to migration and then reopened at the new location. Depending on the communication pattern, either eager or lazy opening of the channel is possible. Eager opening of the channel introduces higher startup costs, while lazy opening introduces communication delay for the first communication attempt. But communication between agents is more difficult to achieve due to arbitrary movement of both sides of a connection. Large evaluation appears to be a well suited mechanism. Otherwise, the system could spend most of the time updating communication links instead of doing useful computation. There are different ways [Milojicic 96] of maintaining communication channels between mobile agents. One can update the agent's current location at the original site, or maintain a chain of forwarding addresses which can eventually be collapsed. Similarly, an agent can register its name at predefined name servers, allowing the re-opening of communication channels to rely on the naming mechanism to resolve the agent's new location. These approaches trade-off performance and complexity with the amount of state information that needs to be maintained.

2.6 Distributed Persistence

An object is said to be persistent if it can live outside the process that has created it. Object persistence could be achieved by storing the state of the object on a persistent storage device. In many distributed applications there is a need for persistent objects. For example, when the lifetime of an object extends the lifetime of a single application, persistence provides the appropriate solution and persistence can also be applied to

achieve fault-tolerance, thus ensuring that an object can survive node crashes. Because a persistent object maintains a copy of its state on stable storage, node crashes are never fatal. The object can simply be reinstantiated from its stable copy.

2.6.1 Persistence with different sites

Most mobile agent systems use the local storage to save the agents. We could use similar concepts and methods [Moons 92] in distributed object management to extend persistent functions. In mobile agent systems, we could set up one or several Persistent Storage Managers (PSM) which back up the agents on a stable storage medium. The PSM could reside in one agent space and every agent has to be associated with a PSM. There is no need for an agent and its PSM to be situated on the same physical node. When an agent state changes, it doesn't need to send its changes to the PSM immediately. Because agents are autonomous, they will call PSM when they have modified state and information to save. Because the copy of an agent is saved in a different site, when the node that hosts a persistent agent crashes, the agent can still be reinstantiated on another node.

Now we introduce the persistent working principle for an agent and its copy on a different site. A persistent agent can disappear from memory because of a space or node crash. In this case, the agent now solely exists on stable storage. When an agent in the space accesses it, the action will fail. So the agent invoking the action will query the PSM to request its service. Agent reinstantiation is triggered by this request as in Figure 8. When invoking an operation on a persistent agent, the invocation request is delivered to

the node where the agent is believed to reside (step 1). Due to a space or node crash, the persistent agent may no longer be available. In this case, the agent1 looks up the PSM it knows (step 2). The PSM then takes it from stable storage (step 3 and step 4), reinstantiates this agent to a space (step 5) and informs agent1 of the location of the reinstantiated agent (step 6). Now agent1 can proceed with its interaction with the persistent agent (step 7). Also the persistent agent could be reinstantiated at the same place as with the PSM then migrate to the space where agent1 is hosted.

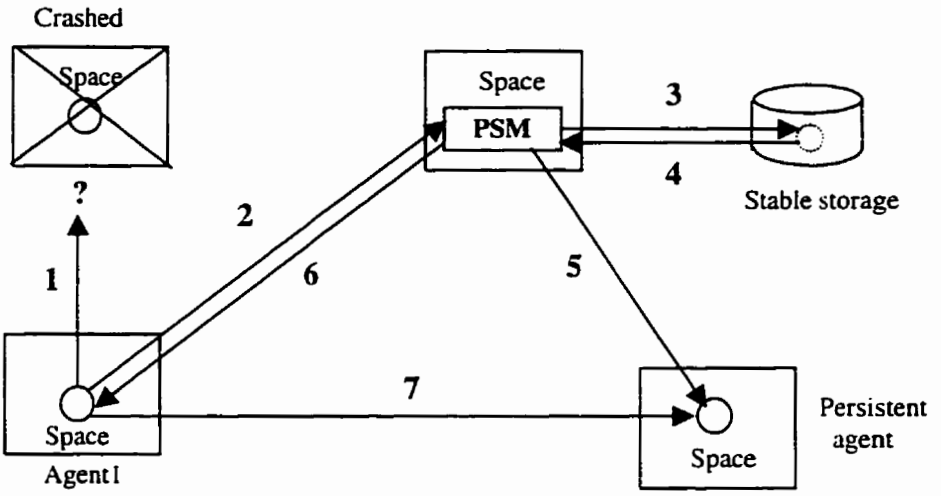


Figure 8: Distributed Persistence

2.6.2 Consistency versus Availability

Network failure will make it difficult for the PSM to decide whether a persistent agent has crashed. When the agent involved is situated in a separate network partition, the PSM has no way to differentiate between a crashed and an unreachable agent. Reinstantiating the agent might thus result in multiple instances, which is clearly intolerable. We could

use two solutions introduced in [Moons 92]. They are the strict-consistent and relaxed-consistent persistent agent.

- With strict consistency we guarantee that a persistent agent removes its memory instance within a time period T, as soon as its PSM becomes unreachable. The time period T is called the die-out interval. Correspondingly, whenever the PSM is asked to reinstantiate an agent, it delays this request for a period $T+\epsilon$, with ϵ representing the network diameter in time units. When the period $T+\epsilon$ has passed, the PSM can be assured that there are no longer live instances of the agent in memory, so reinstantiation will not introduce duplicates.

- For many applications it is however better to have access to slightly out-of-date information, than to have no information at all. Relaxed consistency offers a solution in these circumstances. A relaxed-consistent persistent agent does not crash when its PSM becomes unreachable. Instead, it is converted into a zombie agent. A zombie continues to serve requests, as long as they don't modify the persistent agent's state. Attempts to obtain write access to persistent memory will cause the corresponding action to abort. Zombie agents remain available until the PSM comes back on line. At that moment, they are either revived or destroyed.

2.7 Distributed Garbage Collection

Distributed garbage collection is harder than local garbage collection because the local collectors must be coordinated to consistently keep track of changing references between

address spaces. This consistency problem is further complicated by the common failures of distributed systems such as lost, duplicated or later messages and crash of individual spaces. Distributed garbage collection poses a challenging problem: reclaiming all kinds of data structures while achieving efficiency scalability and fault-tolerance. A number of proposals have attempted to design a distributed GC (Garbage Collection) that fulfills all these requirements. The great number of incomplete proposals reflects how difficult the challenge is.

2.7.1 Distributed Reference Counting

In Reference Counting a counter is associated to each object denoting the number of references to it. Upon object allocation, the associated counter is initialized to one. Each time a reference to this object is duplicated, the object's counter is increased by one. Conversely, each time a reference to an object is discarded, its counter is decreased by one. Therefore, reference counting preserves for each object the invariant that the value of the counter is always equal to the number of references to the object. Reference counting detects garbage objects immediately after the last pointer to an object has been deleted. When an object's counter drops to zero, it implies that the object is no longer reachable and can be safely reclaimed. The native extension of reference counting to distributed systems keeps a reference count with each public object. The reference count is updated on each duplication or deletion of a reference. The count number is updated by sending a control message. In a distributed system, those control messages must be delivered reliably without loss or duplication to preserve the reference counting invariant. Also control messages should be delivered to their destinations in causal order to prevent

race conditions. For example, the decrement/increment race can occur when a sender space duplicates a reference and deletes it immediately after as shown in Figure 9.

Space1 holds a reference to object C located in space 3. Object A sends that reference to B and immediately deletes its own pointer to C. Upon receiving the message containing the reference C, Space2 installs it and consequently sends an increment message to Space3. Upon deleting its own reference to C, Space1 sends a decrement message to Space3. If the decrement message from Space1 arrives to Space3 before the increment

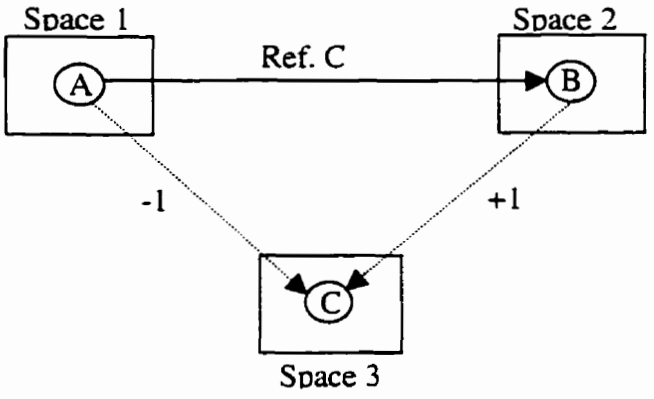


Figure 9: Race Conditions between Decrement and Increment Messages

message from Space2, then the corresponding counter of that object will drop to zero. Since the object C is not reachable from any object, the object C will be reclaimed prematurely. One way to overcome these potential race conditions is to use acknowledge messages mechanism. For example, the decrement message from Space1 could not be sent unless it receives the increment acknowledge message from Space2. But the native extension of reference counting is still not resilient to message failures because increment and decrement messages are not reliable. A number of adaptive methods [Plainfosse 95]

have been suggested to improve resilience to either message failures or race conditions. For example, Weighted Reference Counting (WRC), indirection entry items method, and optimized Weighted References Counting (OWRC).

2.7.2 Reference Listing

For expressing this method clearly, the concepts of stub and scion are used as in [Plainfosse 95]. Remote reference is composed of a local pointer, a stub and scion. The local pointer points to a stub which in turn remotely refers to a scion. The scion holds a local pointer to the public object. Such public objects can be remotely invoked through the remote reference, as opposed to local objects which are only pointed locally. Reference listing differs from reference counting in the way scions are managed. Instead of keeping a counter, a space allocates a list of separate scions, one for each client space that owns a reference to the same object. Each scion contains the identity of its predecessor space. The invariant is that each stub referring to an object has a corresponding successor scion for that particular object. Increment and decrement messages are replaced, respectively, by insert and delete messages (collectively called control messages). A delete message informs a scion that it is no longer referenced.

Reference listing improves resilience to message and space failures over reference counting techniques at the expense of a some memory overhead. The major advantage of reference listing over reference counting is that messages are idempotent hence resilient to message failures (duplication and loss). For instance, the same delete message may be sent several times without consequences on the invariants preserved by the particular

technique. If a previous delete message has already been received the following one is simply ignored; if the previous delete message has been lost then the following one is processed. Resilience to space failures relies on the ability for each owner space to compute the set of its clients by looking through the scion lists so it can prompt one of these to send a live (or delete) message. Additionally the owner may explicitly query a particular reference that is suspected to belong to a garbage distribution cycle. Furthermore, if one of these clients is down then the owner space can take the proper decision between two alternatives: (1) keep the objects referred to by the crashed space until it recovers, or (2) reclaim at once the objects that the crashed space refers to. The former policy assumes that scions and stubs lists will be recovered. i.e., because they are backed up on stable storage. The latter policy assumes that a crashed space will not recover.

2.7.3 Tracing-based Distributed Garbage Collectors

Reference counting and reference listing collectors can not reclaim garbage cycles spanning spaces. Therefore, such acyclic techniques only work if those cycles are rare enough to be neglected. A standard approach to distributed tracing is to combine independent local, per-space collectors, with a global inter-space collector. The two types of collector interfaces are through stubs and scions. We first introduce two concepts: mark & sweep. They perform garbage collection in two temporally distinct phases: the mark phase is first responsible for garbage detection whereas the sweep phase performs garbage reclamation. Each object carries a special color bit set to color white or to color black. By convention, black objects are live whereas white ones are garbage. At the

beginning of mark phase, all objects in the heap are marked white. The mark phase traverses the graph of reachable objects from the root, coloring each object encountered black. Subsequently, the sweep phase traverse the heap linearly and reclaims all white objects. The sweep phase reclaims garbage objects in place by chaining them together. The memory associated with the list of garbage objects is reused upon allocation.

In this method, two problems are required to be solved. The first is to synchronize the distributed mark phase with independent sweep phase. During the mark phase local collectors receive and send marking messages exchanged between clients and the owner space. A local GC can be resumed if it receives a marking message for an object it owns. Therefore, spaces are alternatively cooperating to the global marking and alternatively waiting for a marking message. The mark phase is complete when all reachable public objects have been marked and there is neither marking or acknowledgment message in transit. Afterwards, each space independently triggers a sweep phase in order to reclaim public and local garbage objects. The second problem is to maintain the consistency of scions with stubs, in the face of message and space failures, and of race conditions. In fact, if local GCs, mutators, and the inter-collector all operate in parallel with each other and messages are not instantaneous, then strict consistency is not achievable.

There are three main distributed GC techniques. One is the tracing with the timestamps. It is based on distributed mark-and-sweep. When mark bits are replaced by timestamps, a global GC propagates a stub's timestamp to its successor scion. The key idea of the algorithm is that a garbage object's timestamp remains constant whereas non-garbage

object's timestamps increases monotonically. A timestamp threshold is computed in order to provide the barrier synchronization between the mark and the sweep. Scions and stubs both contain a timestamp initialized with a global clock. Each local GC repeatedly traces objects from the local root and from the scions. A stub reachable from the root is marked with the time at which marking started; one reachable from a scion receives that scion's timestamp. Scions with a timestamp less than the global threshold are collected. Scions that carry timestamps less than the threshold can be safely reclaimed. Hughes' algorithm collects both cyclic and acyclic distributed garbage since it is based on tracing.

The second method is Centralized Reference Service. It computes global accessibility of objects on a highly available centralized service. This service is logically centralized but physically replicated, hence achieving high availability and fault-tolerance. All objects and tables are backed up in stable storage. Clocks are synchronized and message delivery delay is bounded. These assumptions enable the centralized service to build a consistent view of the distributed system.

The third method is Tracing Within Groups. It combines reference counting and mark-and-sweep in order to perform garbage collection within groups. A group is a dynamic collection of spaces that may overlap or include other groups. The dynamic property of groups enable them to remove failed spaces in order to not block garbage collection. Group nesting allows one to build a hierarchy of groups in order to support cyclic garbage collection within networks as large as the world. A space belonging to a group is a member of that group. Conversely, external spaces of a group do not belong to that group. The algorithm proceeds in several steps. The first step is a group negotiation.

During this step, spaces exchange messages to build up a group. The next step, initial marking, distinguishes inter-group from intra-group references. For this purpose, each space sends a decrement message for each stub it holds within the group. At the end of the initial marking, scions with a counter equal to zero are internal to the group and colored white; the others are referred from at least one external space, and will be colored black. The following step, global marking, performs a global mark-and-sweep within the group. This step relies on local tracing garbage collections to propagate the black color from scions down to stubs. The marking phase first traverses the local root and then the list of list scions, first the black ones, then the ones white. This order of traversing scions prevents whitening of black objects. At the end of tracing, all blackened stubs are reachable either from a root within the group, or from some external space. Conversely, white stubs are garbage. Each space sends a color message containing a list of black stubs it holds to each corresponding space within the group. The marking step completes when all spaces have sent a color message to each peer and when there is no more color message in transit. Note that color message may lead to a blackening of white stubs, leading to the sending of additional color messages. At the end of the marking step, white scions can be freely reclaimed. Each space runs a sweep step to reclaim unreachable global objects.

From the analysis above, we found that mobile agent systems have much in common with the distributed object systems, especially in object migration, process migration and operating system support. Some concepts and approaches of mobile agents come from the domain of distributed object system. But the mobility and autonomous nature of

mobile agents add complexity in the implementation of mobile agent system. Some problems need to be considered in detail. For example, how to decide the optimized portion of object graph in a hybrid migration environment? How to find the best way to keep the open channel when the agent moves? How to increase the fault-tolerance in tracking mobile agents? Also some work in mobile agent management exists that this section has not covered, for example, naming facilities and security problems. They should be investigated in the future.

CHAPTER 3 LITERATURE SURVEY

As mentioned above, mobile agents have been used in many fields because of their advantages over client/server methods. As such, I am only going to present mobile agent applications in computer networking. Today's telecommunication service and management architectures are insufficient for the highly dynamic computer networking of the future. Despite their "Open" philosophy, current telecommunication standards such as IN and TMN are too unflexible to provide a universal architectural basis for future telecommunication applications. Because of the underlying unflexible client-server concepts such systems are not safe from failures and bottlenecks, and are relatively hard to configure and to extend. They are therefore not very well suited for the dynamic characteristics of a modern network. Currently the most promising candidate for the future telecommunication system architecture is the "Telecommunications Information Networking Architecture" (TINA) developed by the international TINA consortium (TINA-C). Based on object-oriented concepts and standards (e.g. ODP), TINA has been proposed as an object-based system architecture which allows the flexible distribution of individual components communicating transparently through a Distributed Processing Environment (DPE). Although there are possibilities for dynamic configuration and object migration, TINA is at its core still based on the traditional client-server concept. Mobile, intelligent agents are not yet part of TINA. Due to their autonomy, mobility, persistence, ability to learn and their communication and cooperation skills, agents appear to be well suited for dynamic telecommunication environments. They lend themselves easily to the task of representing service providers and users in electronic communications, and are therefore almost predestined for this particular application field.

Although there are a lot of papers discussing general ideas about the mobile agent advantages and their applications, few research papers focus on concrete applications especially in computer networks or telecommunications networks aspects. The following is a list of the project instances surveying mobile agent applications in the computer networking field:

3.1 Perpetuum Mobile Procura Agent Project

The Perpetuum research project [Perpetuum Project] has the following aspects:

1) Fault management (Diagnosis/Recovery)

Concentration is on preventing or detecting and fixing network problems by distributed artificial intelligence.

2) Swarm Intelligence for Network Management

This research involves studies of the behavior of societies of simple ant-like agents that achieve tasks by very high degree of cooperation. Insight into the evolutionary aspects of the behavior of agent communities generating some ideas on the possible uses of such societies for network management purposes.

3) Plug and Play Networks (Auto-provisioning)

This goal of this aspect of the research is to design and implement a scheme for automatic provisioning of components and services. Each component provided by vendor contains a provisioning agent that is injected into a network after plugging in the component. The

agent is provided by the vendor. Its role is two-fold: a) Communicate to any concerned parties that a new component has been plugged in (including provisions for the delivery of any required data), and b) Obtain any data that is needed on the component/service side to function properly.

4) Self-repairing Networks

The idea is to inject all sorts of agents into the network that would be intelligent enough to take care of most of the problems by activating recovery routines or by planning other required activities. The intelligence can be build into the agents and/or come from components provided by vendors.

5) Network Security

The most common concern about mobile agents cruising telecommunication networks is network security. This is also the obstacle to the popularity of mobile agents in general.

6) Automatic Setup of ATM PVC's

Setting up a PVC in an ATM network involves several parties: the endpoint switches (the users of the PVC) and the provider of the connecting route (usually, an operating company). Very often, the manager has to deal with heterogeneous equipment and multiple APIs. A specialized mobile agent can perform the same task automatically. The agent visits the involved parties and communicates with them.

3.2 FOKUS Project

This project [FOKUS Project] is divided into several sub-project related to mobile agents as follows:

1) BCNM

The goal of this project is to develop Java-based, dynamically extensible solutions for Customer Network Management in a Global Broadband Telecommunications environment. There is no detailed information released.

