Network Performance Management Using Mobile Software Agents

Christos Bohoris

Submitted for the Degree of
Doctor of Philosophy
from the
University of Surrey

Centre for Communication Systems Research
School of Electronics and Physical Sciences
University of Surrey
Guildford, Surrey GU2 7XH, UK

June 2003

Summary

In recent years a lot of promise has surrounded the potential impact of mobile software agents in the area of network management. The work aims to present a clear direction of practical exploitation of mobile agents for network management tasks. Three different case studies of network performance management were examined in order to evaluate the effectiveness of the agent mobility strategy and autonomous behaviour applied within the specific context. This work importantly allowed the identification of 'Constrained' mobility, an agent migration strategy especially suited for network management tasks, involving a mobile agent autonomously migrating to a single network element where its execution is confined. The mobile agent benefits identified relate primarily to the easy support for programmability of network elements and the autonomous, self-configurable agent operation. An important drawback is that the advanced capabilities of modern mobile agent frameworks typically incur significant performance overheads and these were confirmed through a detailed performance evaluation comparing mobile agents to distributed object and mobile code approaches. In the direction of addressing this drawback, the work proposes network management solutions based on specially formulated execution environments that retain important mobile agent benefits while reducing network performance overheads.

Key words: Mobile Agents, Performance Management, Distributed Objects, Software Mobility, Software Autonomy, Mobile Code

Email: c.bohoris@eim.surrey.ac.uk

WWW: http://www.ee.surrey.ac.uk/CCSR/Networks/

Dedicated to my grandfather George Petris who passed away on the 22 of March 2003.

Acknowledgments

I would like to thank my supervisor George Pavlou for recognizing my potential and providing me this great opportunity to do research. The knowledge and advice he passed on to me during the last 5 years will be an invaluable asset as I move on to my future career steps.

Also, I would like to express my gratitude to my second supervisor Antonio Liotta for his guidance, support and constructive criticism. In addition, his encouragement and enthusiasm have been an important driving force for me to overcome the difficulties encountered during my research explorations.

Many thanks to a number of colleagues in the Networks Group of CCSR: Stelios Gouveris, Alvin Yew, Haitham Cruickshank, Ilias Andrikopoulos, Paris Flegkas, Panos Trimintzios, Sunil Iyengar, Lloyd Wood, Siva Sivavakeesar, Shiping Fang, Enric Jaen, Ning Wang and Stelios Georgoulas. It has been a genuine pleasure to work with them.

I would like to acknowledge the contribution and support of my friends: Kostas Sfyrakis, Panos Alexandropoulos, Kyriakos Georgiou, and Vasilis Volonakis. Their friendship has been invaluable to me and has brought important balance and joy in my life.

Finally, I would like to thank my parents Zaharoula and George Bohoris, my brother George and my grandparents Kalliopi and George Petris for caring, supporting and always being there for me throughout my many years of study.

Contents

List of Figures

Abbreviations

3GPP	3rd Generation Partnership Project
API	Application Programming Interface
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
COD	Code on Demand
CORBA	Common Object Request Broker
CS	Client-Server
DiffServ	Differentiated Services
FIPA	Foundation of Intelligent Physical Agents
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organisation for Standardization
ITU-T	International Telecommunication Union (-Telecommunications sector)
LAN	Local Area Network
MA	Mobile Agent
MbD	Management by Delegation
MIB	Management Information Base
MO	Managed Object
NE	Network Element
NP	Network Provider
ODP	Open Distributed Processing
OMG	Object Management group
OSA	Open service Architecture
OSI	Open System Interconnection

QoS	Quality of Service
REV	Remote Evaluation
RPC	Remote Procedure Call
Java-RMI	Java Remote Method Invocation
JVM	Java Virtual Machine
RMON	Remote Monitoring
SLA	Service Level Agreement
SLS	Service Level Specification
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
TINA	Telecommunications Information Networking Architecture
TMN	Telecommunications Management Network
UMTS	Universal Mobile Telecommunications System
VHE	Virtual Home Environment
VPN	Virtual Private Network

Thesis Related Publications

Journal Papers

[Boho03] **C. Bohoris**, G. Pavlou, A. Liotta, *"Mobile Agent-based Performance Management for the Virtual Home Environment"*, Journal of Network and System Management (JNSM), vol. 11, no. 2, pp. 133-149, Plenum Publishing, June 2003.

Conference Papers

[Liot02b] A. Liotta, A. Yew, **C. Bohoris**, G. Pavlou, *"Delivering Service Adaptation with 3G Technology"*, In the Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM '02), Montreal, Canada, Lecture Notes in Computer Science, vol. 2506, pp.108-120, Springer, October 2002.

[Boho02] **C. Bohoris**, A. Liotta, G. Pavlou, *"A Hybrid approach to Network Performance Monitoring based on Mobile Agents and CORBA"*, In the Proceedings of the 4th International Workshop on Mobile Agents for Telecommunication Applications (MATA '02), Barcelona, Spain, A. Karmouch, T. Magedanz, J. Delgado, eds., pp. 151-162, Springer, October 2002.

[Yew01] A. Yew, **C. Bohoris**, A. Liotta, G. Pavlou, *"Quality of Service Management for the Virtual Home Environment"*, Proceedings of the 12th IEEE/IFIP International Workshop on Distributed Systems: Operations and Management (DSOM '01), Nancy, France, O. Festor, A. Pras, eds., pp. 179-190, INRIA Press, October 2001.

[Boho00c] **C. Bohoris**, A. Liotta, G. Pavlou, *"Evaluation of Constrained Mobility for Programmability in Network Management"*, In Services Management in Intelligent Networks, Proceedings of the 11th IEEE/IFIP International Workshop on Distributed Systems: Operations and Management (DSOM '00), Austin, Texas, USA, A. Ambler, S. Calo, G. Kar, eds., pp. 243-257, Springer, December 2000.

[Boho00b] **C. Bohoris**, A. Liotta, G. Pavlou, *"Software Agent Constrained Mobility for Network Performance Monitoring"*, Proceedings of the 6th IFIP Conference on Intelligence in Networks (SmartNet '00), Vienna, Austria, ed. H.R. van As, pp. 367-387, Kluwer, September 2000.

[Boho00a] **C. Bohoris**, G. Pavlou, H. Cruickshank, *"Using Mobile Agents for Network Performance Management"*, Proceedings of the IFIP/IEEE Network Operations and Management Symposium (NOMS '00), Hawaii, USA, J. Hong, R. Weihmayer, eds., pp. 637-652, IEEE, April 2000.

Book Contributions

[Gali02] A. Galis, S. Covaci (eds.), *"Mobile Intelligent Agents Applied to Communication Management Systems"*, CRC Press, ISBN: 0849392012, January 2002.

Other Papers

[Liot02c] A. Liotta, A. Yew, **C. Bohoris**, G. Pavlou, *"Supporting Adaptation-aware Services through the Virtual Home Environment"*, Proceedings of the Hewlett-Packard Openview University Association Plenary Workshop 2002 (HP-OVUA), June 11-13, 2002.

[Yew01b] A. Yew, A.Liotta, **C. Bohoris**, G. Pavlou, *"Towards Quality of Service Aware Services for the Virtual Home Environment"*, Proceedings of the