2) MAGNA - Mobile AGeNt Architecture

The goal of the MAGNA project is to enhance the concepts of TINA and similar architectures (e.g. OMG's OMA architecture) towards a generic architecture for future telecommunication applications, in which the traditional client-server concept is harmoniously complemented by agent concepts. This architecture is the foundation for the longer term goal of developing a comprehensive framework (MAGNA agent framework) for the developement of agent-based telecommunication applications. Additional framework constituents which will be realized in the course of the MAGNA project, are a CORBA and Java based reference platform for agent systems (MAGNA Agent Platform), a design and development methodology, corresponding tools and generic application components which can be used as building blocks for new, agent-based telecommunication applications. Analysis and design methods, tools and generic components will form the basis for the long-term development of an integrated agent development environment (MAGNA Agent Development Environment). The theoretical

and practical results of the project will be used for the realization of a number of agent-based telecommunication services which exhibit the aspect of rapid, decentralized provisioning of intelligent, partially multimedia-enabled services ("Intelligent services on demand") in a suitable way. Among the services which will be realized will be an agent-based "1-800" service ("free phone service"), as well as agent-based abbreviated dialing and incoming/outgoing call screening functions for telephone connections. Additional plans call for agent-based management applications, e.g. for the distributed configuration and management of Virtual Private Networks (VPNs).

3) MAMS

A co-operation project with NEC (Japan) for developing an Intelligent Mobile Agent platform and network management solutions for broadband ATM environments. No detailed information has been released.

4) MIAMI

MIAMI is the short name for Mobile Intelligent Agents for Managing the Information Infrastructure. The emerging Open European/Global Information Infrastructure (EII or GII) is characterised by increased distribution, its dynamic nature, and the complexity of its resources. To manage such an environment, increased intelligence in management solutions and the mobility of such solutions are becoming major requirements. The MIAMI project, within this context, is going to focus on the following key objectives: a) create a unified mobile intelligent agent (MIA) framework by validating, refining and enhancing the OMG MASIF standards according to the requirements for an Open

European Information Infrastructure. b) develop mobile intelligent agent (MIA) based solutions for the management of the Open EII and for the provision of advanced communication and information services, and c) create a reference implementation of the Unified MIA Framework and solutions in order to evaluate the service solutions in a Pan-European business environment.

5) NIMA

A co-operation project with Hitachi (Japan) for developing Intelligent Mobile Agent-based network management applications for SDH/SONET telecommunications transmission networks. The major goal of this project is to design, implement and validate/evaluate MIA-based solutions and functions for telecommunications network management. This goal can be further refined into the following sub-objectives: a) adaptation of the existing mobile agent platform of FOKUS to support MIA-based (Mobile Intelligent Agents) network management solutions and functions. b) development of a MIA framework for telecommunications network management development of MIA-based management applications for HITACHI (SDH-based) network scenarios, and c) Validation of the MIA-based network management paradigm on simulated network environments. This project will focus on MIA-based fault management functionality and will address those performance/configuration functions that are required in the fault management scenarios.

3.3 Distributed Artificial Intelligence (DAI) Research Unit

This project [DAI Project] has developed and applied agent and multi-agent techniques to real world problems in a wide range of commercial and industrial domains. Applications which have been addressed include: telecommunications network management, business process management, electricity management, patient care, concurrent engineering, 3-D scientific data interpretation, digital libraries, and process control. The DAI project has also worked on formalizing a number of key types of behavior which can be observed in multi-agent systems. This work has used a variety of formal techniques to investigate social rationality, cooperative problem solving, negotiation, argumentation, and coordination in multi-agent systems. This project has two sub-groups related to mobile agents, neither of which provide a detailed description of how to realize their functions:

Agent CAC - Market Based Control for the Management of Telecommunications Networks:

This EPSRC funded project investigates a novel approach for managing Connection Admission Control (CAC) in Asynchronous Transfer Mode (ATM) telecommunication networks. The project [Gibney 97] addresses the application of an intelligent multi-agent system, in which agents are modeled as self interested decision makers in a computational market, to network level resource allocation. The objective of the project is to implement demonstration and simulation software to evaluate the resource allocation performance of an intelligent multi-agent system compared to conventional techniques. They contend that, combining agent technology with fuzzy techniques to learn the traffic behavior in a link and use this as a basis for controlling connection access would add a

new dimension to management and traffic control in ATM networks resulting in ATM nodes that both learn (adapt) and co-operate.

3.4 Routing with Swarm Intelligence

Some work has been done on routing with swarm intelligence ([White 1997], [White 1998]). It describes how biologically inspired agents can be used to solve control and management problems in telecommunications. These agents, inspired by the foraging behavior of ants, exhibit the desirable characteristics of simplicity of action and interaction. The collection of agents, or swarm system, deals only with local knowledge and exhibits a form of distributed control with agent communication effected through the environment. They explored the application of ant-like agents to the problem of routing in circuit switched telecommunication networks.

The purpose of the study was to assess the feasibility of applying swarm intelligence to telecommunication network dynamic routing. They expected to develop an algorithm in the context of ATM networks and a software demonstrator in order to show the importance of pursuing research in this direction.

Swarm intelligence research originates from the work on the emergence of collective behaviors of real ants. Ants have a small amount of cognitive capability, limited individual capabilities, and react instinctively in a probabilistic way to their perception of their immediate environment. They can, for example, find a path between the nest and a food source by laying down on their way back from the food source a trail of an

attracting substance, called pheromone. Ants wandering randomly around the nest can then be attracted by this trail and this will rapidly lead them to the food source. By laying down a pheromone trail of different density depending on the quality of the food source the have found, the colony becomes able to discriminate between food sources of different kinds and qualities.

The advantages of swarm intelligence are twofold. Firstly, it offers intrinsically distributed algorithms that can use parallel computation quite easily. Secondly, these algorithms show a high level of robustness to change by allowing the solution to dynamically adapt itself to global changes by letting the agents (the ants) self-adapt to the associated local changes.

3.4.1 The Ant System

The Ant System is a general-purpose heuristic algorithm, which can be used to solve diverse combinatorial optimization problems. The Ant System (AS) has the following desirable characteristics:

1. It is versatile, in that it can be applied to similar versions of the same problem. For example, there is a straightforward extension from the travelling salesman problem (TSP) to the asymmetric travelling salesman problem (ATSP).
2. It is robust and general purpose. With minimal modifications, it can be applied to other combinatorial optimization problems such as the job shop scheduling problem (JSP) and the quadratic assignment problem (QAP).

3. It is a population-based heuristic. As such, it allows the exploitation of positive feedback as a search mechanism, as described in a later section. Consequently, it makes the system amenable to parallel implementations.

The AS is an example of a distributed search technique. Search activities are distributed over ant-like agents, i.e. entities with very simple basic capabilities. These agents, in a metaphorical and highly stylized way, mimic the behavior of real ants. In fact, research on the behavior of real ants has greatly inspired the AS. The research inspiration for the AS arises from the work of ethologists, who attempted to understand how almost blind animals like ants could manage to establish shortest route paths from their colony to feeding sources and back.

Research found that the mechanism used for the communication of information among individuals regarding paths, and used to make routing decisions, consists of the sensing of pheromone trails. A moving ant lays some varying quantity pheromone on the ground, thus marking the path by a continuous trail of the substance. While an isolated ant apparently moves at random, an ant encountering a trail laid by an ant of its own species can sense it and decide to follow it with high probability. In this way, the trail is further reinforced with more pheromone. The collective behavior of many ants attempting to find a path that emerges is in the form of an autocatalytic process where, the more the ants follow a trail, the more attractive that trail becomes for being followed. The process forms a positive feedback loop, where the probability with which an ant chooses a path will increase with the number of ants that previously chose the same path.

The Ant System is a new search methodology based on a distributed autocatalytic process and its application to the solution of a classical optimization problem. The general idea underlying the Ant System search paradigm is that of a population of agents each guided by an autocatalytic process directed by a greedy force. If an agent were to search alone, the autocatalytic process and the greedy force would tend to make the agent converge to a sub-optimal solution. When agents interact it appears that the greedy force can give the right suggestions to the autocatalytic process and facilitate rapid convergence to very good, often optimal, solutions without getting stuck in local optima. We speculate that this behavior arises because information gained by agents during the search process is used to modify the problem representation. In some sense, the region of the space considered by the search process is reduced. Even if no tour is completely excluded, bad tours become highly improbable, and the agents search only in the neighborhood of good solutions.

The main contributions of the Ant System are:

1. Positive feedback is employed as a search and optimization tool. The idea is that, if at a given point an agent (ant) has to choose between different options, and the one chosen has good results, then in the future that choice will appear more desirable than it was before.
2. Synergy can arise and be useful in distributed systems. In AS, the effectiveness of the search carried out by a given number of cooperative ants is greater than that of the

search carried out by the same number of ants, each one acting independently from the others.

3. The Ant System can be applied to different combinatorial optimization problems. Namely, AS provides a general search heuristic that can be applied to problems that can be represented as a search on a graph.

3.4.2 Swarm Intelligence Algorithm

The swarm algorithm solution to the routing problem relies on the movements of artificial ants on the associated graph designed to make the global shortest path emerge. When a connection request is made, a new colony of ants is created and associated nests are positioned on the source and destination nodes. These artificial ants correspond to a special class of automata called reactive agents that react to their local perception of the environment by stochastically adopting predefined behaviors. The shortest path then emerges from the movement of all these ants.

There are three classes of ant. These are:

1. Explorer ants: these ants search for a path from a source to a destination.
2. Allocator ants: these ants allocate resources on the links used in a path from a source to a destination.
3. Deallocator ants: these ants deallocate resources on the links used in a path from a source to a destination.

The algorithm is quite straightforward. Both source and destination nodes of the connection are considered as nests. The explorer ants leave the nest and explore the network following their local rules:

1. On each node, they choose a path with a probability proportional to the heuristic value (function of the cost and the pheromone level) associated with the link.
2. The ants can not visit a node twice (they keep a tabu list of their visited nodes) and cannot use a link if there is insufficient bandwidth available.
3. Once the destination is reached, the ants return from whence they came by popping their tabu list. On their way back, they lay down a pheromone trail.

When the source node judges that a unique path has emerged, it sends a special kind of ant, the allocator, in order to allocate the bandwidth on all links used between the source and the destination. Pheromone laid on a link will evaporate over time. This is controlled by a constant evaporation rate.

They tested this algorithm on a network. The algorithm appears to work best on networks where connections are relatively long lived - in the terminology of ATM, permanent virtual circuits rather than switched virtual circuits. All of the above points can be addressed with the application of Java, a mobile agent framework. A further benefit of having agents roam the network is that they can gather transmission delay and cell loss information. This can be fed back to the management system, as statistics about how well the algorithm and network are operating.

3.4.3 Summary

Investigations and experiments have shown that this multi-agent search technique, called Swarm Intelligence, can solve routing problems on networks. The main strength of the algorithm is in the way it can adapt to new situations. Real ants are able to quickly find a new path when their environment changes, and the swarm algorithm also exhibits this behavior. The algorithm also appears to perform well in problems associated with a large graph.

Adaptive behavior could be used in managing incidents, or network failures. The fact that it can adapt to totally new situations may be very useful. This way, during the failure of some of its physical elements, a network would be able to manage routing. It is interesting too, that the agents can behave and act without human intervention. During network failures, this algorithm could behave as the "immune system" of the network and insure that during an incident the routing of data would still be done. Also the potential benefits of this kind of approach are great: no overall control, simple agents, robustness and adaptation. This work is quite compelling.

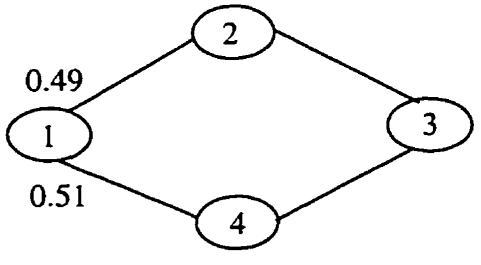
3.5 Ant-based load balancing in telecommunications Networks

The work ([Schoonderwoerd 96], [Schoonderwoerd 97]) used the same principle as in the swarm intelligent routing in a research project at Hewlett-Packard Laboratories Bristol. It presents a novel method of achieving load balancing in telecommunications networks. In telecommunications networks, nodes carrying an excess of traffic can become congested, causing calls to be lost. The approach presented here is to solve these

congestion problems, which is modeled as the trail laying abilities of ants. Artificial ants move across the network between randomly chosen pairs of nodes; as they move they deposit simulated pheromones as a function of their distance from their source node, and the congestion encountered on their journey. They select their path at each intermediate node according to the distribution of simulated pheromones at each node. Calls between nodes are routed as a function of the pheromone distributions at each intermediate node.

3.5.1 Algorithm Description

In this method, routing tables in the network nodes were replaced by tables of probabilities, which they call 'pheromone tables', as the pheromone strengths are represented by these probabilities. Every node has a pheromone table for every possible destination in the network, and each table has an entry for every neighbor. The entries in the tables are the probabilities which influence the ants' selection of the next node on the way to their destination node. Figure 10 shows a possible network



Destination Node	Next Node 2	Next Node 4
2	0.95	0.05
3	0.49	0.51
4	0.05	0.95

Figure 10: The pheromone tables for node 1

configuration and a pheromone table. For example, ants travelling from node 1 to node 3 have a 0.49 probability of choosing node 2 as their next node, and 0.51 of choosing node 4. 'Pheromone laying' is represented by 'updating probabilities'.

At every time step during the simulation, ants can be launched from any node in the network. Each ant has a random destination node. Ants move from node to node, selecting the next node to move to according to the probabilities in the pheromone tables for their destination node. Arriving at a node, they update the probabilities of that node's pheromone table entries corresponding to their source node i.e. ants lay the kind of pheromone associated with the node they were launched from. They alter the table to increase the probability pointing to their previous node. When ants have reached their destination, they die. Take as an example an ant in the network of Figure 10 that is launched at node 3 with destination node 2, and has just traveled from node 4 to node 1. This ant will first alter node 1's table corresponding to node 3 (its source node) by increasing the probability of selection of node 4; it will then select its next node randomly according to the probabilities in the table corresponding to its destination node, node 2. (Note that just for the purpose of illustration this particular ant traveled an ineffective route from node 3 to 2, namely 3-4-1-2. This is possible because ants do not necessarily walk the shortest way, due to the randomness of their walk.) In this way, ants moving away from their source node can only directly affect those ants for which this source node is the destination node. This is unlike the trails of bidirectional trail laying ants, in which a trail laid in one direction can directly affect ants travelling in either direction. However, the ants which can be directly influenced by an ant travelling from a source

node S to a destination node D will include those travelling from D to S; these are the very ants which could be expected to have most influence on ants travelling from S to D, and so ants travelling from S to D may have a strong influence on ants subsequently travelling that route via their effect on the ants travelling on the opposite route.

This way of directly updating probabilities differs from the way ants lay pheromones, but is functionally equivalent. The tables used here give the probabilities of alternative choices between paths directly, whereas the pheromones of real ants are basically a code that is effectively converted into probabilities by the ant's nervous system. The method used to update the probabilities is quite simple: when an ant arrives at a node, the entry in the pheromone table corresponding to the node from which the ant has just come is increased according to the formula:

$$P = \frac{P_{old} + \Delta P}{1 + \Delta P}$$

Here p is the new probability and Δp is the probability (or pheromone) increase. The other entries in the table are decreased according to:

$$P = \frac{P_{old}}{1 + \Delta P}$$

Since the new values sum to 1, they can again be interpreted as probabilities. Note that a probability can be reduced only by the operation of normalization following an increase in another cell in the table; since the reduction is achieved by multiplying by a factor less

than one, the probability can approach zero if the other cell or cells are increased many times, but will never reach it.

A primary requirement of this work was to find some simple methods of encouraging the ants to find routes which are relatively short, yet which avoid nodes which are heavily congested. Two methods are used. The first is to make Dp, the value used to change the pheromone tables, reduce progressively with the age of the ant. When the ant moves at one node per time step, the age of the ant corresponds to the path length it has traced; this biases the system to respond more strongly to those ants which have moved along shorter trails. The second method, which is related to the first, is to delay ants at nodes that are congested with calls to a degree which increases with the degree of congestion. This delay has two complementary effects:

- It temporarily reduces the flow rate of ants from the congested node to its neighbors, thereby preventing those ants from affecting the pheromone tables which are routing ants to the congested node, and allowing the probabilities for alternative choices to increase rapidly.
- Since the ants are older than they otherwise would have been when they finally reach the neighboring nodes, they have less effect on the pheromone tables.

To determine the route for a call from a particular node to a destination, the largest probability in the pheromone table for this destination is looked up. The neighbor node corresponding to this probability will be the next node on the route to this destination. The route is valid if the destination is reached, and the call is then placed on the network, unless one of the nodes on the route is congested; in that case the call fails to be placed

on the network. In this way, calls and ants dynamically interact with each other. Newly arriving calls influence the load on nodes, which will influence the ants by means of the delay mechanism. Ants influence the routes represented by the pheromone tables, which in their turn determine the routing of new calls. The load on the network at any given time influences which calls can subsequently be placed on the network and which calls will fail; which of course determines the load at a later stage.

A network initialized with random or uniform entries in the pheromone tables will not initially contain any useful information about consistent (i.e. non-circular) call routes, let alone good routes. It therefore makes little sense to examine network performance during this phase. However, even in the absence of calls on the network, the ants will bias towards shortest paths. There are three mechanisms contributing to this:

- The shortest routes will be completed first, and will subsequently direct other ants to their sourcing nodes first.
- Shorter routes involve less branching, so the number of ants travelling over these routes and laying pheromones will be larger than on longer and more branched routes.
- Ants travelling over shorter routes will be younger when they arrive, and will therefore exercise more influence on the pheromone tables.

After a short time the highest probabilities in the pheromone tables of each node will define relatively short routes, and circular routes will have been eliminated. When the routes are sufficiently short, calls can safely be put on the network; subsequent adaptation will then be influenced by any congestion caused by calls. All ant networks were

therefore initialized with equal probabilities for neighbor nodes in each pheromone table, and allowed to run for a fixed period before calls were applied.

So the basic principles for this method can be characterized as follows:

- Ants are regularly launched with random destinations on every part of the system.
- Ants walk randomly according to probabilities in pheromone tables for their particular destination.
- Ants update the probabilities in the pheromone table for the location they were launched from, by increasing the probability of selection of their previous location by subsequent ants.
- The increase in these probabilities is a decreasing function of the age of the ant, and of the original probability.
- This probability increase could also be a function of penalties or rewards the ant has gathered on its way.
- The ants get delayed on parts of the system that are heavily used.
- The ants could eventually be penalized or rewarded as a function of local system utilization.

3.5.2 Summary

A load management technique for networks has been presented, that is modeled on the trail laying and following abilities of ants. Routing tables in telecommunications networks could be replaced by tables of probabilities. These probabilities also indicate good routes, and have more information about how good alternatives are. To obtain a

good set of probabilities, mobile agents move around on the system to update them. These agents are modeled on ants, and the way they update the probabilities is similar to the way ants organize routes by laying pheromones. Therefore in this piece of work it calls the mobile agents 'ants', and speaks about 'pheromone tables' instead of probability tables. Each node has its own 'scent' or 'pheromone' that attracts ants with this particular node as a destination. An ant is launched on a source node and updates the probabilities in the pheromone tables for this node, while randomly walking according to the probabilities for the node that it is trying to reach. The behavior of each individual ant is simple and has a large random component. Some additional mechanisms were added to the system, like congestion dependent delay; increasing influence for younger ants and ants on weak trails.

This is a completely decentralized adaptive control system for telecommunications networks. The principles of the ant algorithm are simple and general. The general framework for ant-based solutions presented here is capable of solving load balancing problems in large networks, both circuit switched and packet switched.

3.6 Network Survivability

This project [Black 97] is sponsored by the Defense Advanced Research Projects Agency (DARPA) of the US Dept. of Defense and various industrial sponsors of The Open Group Research Institute. Its objectives are:

1. Automatically halt the spread of malicious mobile code, by innoculating hosts or firewalls against the attacker. The danger of attacks by mobile code is growing, due to active networks, mobile agents, and growing reliance on Web applets.
2. Strengthen an intrusion detection and response system (IDS) so it works even when components are compromised. This survivability will make the IDS far more difficult to subvert and turn against its owner. The approach should be general enough to apply to other IDSes.
3. Architect the directory and configuration facility of the Common Intrusion Detection Framework (CIDF), so that IDS components can find each other and can verify each other's authenticity and authority, even in a large-scale IDS.

Their research is on the infrastructure for distributed detection and response to network attacks. One such environment is the Internet, where the potential for attacks and the limits of current responses are demonstrated by recent SYNflooding attacks on Internet sites. A SYNflooding attack exploits resource exhaustion via partial establishment of TCP connections. The attacker bombards a TCP port on a target host with numerous TCP connection requests (SYNs) that cannot be completed because they appear to come from random source IP addresses. The target host creates a half open TCP connection for each request and attempts to synchronize (SYN+ACK) with the host at the (random) source IP address. This synchronization fails by timing out when the source IP address is for a host that does not exist or will not respond (e.g., a host behind a firewall). The SYNflooding attack exhausts resources used by half open connections, causing the target to reject or rapidly time out new connection requests. SYNflooding attacks are most effective when

directed at ports that can expect to receive arbitrary connection requests (e.g., web server ports), and hence have no reasonable way to detect forged source IP addresses. The goal is to provide a distributed infrastructure for automatic or semi-automatic detection, tracing, and suppression of attacks. A promising approach to building a distributed response infrastructure is to employ mobile agents. In this paper, they define a mobile agent as an encapsulated active object (code and data) that performs computation and can make its own movement decisions. Mobile code is involved in the execution of

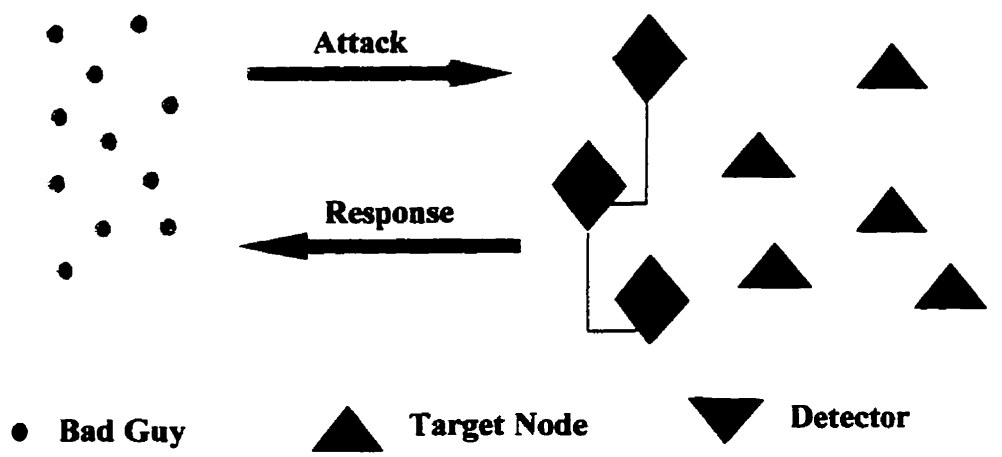


Figure 11: Network Survivability

the mobile agent, and may also be part of the mobile agent's data (e.g., a mobile agent transporting a bug fix along with instructions for applying it). Mobile code is a fundamental requirement for flexibility of response and system evolution because it is impossible to predict in advance the responses and preventive measures that will be required by future attacks. The ability of mobile agents to make independent decisions (e.g., reevaluation of trust assumptions, redirection of communication in response to

failure or compromise) can help avoid long decision cycles for situations in which human intervention is not required. Its working principle could be described by the following figure. But the detailed information on how to realize this function is unavailable.

3.7 Network-Aware Mobile Programs

This work was done in the Department of Computer Science at the University of Maryland, College Park. In papers ([Ranganathan 97], [Ranganathan 96]), they investigate network-aware mobile programs, programs that can use mobility as a tool to adapt to variations in network characteristics. They present infrastructural support for mobility and network monitoring and show how "adaptalk", a Java-based mobile Internet chat application, can take advantage of this support to dynamically place the chat server so as to minimize response time. For different applications, different resource constraints are likely to govern the decision to migrate, e.g. network latency, network bandwidth, memory availability, server availability. Their conclusion was that on-line network monitoring and adaptive placement of shared data-structures can significantly improve performance of distributed applications on the Internet. Network-awareness is particularly important to applications running on mobile platforms which can see rapid changes in network quality.

In order to adapt to network variations, mobile programs must be able to decide when to move, what to move and where to move. There are three types of network variations which may be cause for migration: (1) population variations, which represent changes in the distribution of users on the network, as sites join or leave an ongoing distributed

computation; (2) spatial variations, i.e. stable differences between in the quality of different links, which are primarily due to the hosts' connectivity to the Internet; and (3) temporal variations, i.e. changes in the quality of a link over a period of time, caused presumably by changes in cross-traffic patterns and end-point load. Spatial variations can be handled by a one-time placement based on the information available at the beginning of a run. Adapting to temporal and population variations requires dynamic placement which needs a periodic cost-benefit analysis of current and alternative placements of computation and data. Dynamic placement decisions have two partially conflicting goals: maximize the performance improvement from mobility and minimize the cost of mobility. If an opportunity for improving performance presents itself, it should be capitalized upon; however, reacting too rapidly to changes in the network characteristics can lead to performance degradation as the performance gain may not offset the mobility cost.

In traditional client-server methods, the server is static. For the work cited here, a mobile chat server for their experiments has been developed. The application, called adaptalk, monitors the latencies between all participants and locates the chat server so as to minimize the maximum response time. This application was selected because it is highly interactive and requires fine-grain communication. If such an application is able to take advantage of information about network characteristics, it is expected that many other distributed applications over the Internet would be similarly successful. The resource that governs the migration decisions of adaptalk is network latency. To provide latency information, they have developed Komodo, a distributed network latency monitor. To

evaluate if mobile applications can take advantage of network-awareness, the performance of adaptalk with and without mobility was examined. Their evaluation had two main goals: (1) to determine the performance benefits, if any, of network-aware placement of the central chat server over a network-oblivious placement; and (2) to determine if dynamic placement based on online network monitoring provides significant performance gains over a one-time placement based on initial information. The results indicated that on-line monitoring and dynamic placement can significantly improve performance of distributed applications on the Internet.

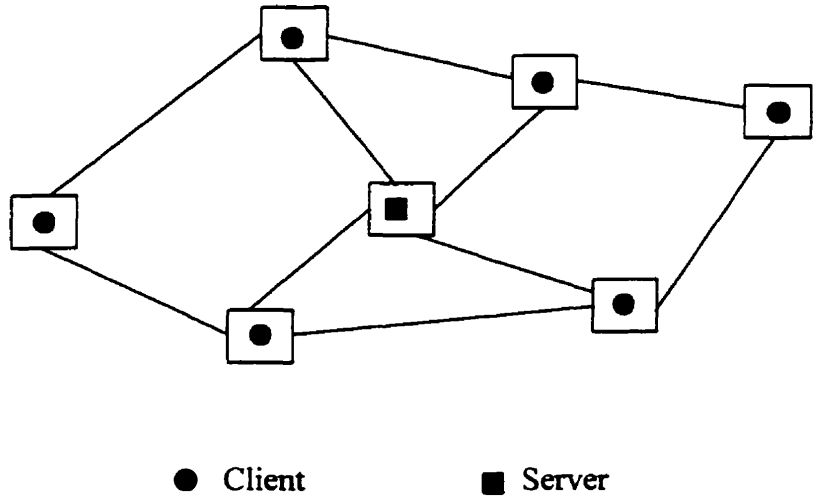


Figure 12: Network-Aware Mobile Server

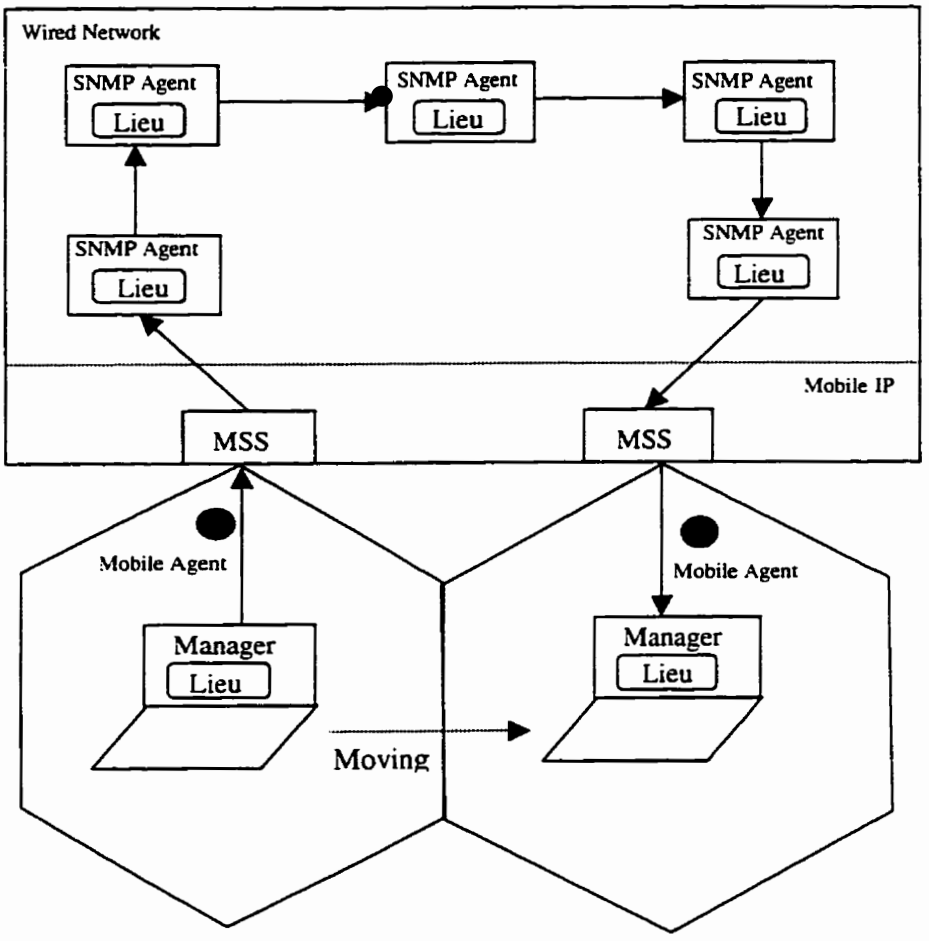
There are two main features of the adaptalk mobility policy. (1) Continuous tracking of the instantaneously most-suitable-site and (2) Deferral of server-motion until the potential for a significant and stable performance advantage has been seen. The first feature allows it to quickly take advantage of opportunities for optimization; the seconds helps ensure the gain is greater than the cost. The goal of adaptalk is to minimize the maximum

response-time seen by any participant. The suitability of a participating machine as the location of the central server as characterized by the maximum network latency between it and all other participants. The machine that achieves the lowest measure is designated the most-suitable-site (Figure 12).

3.8 Mobile Network Manager

The work introduced in the papers ([Sahai 97], [Sahai 98-1], [Sahai 98-2]) built a Mobile AGENT environment for distributed Applications (MAGENTA) and applied it in the domain of network management in order to implement a Mobile Network Manager (MNM). The architecture of MAGENTA comprises of lieus and agents as shown in Figure 13. A lieu is a place or location where an agent can originate, reside, execute and interact with the system as well as with other agents. A MNM is a network manager executing on a portable computer and managing a network comprising of static network components executing SNMP agents. The MNM queries the SNMP agents about the network management variables and obtains their values. In order to demonstrate the functionalities of the MAGENTA environment they have enabled the MNM through mobile agents. The MNM and the SNMP agents are integrated with MAGENTA lieus so that they are capable of sending, receiving and storing the agents. The agents are utilized to study the performance of network components, install software, audit the network components (by collecting the information on disk usage, users utilizing the machine, application configuration etc) and for network discovery. The MNM operates in either tethered mode or in wireless mode. The MNM typically sends a large number of agents to implement a variety of desired actions. After sending the agents it might disconnect, after

completing its itinerary the agent waits at the last lieu of its itinerary. If the MNM had quit gracefully informing all other lieus about its disappearance, the agent waits for the MNM lieu to connect back. In case the MNM had disappeared abruptly, the agent tries to



Initial Cell of the Portable Final Cell of the Portable

MSS - Mobile Support Station

Figure 13: Mobile Network Manager in the wireless mode

reach the MNM lieu being unaware of its absence. As it fails to reach the MNM lieu the agent again waits for the MNM lieu to connect back. The lieu hosting the agent at this moment informs all other lieus about the disappearance of MNM lieu. When the MNM connects back all the lieus are informed about the appearance of the MNM lieu. The agent waiting for the MNM at the last lieu returns back to it with the results.

A MNM operating in tethered mode makes a PPP/SLIP connection to the wired network. In such a case after the connection is established the portable computer is assigned an internet address local to the LAN to which it connects. This connection can be torn down after launching the agents and might be reestablished once the administrator deems it fit to reconnect. In wireless computing the total domain is divided into cells. Each of the cells has a Mobile Support Station (MSS) of their own. The responsibility of serving the Mobile Host (MH) as it moves from one cell to another cell changes from one MSS to another. The operation of MNM in the case of a wireless network is shown in Figure 13. The MNM sends an agent from a particular cell and moves to another cell. The agent moves on its itinerary and tries to come back to the MNM after accomplishing its task. The agent is directed to the new cell of the MNM by the MSS. The agent thus returns with the results of tasks performed by it.

The suitability of mobile agents in the context of its utilization in mobile user aware applications as compared to existing client-server mechanisms was studied. Especially in the case of mobile users, the network bandwidth usage and the response times are important factors as they have a fallible and costly link which should not be overused.

Also the response times must be known so that they can reconnect back after an approximately known time. They thus implemented two MNMs, one utilizing mobile agents and the other utilizing the client server mechanism. It is evident from the experiment that the client-server technique is suitable for SNMP monitoring when it is to be done for a limited amount of time. When large number of samples are desired, in terms of bandwidth utilization it is better to use the mobile agent technique. In their case the threshold obtained was of 300 samples. With a sampling rate of 0.5 second, they derived that if a NMS intends to observe a SNMP variable for more than 2 minutes and 30 seconds it is preferable to use the mobile agent technology. Also the response time in retrieving the samples were initially found better in the client-server technique. But for larger number of samples the response time was better in sending an agent to retrieve the values than using a client-server technique to continuously query and obtain the values of the SNMP variable from the SNMP agent.

CHAPTER 4 MONITORING Of NETWORK ELEMENTS

4.1 Introduction

This chapter describes the development of a Network Management System developed by the author employing mobile agents. The system developed here uses mobile agents for building an NMS. It integrates Advent's SNMP and ObjectSpace's Voyager package. The system is implemented in Java which was selected for its platform independence and security features. Mobile agents require portable code systems like Java and the Java Virtual Machine where classes can be loaded at runtime over the network.

Network Management Systems (NMS) typically operate on large, heterogeneous, and asynchronous networks, and they must provide features including security, distributed control, portability, and dynamic operation in response to failures of system components. The high level of complexity inherent in an NMS requires a correspondingly high level of abstraction within the development and methodology of the NMS software.

The use of agents is ideally suited for an NMS. In fact SNMP, CMIP and TNM use the concepts of OO design and agent technology, Moreover, intelligent agents have already been demonstrated performing some network management activities in commercial packages. For examples, agent based packages have been implemented to manage and to perform operating system maintenance on networks, and to monitor specific devices in networks.

The use of mobile agents in the management of large networks is still in its early conceptual stages. Ideally, agents should be created using a language which is portable between network platforms, mobile, and autonomous in their operation. This would allow reliable agents to be created which implement an NMS system without requiring vendor implementation or support. This greatly increases the flexibility, maintainability, and versatility of the NMS system.

The system under development here investigates mobile agents for use in monitoring network elements, communications among mobile agents, and the use of a web browser as a control interface. The system was implemented in the Java language which is selected for its security features and platform independence. Agent servers are established within a network and allow agents to be created which are mobile throughout the network. These agents perform monitoring operations such as monitoring input or output traffic counts through the network element. This system is implemented using the Java language and integrated by Java JDK1.1.3, ObjectSpace's Voyager [Voyager Package] and Advent SNMP package [Advent SNMP Package]. An overview of Voyager and the SNMP package are included here for completeness.

4.2 ObjectSpace Voyager Package

Voyager is a Java-based mobile agent system that exhibits several unique and innovative features [Joseph 97]. Virtual Object is Voyager's key communication framework and a tool to support inter-agent communication and control. The following description is

derived from Voyager's users guide [Voyager Package] and the IEEE Internet Computing special issue on mobile agents [Joseph 97].

The Virtual Object is a kind of post-processed proxy to ObjectSpace's Voyager's remote object or agent. Other systems that use an RPC-like mechanism, like RMI, require the developer to go through a series of steps to describe the interface and then the implementation of an object. Voyager takes any existing Java class (source code or class file) and modifies it with the "vcc" tool, a Virtual Code Compiler, which creates the Virtual Object mirror of the source class. Once any arbitrary object is processed with vcc, it exhibits some of the properties of an agent: it can be migrated from server to server and accessed remotely by other virtual objects in an RPC-like fashion; and it can have its own life-cycle.

The communication facility in Voyager is very flexible. Voyager allows for remotely enabling any Java class without modifying the class in any way. Once the class is enabled, an instance of the object can be easily created anywhere in a network and sent regular Java messages. Voyager supports synchronous, oneway, and future message modes. If an object moves, it leaves behind a trail of secretaries that will forward messages to the object's new location. When the object dies, its secretaries also die.

Because Voyager treats an agent just like any other object, users can also create remote agents and send them messages as they move. References to remote virtual objects can be passed as parameters to methods and can be serialized. Method calls across address

spaces have the same semantics as local method calls; they are fully polymorphic, allowing access to local and remote objects through virtual object proxies using exactly the same syntax.

Voyager's migration mechanism is also quite novel. While Voyager provides an agent server (named "voyager") like all the other Java mobile agent systems, it is not necessary to run such a server on all the nodes in networks. This is because a virtual object can migrate not only between agent servers, but also to the Java runtimes of other arbitrary virtual objects. Voyager supports mobility of all serializable objects, not just agents. A class does not have to be modified in any way to gain full support for mobility. An Object can move to a new program, even if it is receiving messages.

The life span of an object defines how long an object should live before it dies and is reclaimed by the system. Voyager typically contains a distributed garbage collector that kills objects when they have no more local or remote references. Agents, on the other hand, can roam a network independently of external references and thus require a more flexible life-span scheme. Voyager supports five different kinds of life span:

- An object can live until it has no more local or remote references.
- An object can live for a certain amount of time.
- An object can live until a particular point in time.
- An object can live until it becomes inactive for a specified amount of time.
- An object can live forever.

An object's life span can be changed at any time. This flexible scheme means an agent can be launched and it is known that the agent will be killed after a specified time period.

Most web browsers allow an applet to establish a network connection only back to the server where it originated. This means that most platforms for distributed computing, such as RMI and many CORBA implementations, allow an object in an applet to communicate only with objects located in the server. Voyager addresses this problem by including an integrated software router that allows a server to act as a gateway, thereby enabling full applet-to-applet and applet-to-program connectivity. This facility also allows Voyager agents to move freely between applets and applications.

A persistent object has a backup copy in a database. A persistent object is automatically recovered if its program is unexpectedly terminated or if it is flushed from memory to the database to make room for other objects. Voyager includes seamless support for object persistence. Voyager includes a high-performance object storage system and works with most popular relational and object databases.

Many of these features of the Voyager were utilized in the development of this agent based network monitoring system.

4.3 Advent SNMP Package

This package is a group of Java class files that provide Java programmers a simple API for developing network management applets and applications that use SNMP. Many

different architectures are supported, such that SNMP can be used in the web browser, on the server, or even on managed elements that have a Java Virtual Machine. Applets developed using this package can be loaded from the network or from local disks and run in any Java enabled browser.

This package can be divided into four categories:

1) SNMP Variable classes

The ancestor of all SNMP variable classes is an abstract class called SnmpVar. This class contains abstract methods for printing, ASN encoding, ASN decoding, etc.

2) SNMP Communication classes

The Advent SNMP package uses the SnmpAPI class to manage sessions created by the user application, manage the MIB modules that have been loaded, and store some key parameters for SNMP communication, e.g. SNMP ports to be used. A SNMP application (manager or agent) often needs to manage multiple sessions on account of interacting with multiple SNMP peers. The SnmpAPI class has a list of sessions attached to it and monitors each of the sessions for timeouts and retransmits. It enables a few methods across all sessions, e.g. checking if responses have come in on any of the sessions, etc. Multiple threads can work with a single SnmpAPI instance

The SnmpSession class is used to manage a session with a SNMP peer. This allows communication with more than one host via a single session, but also allows for separate sessions for hosts which require frequent communication in an application. Each session

runs as a separate thread and provides functions to open sessions (on a particular local port if needed), supports synchronous or asynchronous sending and receiving of SNMP requests, check for responses and timeouts, and closes sessions.

Interaction between the SNMP manager and the agent is via SNMP protocol data units (PDUs). The SnmpPDU class is used to provide the variables and methods to create and use the SNMP PDU. The methods include adding null valued variable bindings and printing all variable bindings.

3) SNMP MIB related classes

MIBmodules allow an SNMP managed agent to let users know about the structure and format of data available on the agent. The MIB modules are usually specified in a MIB module file, which needs to be parsed to understand the syntax and structure of the data available on the agent. The MibModule class provides a means to parse and use the data available in a MIB module file. Each MibModule instance is created from a MIB module file, and user can load and unload MIB modules by creating and deleting these instances. The instance contains all the nodes of the MIB tree as well as defined traps and textual conventions.

The MibNode class represents a node in the parsed MIB tree. A list of instances of this class is contained in a MibModule and represents the MIB tree. This class may also be contained in an SnmpOID instance. A number of attributes and methods in the MibNode class are provided to simplify development of applications using the MIB definitions.

The LeafSyntax class is used to represent any unique syntax, including textual conventions, that are defined in a MIB module. For example, INTEGER (SIZE (1..5)), would have its own LeafSyntax class instance that represents this syntax. For leaf nodes in the MIB tree, the MibNode instance contains a LeafSyntax reference. Thus for a MIB leaf node user can use the LeafSyntax to determine if a value is within the allowed range.

4) Miscellaneous classes

The miscellaneous classes include the client interface class, and the Exception sub-classes.

4.4 System Design

In the NMS developed here, mobile agents are used to monitor the network elements and test communication among mobile agents. The use of a web browser as a control interface is also investigated and developed. The system includes the following objects: SnmpControlApplet object, SnmpGet object, ResultGet object, DrawWindow object, ParseOptions object and MessageAgent object. Figure 14 shows the system's object model with the main objects described as follows.

SnmpControlApplet

This object is the control center of the whole system. It is a Java applet running on web browser providing a standard and platform independent interface. It manages the agents roaming on the network and the returning agents. It manages four SNMP objects:

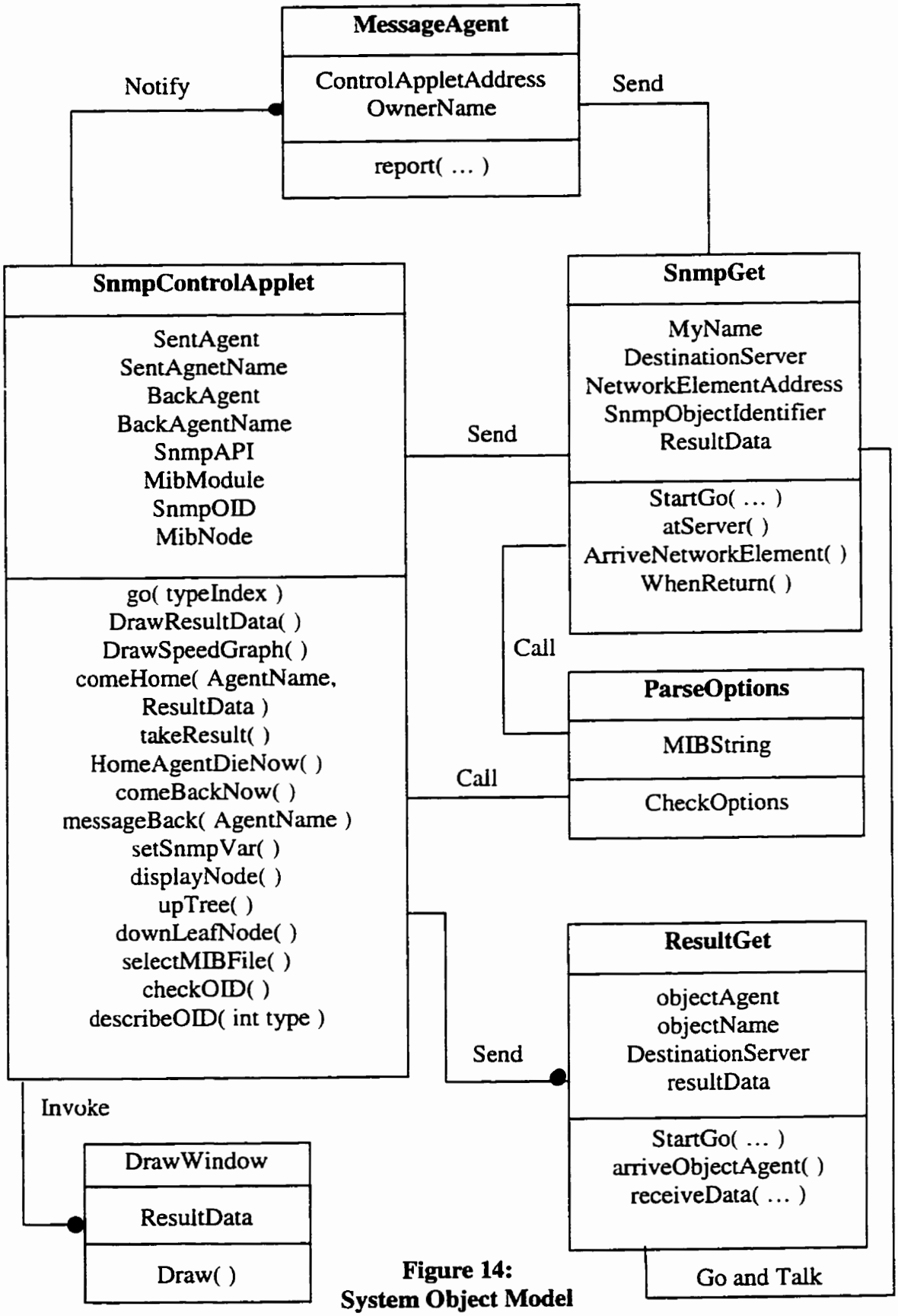


Figure 14:
System Object Model

SnmpAPI, MibModule, SnmpOID and MibNode which allow the user to enter or to choose the SNMP Object Identifier. The MibModule is based on the MIB module file. the user can select different MIB module files. SnmpOID is used to interpret Object IDs and relate SNMP variables to MIB data. It includes methods for printing, converting, and associating MIB nodes. It is also used to check whether the entered OID is valid. MIBNode represents a MIB node in a MIB module tree. It is derived from parsing a MIB module. It contains references to its parents and children. This object can create virtual objects of SnmpGet, ResultGet and send them to remote hosts. At the same time it manages the list of these object's virtual references allowing communication with them. Communication between these objects uses synchronous and oneway message modes. The object will also create a DrawWindow object to display the network data that the agents bring back.

SnmpGet

The SnmpGet object contains the destination address, network element address and SNMP object identifier to be queried on the remote designated site. It is created and sent by SnmpControlApplet. After arriving at the destination node, the object will open an SnmpAPI thread to execute SNMP processes. It will also instantiate SnmpSession to communicate with an SNMP entity. It will then create and send PDUs or receive PDUs in this session. After successfully communicating with Snmp Agent, the object will receive and collect management data. Because the system supports persistent data, this agent can be saved or flushed to a database on disk to prevent loss from a host shutdown or a

broken link. This object has two types: type one will come back automatically after completing a job, while type two will wait for further instruction.

ResultGet

This object is created and sent by SnmpControlApplet. It will go to a designated destination and talk to a designated agent. Upon retrieving the data, it will return automatically.

DrawWindow

This object is created by SnmpControlApplet. The SnmpControlApplet will send result data to this object as parameters for displaying.

ParseOptions

This object is used only for checking the Snmp command string. It is called by SnmpControlApplet and SnmpGet.

MessageAgent

After an SnmpGet (type two) object completes its work, it will create the MessageAgent object to inform the control applet that it has completed its job. This object will then go to the control center server, communicate with SnmpControlApplet and die automatically.

4.5 Operating Principle

Figure 15 illustrates the system's operating principle. The diagram illustrates the agents, SNMP agents, web browser and control applet relationships, it also shows how agents roam on a network and communicate with each other. Since most browsers allow an applet to establish a network connection only back to the server where it originated, the system establishes the Voyager server on a local host acting as the gateway enabling full applet-to-applet and applet-to-program connectivity. This facility also allows Voyager agents to move freely between applets and remote host or network elements. In some situations, the agents can not go directly to the network node, but can go to a nearby site and communicate with a SNMP entity. This will be the situation when a managed network element does not support Java or uses another application. Many simple or dedicated network devices such as routes and bridges will fit into this category.

The system supports persistence. If the host that the agent resides on shuts down, the agent still exists on disk. When the system recovers the control applet sends a message to this agent, the agent could recover from disk to memory and continue execution under the control of the control applet. Upon invocation, the Voyager server is initialized on the host that the agent will go to and assign a port number including the local server identification. The web browser then opens an HTML file which includes the control applet. After inputting some parameters including the SNMP object identifier that the agents will monitor, the user could send two types of agents to go directly to the network elements or to a nearby host. These agents will try to communicate with the SNMP

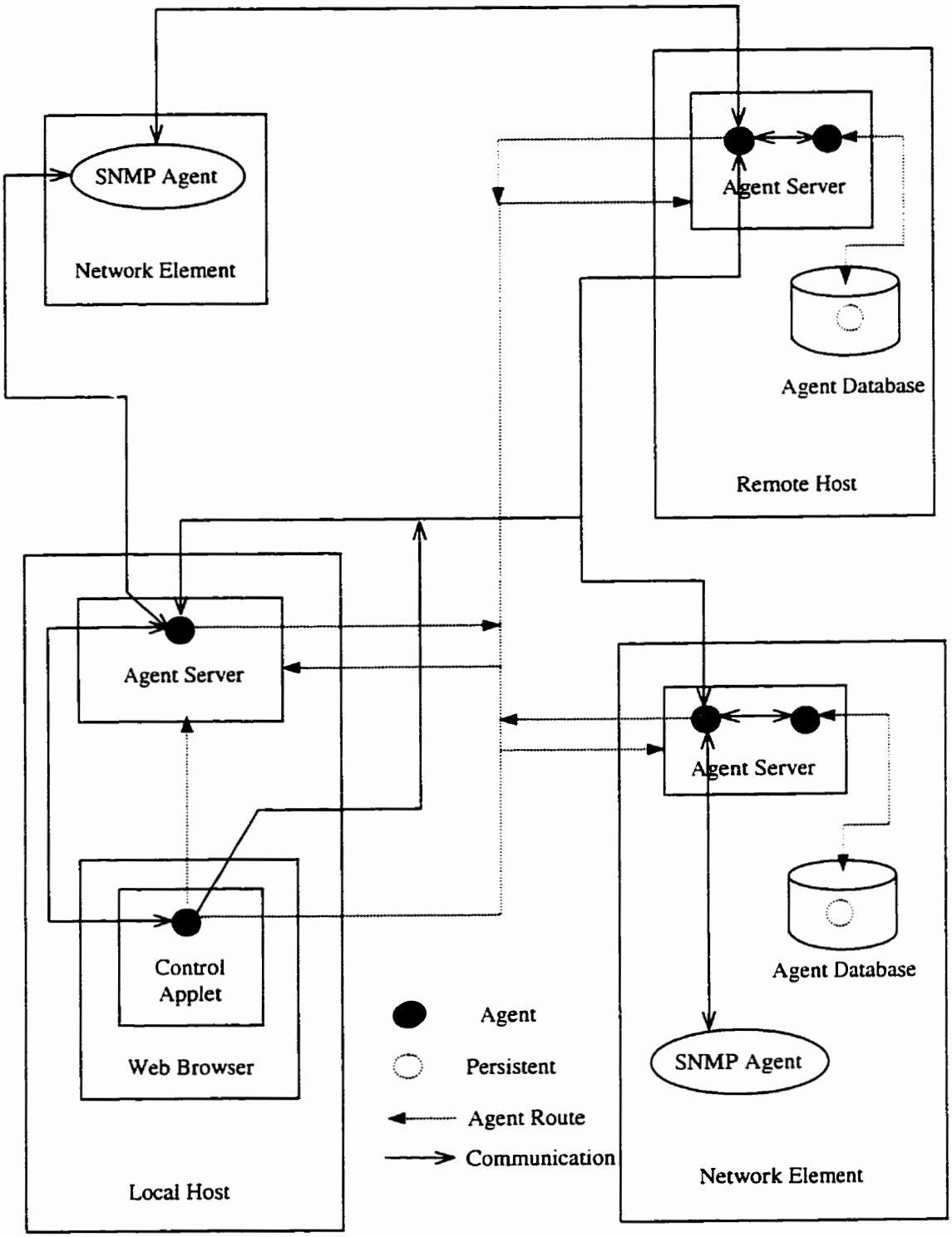


Figure 15:
System Operating Principle

agents and obtain values according to a user's input. After completing the work, the agent will create a simple message agent (MessageAgent) and send it back. The control applet maintains this agent's working state. The control applet could send another agent (ResultGet Agent) to this agent's site and exchange data with this agent locally, then retrieve the data back, or the applet could send a message to this agent and let this agent come back automatically. This allows full exploration of Voyager's message communication mechanism (synchronous, oneway, and future message modes) with the objective of exploiting future negotiation among mobile agents for solving more complex network problems.

4.6 System Implementation

Figure 16 shows the SnmpControlApplet applet running on the hotJava web browser. In the Destination Address and Network Element, the user inputs the destination IP address the agent will go to and network element IP address the agent will monitor. They could be same or different according to the specific situation. The user can also input the domain name string. The Network Element Port Number and SetValue text fields are not used in the system developed here, but are left for future use. The agent could be sent to the remote node for execution at a future time. Because the hosts could potentially be spread all over the world, different hosts may be in different time zones. After the agent arrives at the new host, it will get local time zone, transfer the time zone information to the server where the agent originated and will execute according to starting site time zone. If the time appointed has passed, the agent will execute immediately, otherwise, it

will wait until the time is due. If the Object Identifier is the description string, the agent will ignore the start time, time duration and time interval.

The interface supplies a MIB node browse function. The user should choose the MIB file, for example, rfc1213-MIB. Then according to this MIB file, the user could select a sub node from the sub node window, or from the current node text field the user could choose the parent node by pressing the return key. The user can then go up or down the tree through all nodes. Of course, a user could directly input the OID number or OID string in the Object Identifier field. The Agent Name field is actually the Voyager's agent alias name, according to this name the system could look for this agent on remote hosts. Below the Agent Name field is the message window, in this window, the system will display all necessary information to the user including the activity of agent.

The last two panels are used for managing mobile agents. The first list on the right is the names of outside mobile agents, which show the agent's type and working state. The left text window displays the information corresponding to the agent listed. The second list on the right shows the names of returning agents. The left text window also shows information about the corresponding agent in the right list, including result data. In the first right list, the user sees the agent's working state clearly. For example, when type two agents finish their work, they will send a message agent back to notify the applet, and the applet will display the word "completed" close to the agent's name. When the user presses the "Take Result" button, the applet will send a ResultGet agent to the remote host and take the data back. Figure 16 shows the "agent3 (Query Agent)" as this kind of

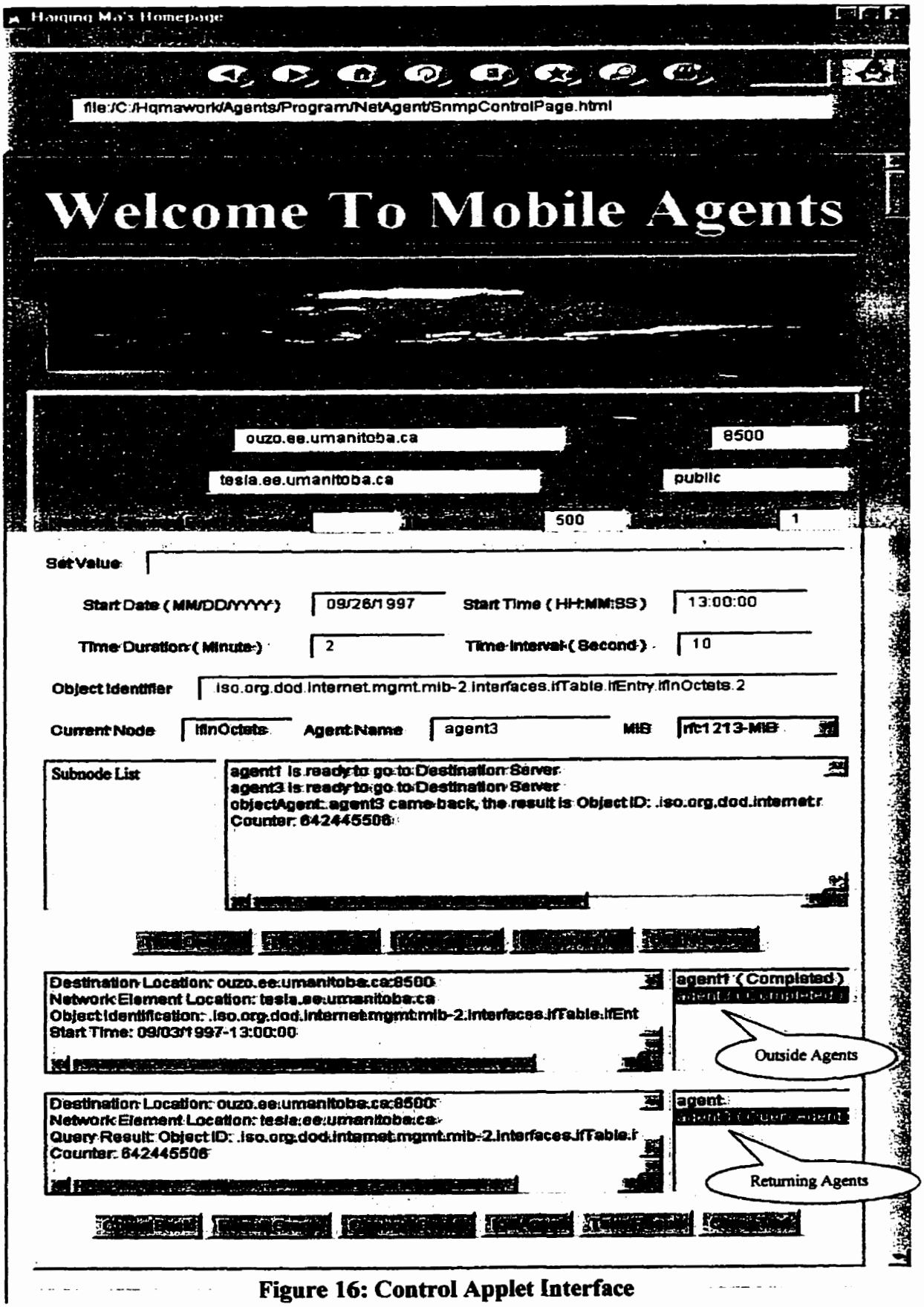


Figure 16: Control Applet Interface

agent. The agent takes data from agent3 which remains on the remote host. When the user presses the "Come Back" button or directly double clicks the agent3, it will automatically come back and leave the first panel to be displayed in the second panel. The user can look at the result data graphically or kill the agents locally.

4.7 Conclusions

An NMS system was developed combining mobile agents, SNMP, Java and web browser technologies. This system is considered a test-bed for the ideas rather than complete solutions for management systems. Most of them can be implemented in standard ways, but I argue that the use of mobile agent is beneficial. The system has demonstrated its strong capabilities in monitoring network elements. When a connection is broken, the agent can still work at remote site and come back when the connection recovers. The system extends SNMP architecture for network management and reduces bandwidth for network management systems by moving agents to the place the data is located. The agent persistence supplies stable properties for the system. When the remote system is down, the agent and its data can still exist on hard disk and recovers automatically after the system works again. The control applet supplies the user with a friendly and standard interface to control and communicate with mobile agents through a web browser. The Java agents successfully roam different platforms (Windows95, Sun Solaris, etc.), illustrating the architecture platform independence which increases deployment flexibility.

CHAPTER 5 RESOURCE NEGOTIATION IN CONNECTION SETUP

5.1 Introduction

In this chapter, I will use Quality of Service, network signaling and burstiness measure as the bases for the resource negotiation models. These concepts will first be introduced for better understanding of my resource negotiation models.

5.1.1 Quality of Service

So far, the Internet has been following the "best-effort" delivery model. There is no admission control and the network offers no assurance about the delivery of the packets. Most of the initial applications on the Internet were elastic in nature, in that they tolerated packet delays and packet losses, and hence they could be served by the "best-effort" model. But, there are new applications today on the Internet that do not necessarily follow the "best-effort" model. These new applications such as voice and video, require a finite bound on the end-to-end delay. So current packet-switching networks providing the best-effort service are no longer adequate for current multimedia distributed applications having stringent performance requirements in terms of through-put, delay, jitter and loss rate. Consequently, the functional and management perspectives must be re-considered. The support of a distributed multimedia application involves several system components which provide services under several constraints. These constraints are usually expressed as parameter values associated with the provider services (provider performance), and by parameter values specified by customers (customer desired Quality of Service).

A description for "Quality of Service"(QoS) is the frequently used general definition by CCITT [CCITT I.350]: "QoS is the collective effect of service performance which determine the degree of satisfaction of a user of the service". An improvement of the QoS definition could be stated that QoS "is described in terms of a set of user-perceived characteristics of the performance of a service, expressed in a user-understandable language and manifests itself as a number of parameters, all of which have either subjective of objective values". QoS parameters include parameters which express the system behavior performance and parameters which express other service characteristics as protection (security) or priority. There are two types of QoS parameters:

- Objective parameters that can be directly observed and measured at the points at which the service is accessed by the service user, e.g. delay and throughput.

- Subjective QoS parameters that depend upon user equipment, e.g speaker quality, and opinion and can not be measured directly.

Judgements on Service Quality are provided from two sides: the provider and the customer of the service. In real life, both opinions might be different. Different parameters are used depending on which layer of the protocol stack is being described. At the network layer, terms such as bandwidth and error rate are used, whereas applications use abstract terms such as frame rates and image quality.

There has been much work in recent years in the field of QoS management for communication networks. This work concentrated on resource allocation in communication networks in order to assure specific QoS characteristics for requested new connections. Much of this work is related to Asynchronous Transfer Mode (ATM) networks which are expected to provide specific QoS guarantees if requested by the user. There have also been several proposals for including resource reservation schemes within the Internet in order to allow for some form of performance guarantees.

QoS is defined by service parameters of the provider which satisfy customer requests. Each time a relation is established, parameter values of the requested services are identified and compared with performance parameters of the provider. The process is called negotiation and its result is a cooperation contract. In order to initialize a contract, the partners could re-negotiate customer demands with respect to the provider current performance. Negotiation and re-negotiation use the operation called matching the parameter values. Matching the parameter values is a cornerstone operation because of the diversity of parameters and their value spaces. When this operation is performed, the contractual values between a customer and its provider become a basis to further evaluate the QoS.

5.1.2 Network Signaling

There are two circuits in ATM network: Permanent Virtual Circuit (PVC) and Switched Virtual Circuit(SVC). The PVC is always present and can be used at will. The SVCs have to be established each time they are used. Connection setup is handled using a highly

complex ITU protocol called Q.2931. Several ways are provided for setting up a connection. The normal way is to first acquire a virtual circuit for signaling and use it. To establish such a circuit, cells containing a request are sent on virtual path 0, virtual circuit 5. If successful, a new virtual circuit is opened on which connection setup requests and replies can be sent and received.

The normal procedure for establishing a call is for a host to send a SETUP message on a special virtual circuit. The network then responds with CALL PROCEEDING to acknowledge receipt of the request. As the SETUP message propagates toward the destination, it is acknowledged at each hop by CALL PROCEEDING. When the SETUP message finally arrives, the destination host can respond with CONNECT to accept the call. The network then sends a CONNECT ACK message to indicate that it has received the CONNECT message. As the CONNECT message propagates back toward the originator, each switch receiving it acknowledges it with a CONNECT ACK message.

The Connection Admission Control (CAC) of an ATM Network is the set of actions taken by the network at connection setup time in order to establish whether a connection can be accepted or rejected. If the new call is accepted, the network should provide the requested QoS for all the connections. The network should check to see if it is possible to handle this connection without adversely affecting existing connections. Closely related to connection admission control is the technique of reserving resources in advance, usually at call setup time. For example, once the traffic descriptor gives the peak cell rate, the network has the possibility of reserving enough bandwidth along the path to handle

that rate. Bandwidth can be reserved by having the SETUP message earmark bandwidth along each line it traverses, making sure, of course, that the total bandwidth earmarked along a line is less than the capacity of that line. If the SETUP message hits a line that is full, it must backtrack and look for an alternative path or reject the connection request.

The new multimedia applications need QoS negotiation to ensure that the requirements of the users are satisfied. But most existing QoS negotiation protocols [Hafid 97] are only concerned with the communication quality in terms of QoS parameters, such as throughput, delay and jitter. Furthermore the negotiation results, in response to the user request, are restricted to an acceptance or rejection of the request. This implies that a second attempt of the user cannot take advantage of information obtained through the first request to change.

5.1.3 Burstiness Curve

In this thesis, I use a burstiness measure or curve to explore the trade-off between bandwidth and buffer size to achieve no cell loss for a given message. The burstiness curve is an approach that indirectly models the traffic source random process [Scott 95]. Cruz [Cruz 91] studied the performance of deterministic fluids by using two parameters (σ, ρ): ρ is the service rate and σ is the maximum buffer content if the fluid flow is fed into an infinite buffer served at rate ρ. By using a buffer of length σ and service rate ρ, no cell loss will occur and the delay jitter will be bounded by σ/ρ. Low [Low 91] [Low 93] extended Cruz's work to define a burstiness curve by treating σ as a function of ρ.

A message is a bounded, nonnegative function m(t); $0 \le t \le T$, where m(t) is the instantaneous rate in cells per second or cps, and $T < \infty$ is its duration. Let M be the set of all messages. Suppose m is served by an infinite buffered server at a constant rate μ cps. The buffer is initially empty. The number of cells buffered at time t is:

$$X_m(t) = \max_{0 \le s \le t} \int_s^t [m(r) - \mu]dr$$

So the maximum buffer occupancy $\max_t X_m(t)$ is:

$$b_m(\mu) = \max_{0 \le s \le t \le T} \int_s^t [m(r) - \mu]dr$$

If the message name, m, is clear from the context, we write $b(\mu)$ and x(t) instead of $b_m(\mu)$ and $X_m(t)$. We may also omit one or both end-points 0 and T in the notation above when the domain of maximization is clear from the context. So the maximum buffer occupancy $b(\mu)$ is a function of bandwidth μ allocated to m; $b(\mu)$ is a burstiness curve defined as follows:

A function $b(\mu)$, $\mu \ge 0$, which is nonnegative, convex, and strictly decreasing for $\mu < M$, with $b(M) = 0$, and $-db/d\mu < \infty$, is called a burstiness curve.

According to this model, the bandwidth-buffer tradeoff of every message has a burstiness curve and every burstiness curve describes the bandwidth-buffer tradeoff of some messages. That means, given a bandwidth and buffer which is determined by the burstiness curve for message m, the transmission of the message will not lose a cell. Based on the burstiness curve, when there is not enough bandwidth, we can request a

larger buffer for compensation or we could look for an optimized combination between the bandwidth and buffer for the best price.

5.2 Mobile Agent Negotiation Models

In the mobile agent models, the call user, the network service provider and agents can delegate their tasks to agents and vest them with authority to act on their behalf. The integration of mobile agents for signaling purposes does not seem sensible for today's networks, since centralization of service control represents the foundation of the networks architecture. However, new telecommunications architectures, such as TINA seem to be more appropriate candidates for the incorporation of mobile agents. The evolving view of connection establishment for connection-oriented services involves a contract negotiation process between a user agent and a network agent. The connection establishment involves negotiations between multiple networks and user agents, in which the parties agree to set up connection to transmit the agreed information streams in a manner to guarantee the agreed QoS, and at an agreed price. The purpose of the contract negotiation for the user agent is to get the price that will maximize the surplus for the user.

5.2.1 Operation Principle

The user's objective for using network services has often been considered as getting the best QoS as possible. That is, if there is no price constraint, the user will always ask for the best QoS service. However, if prices are charged to users, then they have to weigh the benefit of service against the cost charged. Here the benefit of the service means the willingness-to-pay the service price by the user. The benefit may be some monetary

measure of how much a user values the service. If the benefit exceeds the cost, the users will most likely use the service. The difference between benefit and cost is defined as consumer surplus. Most recent contract negotiation proposals [Scott 95] assume that a user's objective for using the network is to maximize their consumer surplus. But here it should be mentioned that the price does not necessarily mean monetary, it may be used as a control parameter to adjust the traffic balance of network or network resource usage.

In the mobile agent negotiation models developed here, I directly use the resources as negotiation objects because any QoS requirement will finally lead to the resource reservation. In this project, the resources include the bandwidth and buffers in each node along the route. The bandwidth and buffers are "substitutable resources" according to burstiness curve to meet a service quality. The network directly offers its bandwidth and buffers for rent, and the users purchase resources freely to meet its desired service quality. Users package the resources into services that best meet their needs. Since the network only guarantees the availability of purchased resources, it is the users' responsibility to shape their traffic in order that the allocated resources can provide the desired QoS. I use bandwidth, buffer and price of the bandwidth and buffer as negotiation parameters. I also assume the negotiation objective of the user is to maximize his or her consumer surplus. The surplus is the result of the benefit minus the cost as following.

$$\text{Consumer surplus} = \text{Benefit}(\text{ resources }) - \text{Cost}(\text{ resource })$$

Therefore:

$$\text{Negotiation objective} = \text{maximal}(\text{ surplus })$$

$$= \text{maximal}\{\text{ benefit}(\text{ resources }) - \text{cost}(\text{ resources})\}$$

At the beginning of the negotiation process, both sides know their objectives. A user agent has information about its resource requirements and valuation of the service, but is unaware of the network's available capacity and the market price demand for network services. On the other hand, a network agent knows the available capacity and resource prices but is unaware of user's valuations of services and their desired resources. The dominant mechanism in negotiations is as follows. The user agent first asks for the required resources from the network agent, including the bandwidth and buffer. Then the network agent replies to the user agent providing the available capacity and the demand price for the resources. A user agent calculates the maximal consumer surplus by selecting the best combination of the bandwidth and buffer according to the burstiness curve function it contains and decides how much of each resource to request for its desired application. The network and user agents agree on the price, amount of resources for each connection. If the cost exceeds the benefit, the user agent can bargain based on its negotiation strategy that the user empowers to it. Also the network agent can adjust the price according to the negotiation policy that the network owner authorizes for it.

5.2.2 Description of Models

I have designed two different mobile agent negotiation models as in Figure 17 and Figure 19, which use different strategies for negotiation. At the same time, I also designed two corresponding message-based models (Figure 21) corresponding to their mobile agent models for comparison.

In model 1, when the calling user initiates the connection, they will create one calling agent on behalf of themselves. The calling agent brings the address of the destination, the

range of the required bandwidth, the burstiness curve function, the benefit function and the negotiation strategy parameters. The calling agent will first go to switch1 and tell switch agent1 its range of bandwidth and buffer requirements. The switch1 will reserve maximal resources it can for the resources that the calling agent requires. Then the switch agent1 will clone itself and the clone of switch agent1 will go directly to the destination for a negotiation meeting. And the calling agent will go to next switch to repeat this work until it reaches destination. The called agent will also join the meeting representing its owner. If switch1 can meet the calling agent1's minimal requirements, the switch1 will reserve the resources for the negotiation and clone the switch agent1. Then the clone of switch agent1 will go to the destination for the negotiation meeting and calling agent1 will go to the next node. If switch1 does not have enough resources, the requirement of calling agent1 is rejected and the calling agent1 will stop travelling and go back to its initial place. If all agents reach an agreement, they will bring this contract and go back to their destinations for resource commitment. Sometimes the agents will adjust the resources according to the contract by relaxing the reserved reservation. The advantage of this model is that it can obtain a global maximal surplus. Because the negotiation occurs in one meeting place, it is easy to negotiate a global maximal surplus. That is, the calling agent first calculates each switch resource cost on individual nodes, then gets the total maximal surplus by comparing the benefit and total cost. It may not be the maximal surplus for some individual nodes under certain resource choice, but it is the global maximal surplus for this resource selection. The formula for the global maximal consumer surplus is:

$$\text{global maximal surplus} = \max_{\text{Min}(R) \le R \le \text{Max}(R)} \{ \sum_{i=1}^{n} \text{surplus}(\text{node } i) \}$$

or:

$$\text{global maximal surplus} = \max_{\text{Min}(R) \le R \le \text{Max}(R)} \{ \text{benefit}(R) - \sum_{i=1}^{n} \text{cost}(\text{node } i) \}$$

In the above formula, the maximum value can only be obtained within the global combined range of the calling agent request resource range and switch agents available resource ranges. The n is the total number of the involved nodes in the connection. R represents resource.

In model 2, when the calling user starts the call, the calling agent will be created and sent to switch1. Then calling agent will negotiate directly with switch agent1, switch agent1 will not create its clone and send it to destination as in model 1. There is no central negotiation meeting in this model. There is only individual negotiation on the way that calling agent visits every switch it travels and discusses with each switch agent. The calling agent first negotiates for current resources. When the calling agent reaches an agreement with switch agent1, it will go to next switch and switch agent1 will reserve the resources. The calling agent will discuss with next switch agent and proceed to the destination. After reaching an agreement with all switch agents and the called agent, the calling agent will go back and inform every switch agent to commit the reserved resources. When the calling agent can not reach an agreement with any one of the switch agents, it will stop travelling immediately and return back. Along its return path, the calling agent will notify every switch agent to release the reservation. The advantage of

this model is that it does not need to send every agent to one destination for a meeting, but it is not as efficient in attempting to get a global total surplus because of the distribution of the negotiation. In this model, it only realizes the local maximal surplus for individual nodes. That is, the total surplus is the sum of individual maximal surplus. The total surplus is not the maximum, but the local surplus is the maximum. The formula is as following:

$$\text{total surplus} = \sum_{i=1}^{n} \underset{\text{Min(R)}\le R\le \text{Max(R)}}{\text{maximum}} \{ \text{surplus(node i)} \}$$

or:

$$\text{total surplus} = \sum_{i=1}^{n} \underset{\text{Min(R)}\le R\le \text{Max(R)}}{\text{maximum}} \{ \text{benefit(node i)} - \text{cost(node i)} \}$$

In this formula, the maximum value can only be searched within the node's local combined range of the calling agent request resource range and switch agents available resource ranges. The n is the total number of the involved nodes in the connection. R again represents the resource.

In order to compare these two mobile agent-based models with message-based methods, I also implemented the two corresponding message-based models. These two message-based systems' negotiation mechanisms are the same as the mobile agent models except that all agents are static and they negotiate the resources by messages as in Figure 21. They use the same negotiation algorithms as in the mobile agent models so we could compare them and evaluate which method is better under various conditions.

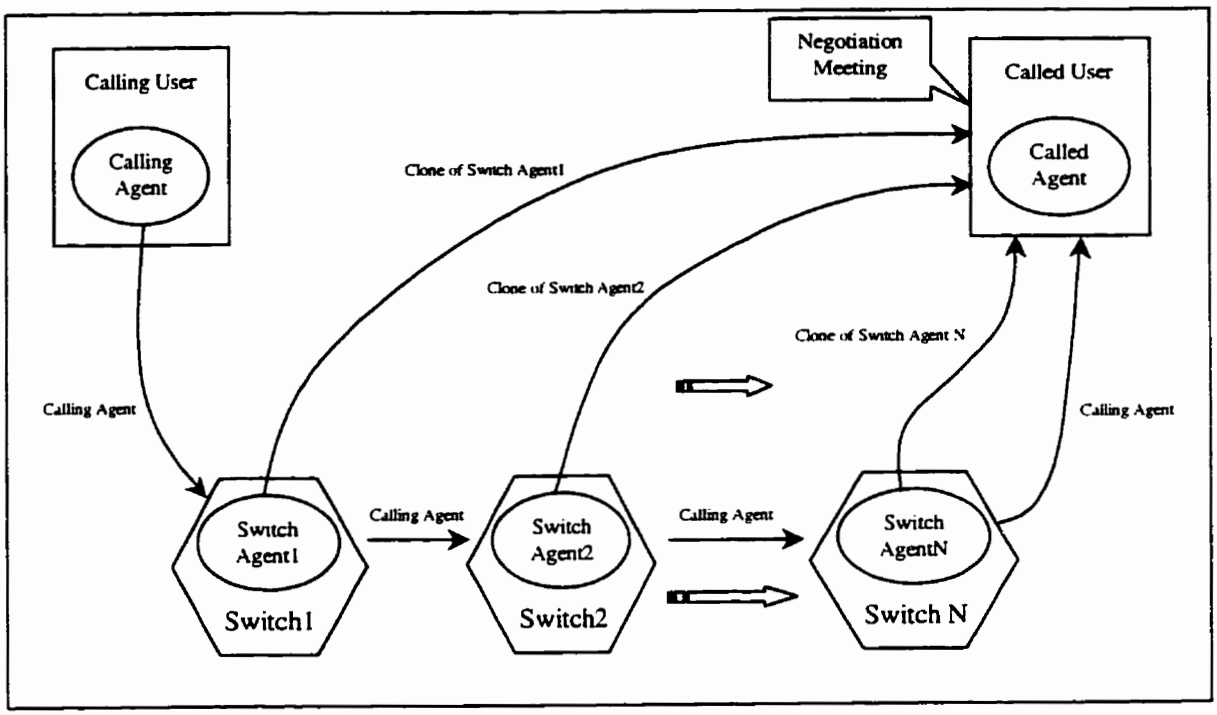


Figure 17: Mobile Agent Model 1 – Call for Negotiation Meeting

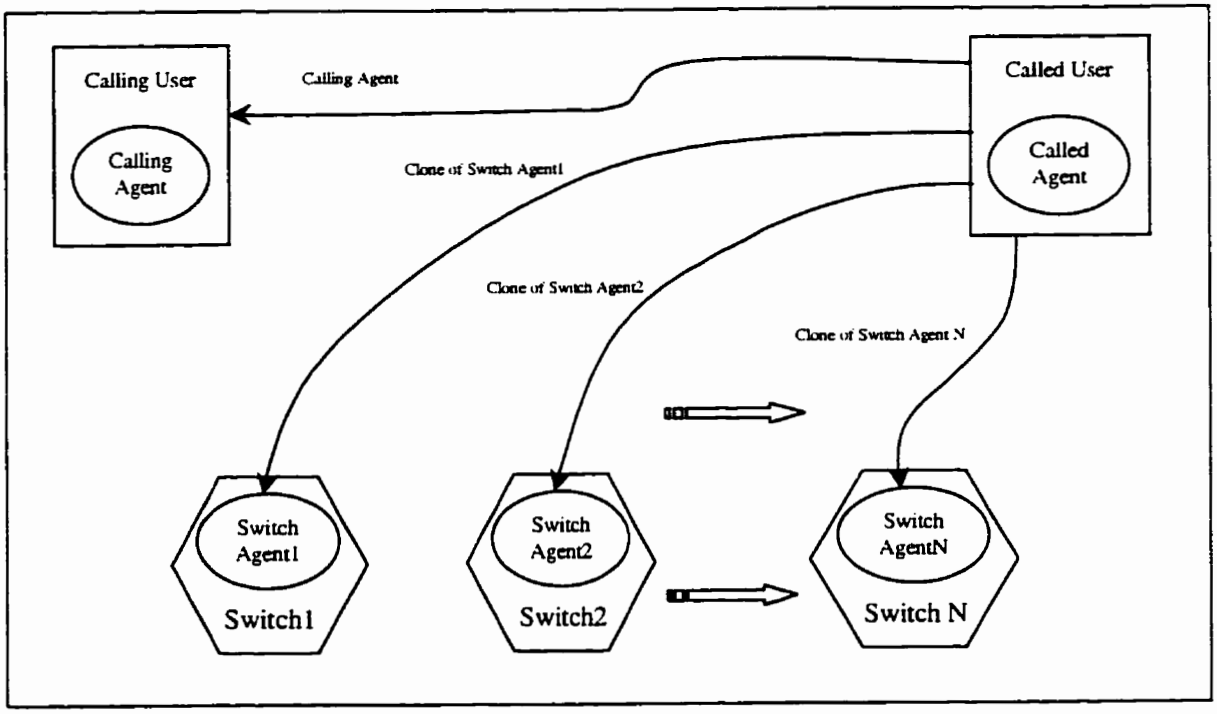


Figure 18: Mobile Agent Model 1 – After Negotiation Meeting

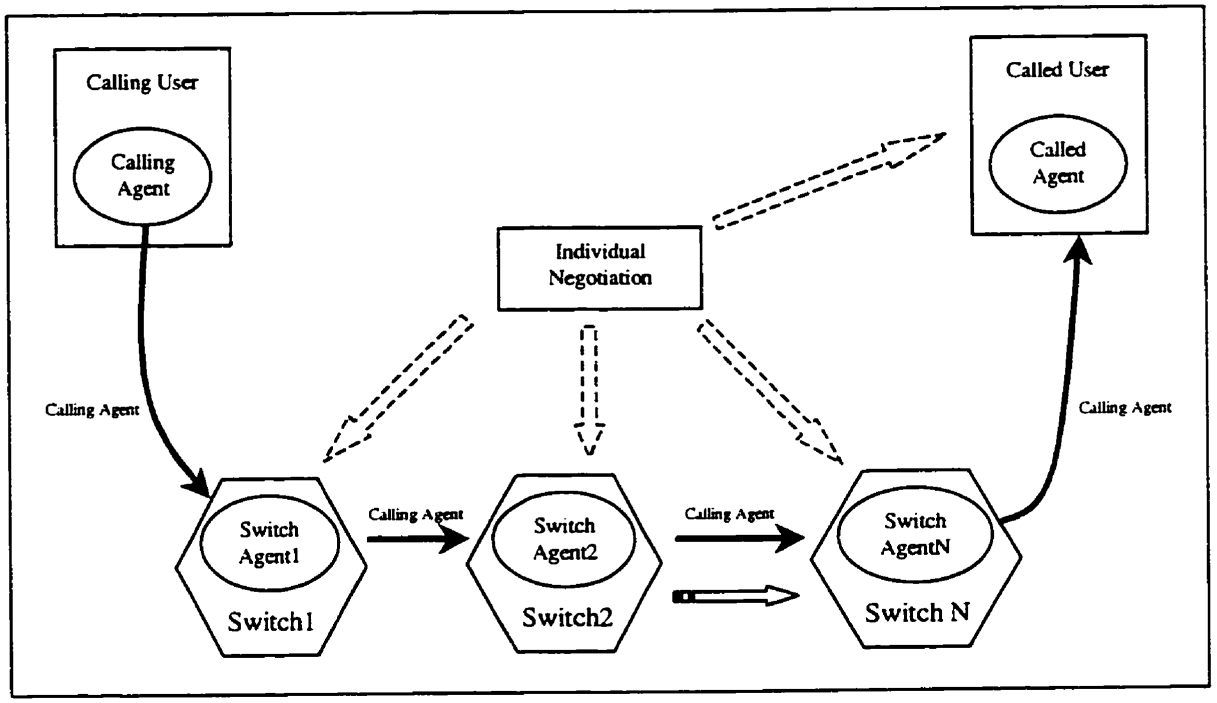


Figure 19: Mobile Agent Model 2 – Call for Individual Negotiation

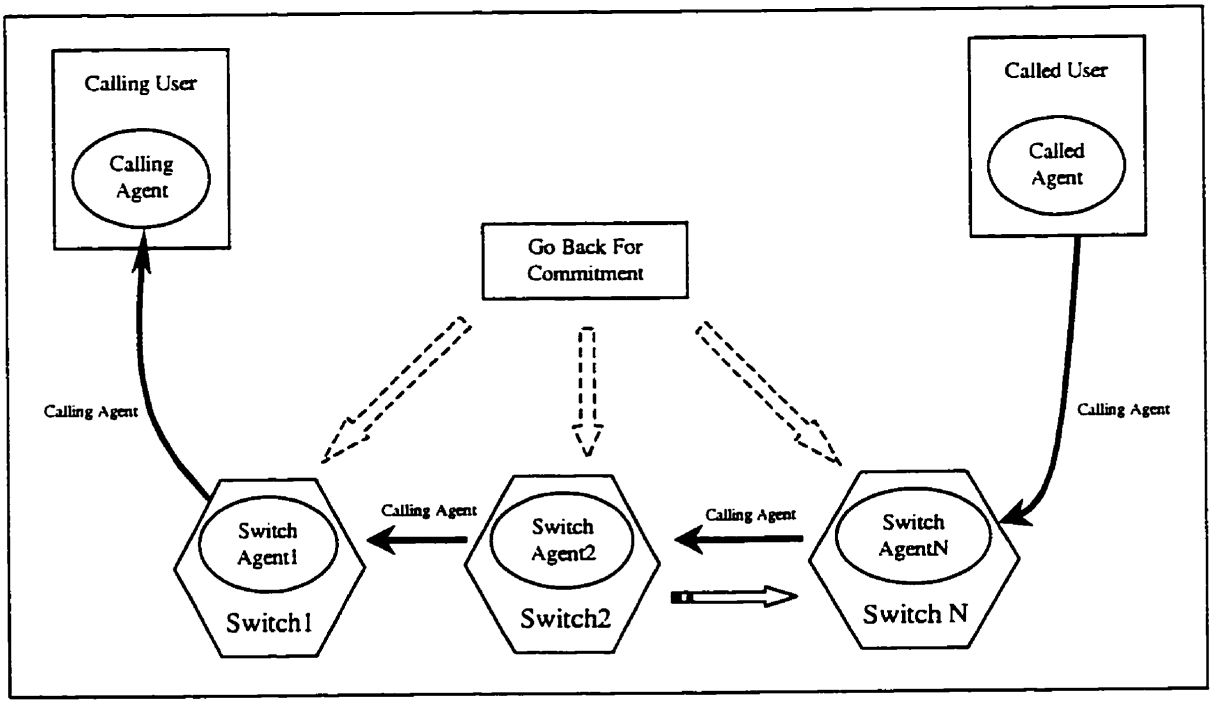


Figure 20: Mobile Agent Model 2 – Return for Commitment

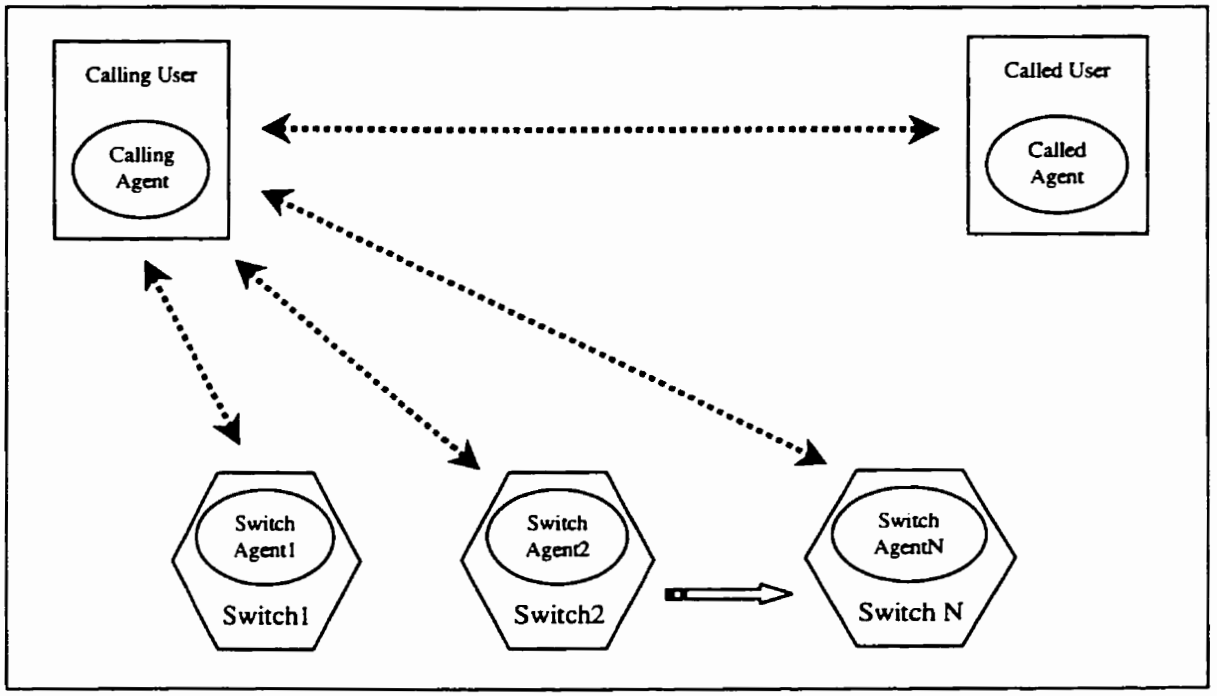


Figure 21: Message-based Method – Negotiation By Messages

5.3 System Implementation

In the simulated implementation of the models, two control centers were set up as in Figure 22. One is a simulated platform control center, the other is a connection control center.

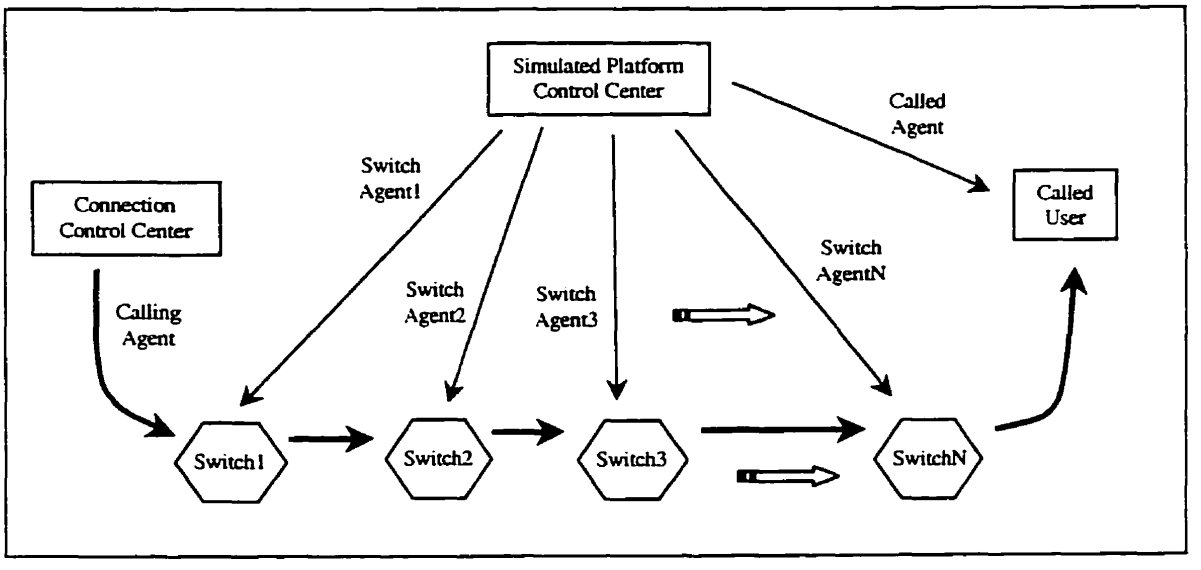


Figure 22: Simulated Implementation System

The simulated platform control center is responsible for sending the switch mobile agents to remote nodes and these switch agents stay at remote nodes and run as the agents representing the owners of the switches. Before creating and sending out these switch agents, the simulated platform control center should input the remote node address, switch agent name, switch resource capacity, resource cost functions and negotiation strategy. The switch resources are bandwidth and buffer. The resources cost functions are bandwidth cost and buffer cost functions as in Figure 23. But the price is not fixed, the network service provider can authorize the negotiation strategy for the switch agent. In this implementation, the simulated platform control center could input the step down

number, the quota of decreasing price in every step, and the bargain iteration number. In this way, the network could adjust its resource pricing at different times.

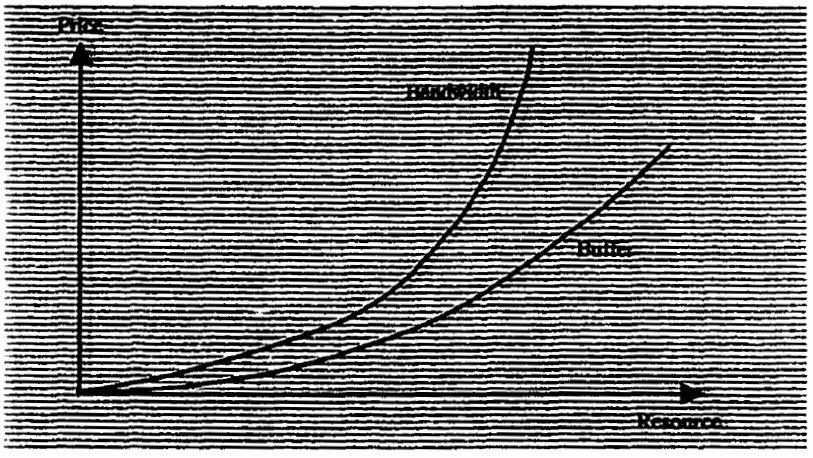


Figure 23: Resource Cost Functions

In this manner, the network could supply an adaptive service to users as in the real world. For example, if a user agent asks for a better price, after the bargain iterative number, the switch agent can decrease the price by the quota allowed for every step. The switch agent cost can be reduced a limited times determined by the step down number. The control center can also send router tables to switch agents. By supplying router tables with different routing information, the control center can create point-to-point connections as well as a multi-path environment for the connection. Also the simulated platform control center could send the called agent to its destination to create a connection endpoint. The called agent should check the connection request and compare it with its capacity. If the request exceeds its capacity, it will send a reject message.

The connection control center is responsible for initiating the call, monitoring the activity of the agent and calculating different statistic information. When the connection control

center creates the calling agent, it should input the address of the destination, the range of the required bandwidth, the burstiness curve function parameters, the benefit function parameter and the negotiation policy parameters. By using the burstiness curve function as in Figure 24, the calling agent can look for the optimized combination between the bandwidth and buffer or when switch agent does not has enough bandwidth, the calling agent can adjust the bandwidth by applying more buffer according to the burstiness curve function.

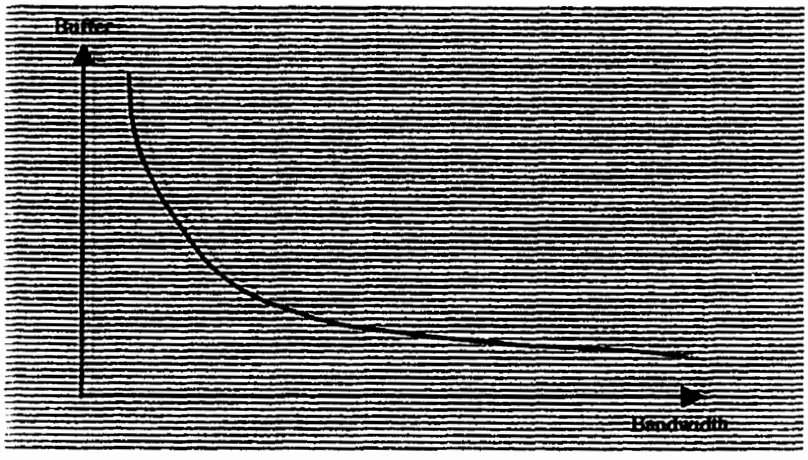


Figure 24: Burstiness Curve

The objective of the calling agent is to look for the maximal consumer surplus within its requirement range. The benefit function represents the willingness to pay for the resource as in Figure 25. Here I only use the bandwidth vs. price. The buffer factor is already considered in the benefit function, because there is a relationship between the bandwidth and buffer within the burstiness curve. When the bandwidth is selected, the buffer is also determined. So it is possible to include the buffer contribution to the price as a benefit function.

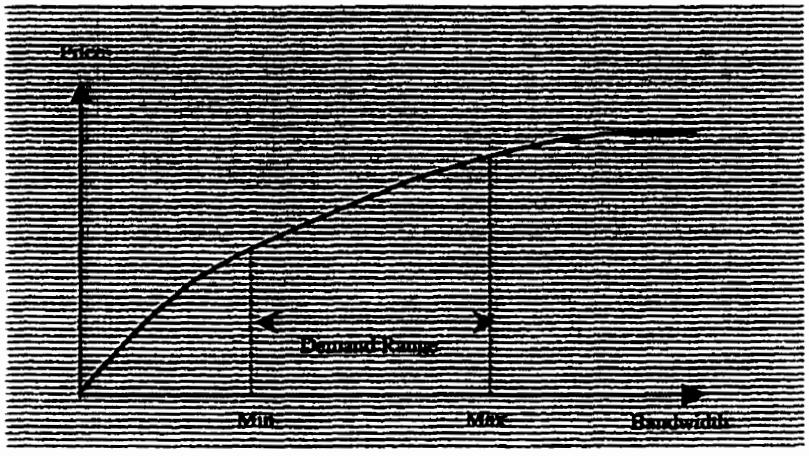


Figure 25: Benefit Function

The negotiation strategy is expressed by several parameters: the iteration times, the step up number and the quota of increasing price for every step. The negotiation iterative step is the same as described in the switch agent. For simplicity, the bargaining only occurs when the resource cost exceeds the benefit. When the cost is lower than the benefit, the calling agent will search the best point in the benefit function for maximal consumer surplus as in Figure 26.

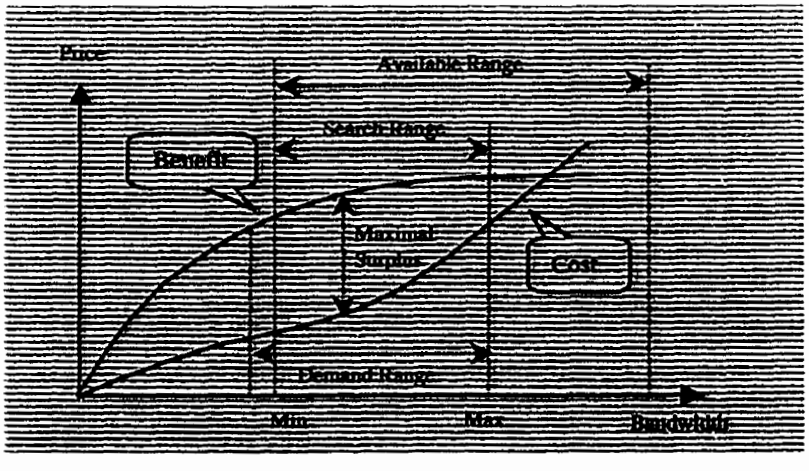


Figure 26: Maximal Consumer Surplus

As mentioned in the previous section, the final search range is determined by the combination of demand range and available range. The consumer surplus and search range are different in two models. In model 1, the maximal surplus is global so the search range is also the combination of all nodes' capacity and request range. But in model 2, the maximal surplus is local for each node, so the search range is only the combination of local node's capacity and request range.

The implementation includes two mobile agent negotiation models and their corresponding message-based models as mentioned above. In mobile agent negotiation model 1, the switch agent will clone itself and the cloned agent representing its creator will go to the destination to negotiate with other mobile agents. In model 2, the switch agent will not clone itself. It will directly negotiate with the calling agent. In the message-based models, all agents are kept static and just negotiate with each other by message passing.

In the implementation of this system, minimal time was spent on the interface and complete consideration of the boundary condition in the program, since the value for that aspect of the research would be minimal and the cost of implementation very high.

When the connection is set, the connection control center will display the connection setup cost time, the negotiation iterative number, the committed resources, the total surplus, the deal price and the information of all agent activities. By using this

information, we can compare the two agent models and their corresponding message approaches.

5.4 Test Results

Figures 27 to 32 are the test results based on the two models. I implemented them in two different environments. One is for a three switch connection, one is for a five switch connection. In the test, I compare the mobile agent methods with their corresponding message methods, as well as provide a comparison between the two mobile agent models. From these test result figures, we could observe the relationship between negotiation iterative number and time cost on establishing a connection. The negotiation numbers on the figures are the average number for each switch, not the total iterative number, because it is easier to compare for different cases.

Figure 27 is the result under model 1 for three switches. When the negotiation number is less than 10, the message method is faster than mobile agent method. But with the increase of the iterative negotiation number, the mobile agent time cost increases very slowly whereas message-based method rises very quickly. So when the number exceeds 10, the message time cost is larger than the mobile agent's. When the iteration number becomes even greater, the gap between message and agent increases. This indicates that the mobile agent method is better than the message approach when the connection establishment needs more iterative negotiation. Actually 10 iterative negotiation messages are not all used for negotiation of resources, half of them are used for initiating, resource commitment or releasing and control signaling. It should be noticed that not

only the iterative negotiation number affects the connection cost time, but also the number of switches or routers along the connection path will have influence on the time cost. The total number of negotiation messages will increase with the number of intermediate nodes. For example, when the calling agent communicates with each switch agent six times for resource negotiation, the total messages needed are different for the five switch and three switch situations. The five switch connection will need 30 messages to complete the negotiation, but the latter only needs 18 messages to complete the call. So the total time cost will be different. With the increasing number of the switches along the path, the negotiation messages will increase and the connection time cost will also increase. In practice, the connection of five switches or routers is not a common case. For example, from The University of Winnipeg to the ECE department of University of Manitoba, there are five routers although they are in the same city. For most connections on the Internet, more than five switches or routers are needed. For example, from the University of Winnipeg to the CBC homepage host, there are 10 switches or routers. In my experiments, I selected three and five nodes between source and destination as an illustration. If the message time cost is bigger than agent's in such cases, it will cost much more time in normal connection situations. For model 2 (Figure 28) for three switch case, a similar result is obtained as that in Figure 27.

The five switch configuration is shown in Figure 29 and 30. Here (from the beginning of the chart) the mobile agent method begins to exceed the message-based method. From the beginning with the negotiation number of 6, and 5 intermediate nodes, the total negotiation number becomes larger. The iterative negotiation number of 6 is the minimal

with each switch agent because of necessary control messages. This result shows that even under less iterative negotiation, if there are many intermediate nodes along the path, the mobile agent method will spend less time than the message method for establishing the connection.

From Figure 31 and 32, we can see that model 1 is better than model 2 in comparing mobile agent methods. In the three-switch case, the result is not obvious. But in five-switch situation, the difference becomes clear. My analysis for this result is: in model 1, the operation is parallel. After completing the negotiation, all agents will go back to their homes at the same time. But in model 2, it works sequentially, the calling agent visits each node step by step.

From the experimental results, we conclude that mobile agent method is better than message-based method in normal situations. Among the two mobile agent models, model 1 is better than model 2 not only for its time cost, but also for global optimization of the consumer surplus as mentioned in the previous section. The model 2 can do global optimization, but it needs the calling agent to repeat its route forward and backward which will not only cost much more time, but also increase the processing complexity of the signaling greatly.

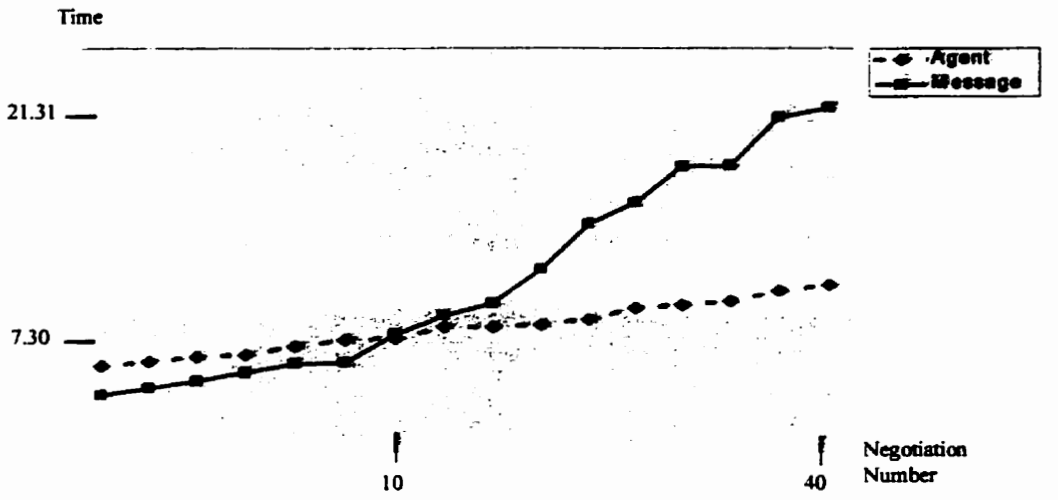


Figure 27: Time Cost In Model 1
For Three Switches Connection

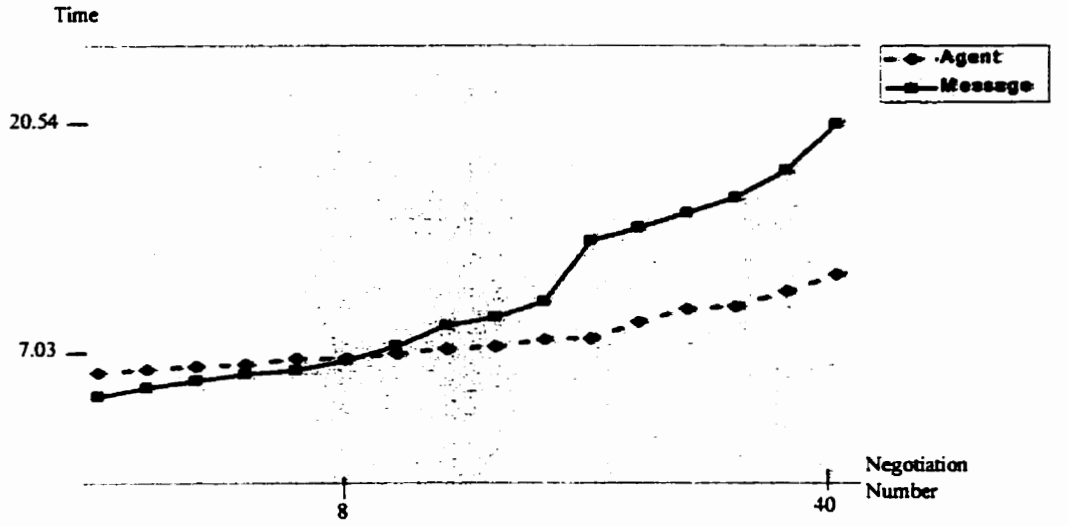


Figure 28: Time Cost In Model 2
For Three Switches Connection

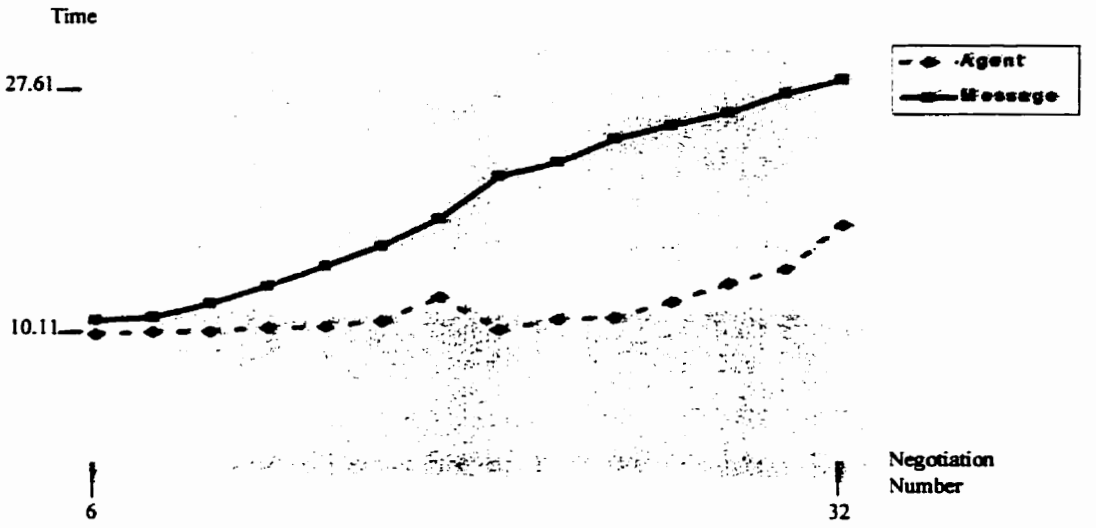


Figure 29: Time Cost In Model 1
For Five Switches Connection

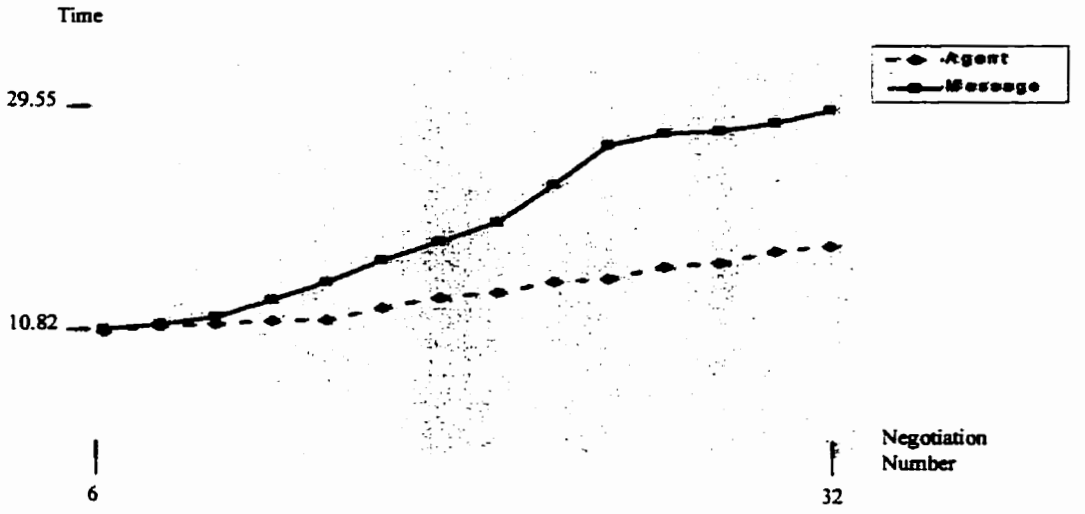


Figure 30: Time Cost In Model 2
For Five Switches Connection

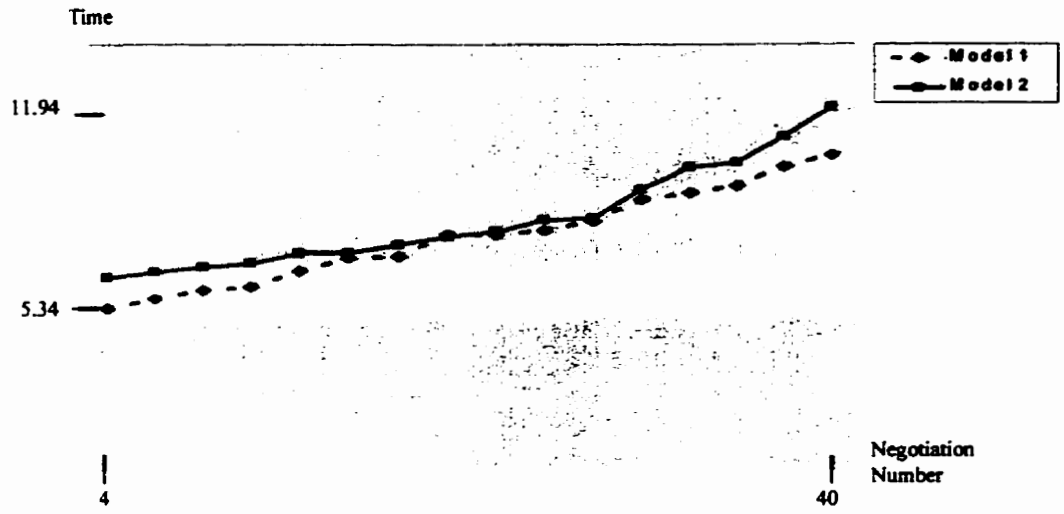


Figure 31: Agent Time Cost For Two Models In Three Switches Connection

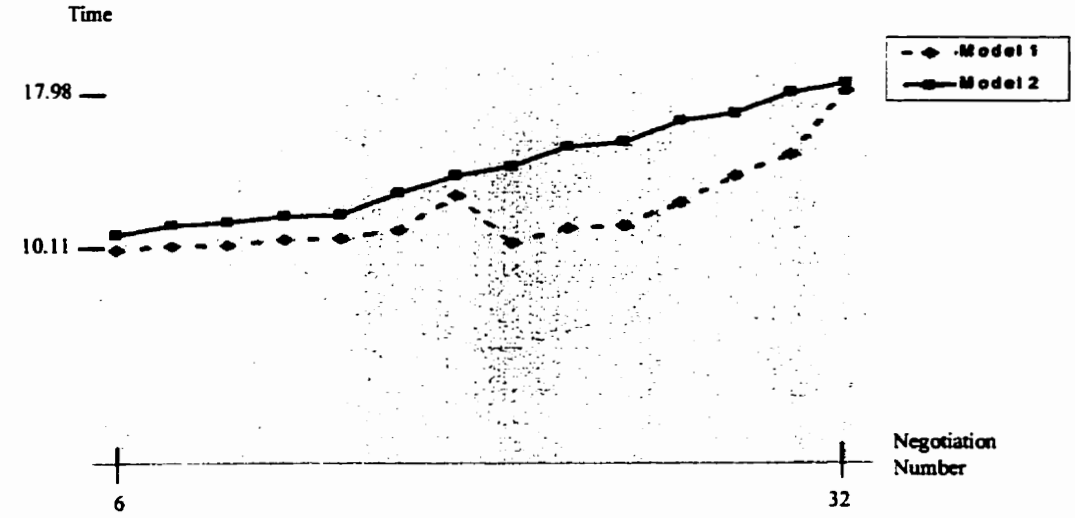


Figure 32: Agent Time Cost For Two Models In Five Switches Connection

CHAPTER 6 CONCLUSIONS

6.1 Summary

In this thesis, I investigated the application of mobile agents in network management and identified their advantages and disadvantages. The focus was on the two fields: network element monitoring through SNMP and network signaling by mobile agents. In these two research fields, the mobile agent shows strong capability and advantages. The implementation results also show that the mobile agent approach is a better approach than traditional message-based methods.

In network element monitoring, a system was developed to investigate the mobile agent's mobility, delegation, communication, persistence and fault tolerant abilities on heterogeneous platforms. The work combined mobile agents, SNMP, Java and web browser technologies. The system developed is a test-bed for the ideas rather than a complete solution for management systems. The operation of the system demonstrated an agent's strong variety of capabilities in monitoring network elements.

In network signaling, two resource negotiation models by mobile agents were implemented in a simulated network environment. Model 1 was found to be suitable for global optimization because of its central negotiation, although its architecture and implementation is more complex than model 2. Model 2 is simple in realization and costs less in terms of traffic than model 1, but it is not suitable for global optimization in resources negotiation. Before implementing the system, I thought model 2 should be better than model 1 in terms of time cost because of its simple architecture, but the result

is contrary. I also implemented the simulated network environment for testing the system. Resource negotiation uses concepts of economic consumer surplus and burstiness curves. At the same time, two message-based systems were designed to compare with their corresponding mobile agent models. The test results show clearly that the mobile agent method is better than a corresponding message-based method in terms of time cost under normal situations. Based on a simulation study, model 1 was found to be better than model 2. The advantage of a mobile agent technique is not only that it has a lower time cost, but also it is more flexible. Mobile agent methods can complete more complicated work than message-based methods because of their inherently stronger distributed nature.

6.2 Contributions

This thesis makes the following contributions to network management:

1. An NMS system was developed combining mobile agents, SNMP and web browser technology. This system is a test-bed of NMS applications using mobile agents. The system demonstrated strong capabilities in monitoring network elements as a result of its distributed management, fault tolerant, platform independent, agent mobility and persistence.

2. Realizing two simulation models of network signaling by mobile agents. These two models implement the resource negotiation by mobile agents using economic consumer surplus and burstiness curve algorithms.

3. Through the test results of network signaling by mobile agents, it is demonstrated that the agent signaling method is faster and more flexible than the message-based method when the call establishment needs more iterative negotiations.

4. Investigating the notion of price in resource negotiation. If there is no price constraint, a user will always ask best QoS service, however, if prices are charged to users, then they have to weigh the benefit of service against the cost charged.

5. Exploring the task delegation by mobile agents. In the mobile agent models of network signaling, the call user, the network service provider and agents can delegate their tasks to agents and vest them with authority to act on their behalf.

6. Investigating the construction of simulated network environment by mobile agents. Because mobile agents can run on behalf of the user and are easy to update on a remote host. The mobile agent method is an ideal method to simulate a network platform. By sending specific agents to remote sites on behalf of an element, the simulated system could be set up easily.

6.3 Future Work

Because of limited time, further work can not be investigated or implemented. This work helps establish the considerable potential to use mobile agents in network management. Some relating to this thesis are as follows. They could be used as a basis for future research.

- In network element monitoring, the mobile agent can supply more functions. For example, the agent can detect problems and fix some of the simple ones or could bring some useful information back for operator analysis. This work may employ techniques from the artificial intelligence field.

- In the network signaling, this work can be extended to multi-path selection. Mobile agents can do many things in this situation. For example, the calling agent can negotiate with all switch agents along these paths and select best the QoS path or chose the cheapest route according to the owner's instruction.

- Mobile agents can also be used in point to multi-point or multicast connections. In this case, the signaling system can take advantage of mobile agents operating in parallel. It is estimated that the mobile agent method will have much stronger capabilities than a corresponding message-based method.

- In network signaling, I do not consider the benefit of the switch agent or its owner: the network service provider. The switch agent only negotiates with a calling agent according to its cost function and negotiation strategy parameters determined by the owner. In actuality the switch agent should have its own negotiation purpose. For example, the switch agent can use price as a control parameter to adjust the traffic balance of the network or network resource usage. This deserves further investigation.

- In this thesis, I do not consider security problems. If this factor is considered, the negotiation meeting place should be re-considered, because all agents will contain their private data. How to encrypt the private data and how to select a safe and efficient central place for negotiation should be taken into account.

- A better negotiation algorithm should be investigated. The current negotiation algorithm is quite straightforward, but is seems to be simpler than the real world. The signaling system should supply much stronger services. For example, when there is not enough resource on the node, the switch agent can provide several proposals for the future resource reservation or give a better price for the future spare time.

- The application of mobile agent on network simulation is another big project. To my knowledge, by using mobile agents, the network simulation will become more flexible. Also the network simulation environment can take full advantage of mobile agent. Mobile agents seem to be an ideal method for this field because of their mobility, delegation ability and distributed properties.

- The traffic amount should be investigated on two mobile agent models and message-based approaches. It is not clear which method costs more in terms of traffic. Normally, the mobile agent should require more in terms of network traffic than messages, because the mobile agent need to transfer its code to a remote site. But when the communication messages of message-based methods reach certain value,

the total traffic of messages will exceed the total traffic of mobile agents because of the local negotiation capabilities of mobile agents.

REFERENCES

[Advent SNMP Package] Advent Network Management Java Development Package http://www.adventnet.com

[Pras 95] A. Pras, Network Management Architectures, Ph.D Thesis, The University of Twente, 1995

[Baldi 97] M. Baldi, S. Gai, G. P. Picco, Exploiting code mobility in decentralized and flexible network management, In Proceedings of the First International Workshop on Mobile Agents, Berlin, Germany, April 1997

[Baumann 97] J. Baumann, F. Hohl, N. Radouniklis, K. Rothermel, M. Straßer, Communication Concepts for Mobile Agent Systems, Proc. 1st Int. Workshop on Mobile Agents MA97, Springer Verlag, 1997

[Birman 94] K. P. Birman, Reliable Distributed Computing with the ISIS Toolkit, IEEE Computer Society Press, 1994

[Black 97] D. Black and C. Kahn, Mobile Agents and Network Survivability Information, Survivability Workshop, San Diego, CA February 12-13, 1997

[Caughey 95] S. J. Caughey, S. K. Shrivastava, Architectural Support for Mobile Objects in Large Scale Distirbuted Systems, Proc. 4th Int. Workshop on Object Orientation in Operating Systems IEEE Lund, Sweden August 1995

[CCITT I.350] CCITT Rec. I.350, General Aspects of Quality of Service and Network Performance in Digital Networks, International Telecommunication Union, Geneva, 89

[Cruz 91] R. L. Cruz, A calculus for network delay, part I: network elements in isolation, IEEE Transactions on Information Theory, 37(1), January 1991, pp.114-141

[DAI Project] Distributed Artificial Intelligence Research Unit (DAI): Intelligent System Group in the Department of Electronic Engineering at Queen Mary and Westfield College University of London, http://www.elec.qmw.ac.uk/dai/

[Dickman 94] P. Dickman, The Bellerophon Project: A Scalable Object-Support Architecture Suitable for a Large OODBMS?, Distributed Object Management, Morgan Kaufmann Publishers, Inc. San Mateo, California 1994, pp.287 - 299

[Dollimore 94] J. Dollimore, Fine Grained Object Migration, Distributed Object Management, Morgan Kaufmann Publishers, Inc. San Mateo, California 1994, pp.182 - 186

[Dorigo 96] M. Dorigo, A. Colorni, The Ant System: Optimization by a colony of cooperating agents, IEE Transactions on Systems, Man and Cybernetics, Vol. 26, No. 2, 1996

[Douglas 95] W. S. Douglas, Network Management 'What it is and what it isn't', Apr. 1995

http://www.sce.carleton.ca/netmanage/NetMngmnt/NetMngmnt.html

[FOKUS Project] FOKUS, GMD - Research Institute for Open Communication Systems, Germany's National Research Center for Information Technology, http://www.fokus.gmd.de/

[Gibney 97] M. A. Gibney, N. R. Jennings, Market Based Multi-Agent Systems for ATM Network Management, Proc. 4th Communications Networks Symposium, Manchester, UK 1997

[Hafid 97] A. Hafid and G. V. Bochmann, An Approach to QoS Management in Distributed MM Applications: Design and an Implementation, Multimedia Tools and Applications Journal, 1997

[Hafid 98] A. Hafid, G.V.Bochmann, R.Dssouli, Quality of Service Negotiation with Present and Future Reservations: A detailed Study, Computer Networks and ISDN Systems, volume 30, issue 8, 1998

[IBM Agent] "IBM Agents Specification Draft" http://www.trl.ibm.co.jp/aglets/documentation.html

[Ingham 96] D. B. Ingham, S. J. Caughey, Little Fixing the 'Broken-link' Problem: The W3Objects Approach, Computer Networks and ISDN Systems, Volume 28, issues 7–11, Proceedings of the Fifth International World-Wide Web Conference, Paris, France, 6-10 May 1996, pp.1255-1268

[Joseph 97] J. Kiniry and D. Zimmerman, Special Feature: A Hands-On Look at Java Mobile Agents, IEEE Internet Computing July/August 1997 http://computer.org/internet/ic1997/w4toc.htm

[Jul 88] E. Jul, H. Levy, Fine-Grained Mobility in the Emerald System, ACM Transactions on Computer Systems, Vol. 6, No. 1, February 1988, pp109 - 133

[Liskov 94] B. Liskov, Distributed Object Management in Thor, Distributed Object Management, Morgan Kaufmann Publishers, Inc. San Mateo, California 1994, pp79-91

[Low 91] S. Low and P. Varaiya, A simple theory of traffic and resource allocation in ATM, Proc. Globecom'91, December 1991, pp. 1633-1637

[Low 93] S. Low and P. Varaiya, Burstiness bounds for some burst reducing servers, Proc. Infocom'93, March 1993, pp. 2-9

[Machiraju 97] V. Machiraju, A Framework for Migrating Objects in Distributed Graphics Applications, M.Sc. Thesis, Computer Science Dept, The University of Utah, June 1997

[Milojicic 96] D. Milojicic, Mobile Agents Implementation Plan, Internal TOG RI Document 1996

[Moons 92] H. Moons, P. Verbaeten, Persistency Support for Mobile Objects in the COMET Heterogeneous Environment, Proceedings of the 1992 International Workshop on Object-Orientation in Operating Systems, Dourdan, France, September 24-25, 1992, pp.38-48

[ObjectSpace 1997] ObjectSpace Voyager Technical Overview Sept. 1997 http://www.objectspace.com/voyager/technical_white_papers.html

[Ozsu 94] M. T. Ozsu, An Introduction to Distributed Object Management, Distributed Object Management, Morgan Kaufmann Publishers, Inc. San Mateo, California 1994

[Perpetuum Project] Perpetuum Mobile Procura Agent Project, Network Management and Artificial Intelligence Laboratory, Department of Systems and Computer Engineering, Carleton University, http://www.sce.carleton.ca/netmanage/

[Plainfosse 95] D. Plainfossé, M. Shapiro, A Survey of Distributed Garbage Collection Techniques, International Workshop on Memory Management, Kinross, Scotland (UK), September 1995

[Ranganathan 97] M. Ranganathan, A. Acharya, S. Sharma, J. Saltz, Network-aware Mobile Programs, USENIX'97 1997

[Ranganathan 96] M. Ranganathan, A. Acharya, J. Saltz, Distributed Resource Monitors for Mobile Objects, IWOOOS'96

[Sahai 97] A. Sahai, C. Morin, S. Billiart, Intelligent agents for a Mobile Network Manager (MNM), Proceedings of the IFIP/IEEE International Conference on Intelligent Networks and Intelligence in Networks (2IN'97), September 1997, Paris, France.

[Sahai 98-1] A. Sahai, C. Morin, Mobile Agents for Enabling Mobile User Aware Applications, Proceedings of the Second International Confernce ACM Autonomous Agents (Agents 98), May 1998, Minneapolis/St.Paul, USA.

[Sahai 98-2] A. Sahai, C. Morin, Towards Distributed and Dynamic Network Management, Proceedings of the IEEE/IFIP Network Operation and Management Symposium (NOMS) February 1998, New Orleans, Lousiana, USA.

[Shaw 97] S. Green, L. Hurst, B. Nangle, P. Cunningham, Software Agents: A Review, Trinity College Dublin May 27, 1997

[Schoonderwoerd 96] R. Schoonderwoerd, O. Holland, J. Bruten, An Ant-Inspired Algorithm for Load Balancing in Telecommunication Networks, HP Labs Technical Reports 1996

[Schoonderwoerd 97] R. Schoonderwoerd, Collective Intelligence for Network Control, M.Sc. thesis on stigmergic control.1997

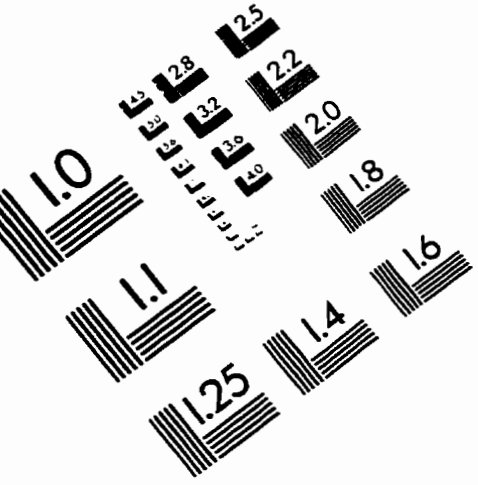
[Scott 95] S. Jordan, H. Jiang, Connection Establishment in High Speed Networks. IEEE Journal on Selected Areas in Communications, vol.13 no. 7, September 1995.

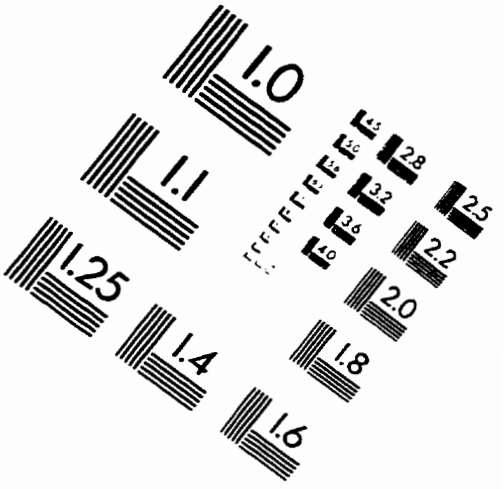
[Voyager Package] ObjectSpace Distributed Java Development Package http://www.objectspace.com

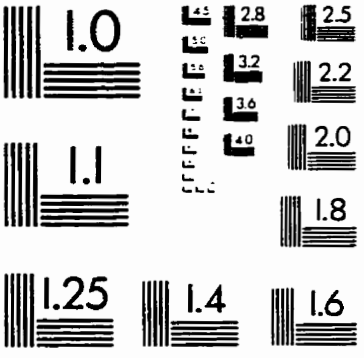
[White 1997] T. White, Routing with Swarm Intelligence Technical Report SCE-97-15. Systems and Computer Engineering, Carleton University, September 1997.

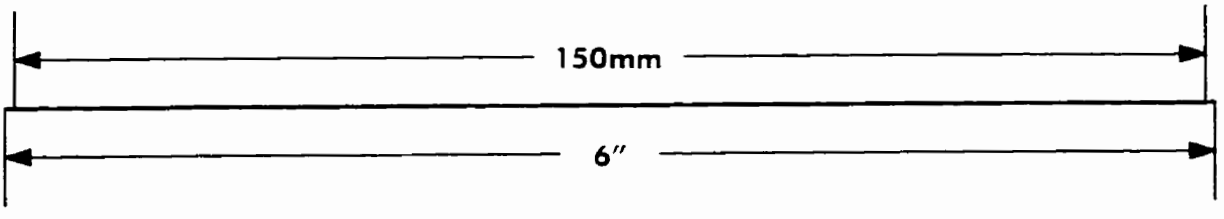
[White 1998] T. White, B. Pagurek, F. Oppacher, Connection Management by Ants: An Application of Mobile Agents in Network Management, Submitted to an International Conference on Evolutionary Computation (ICEC '98).

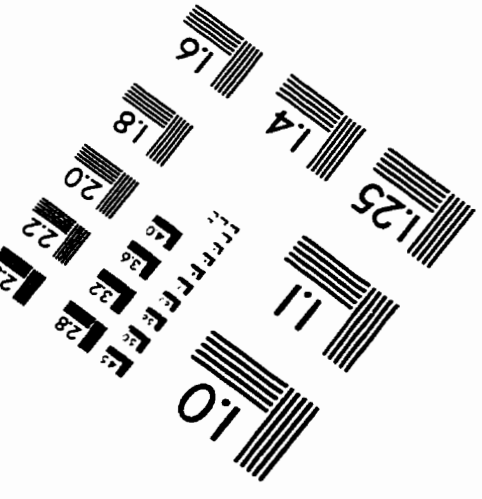
IMAGE EVALUATION
TEST TARGET (QA-3)

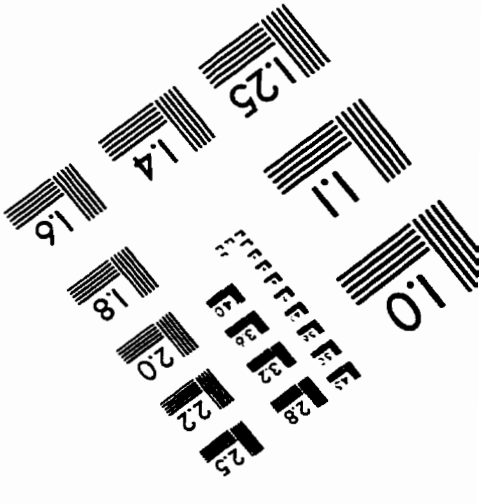












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989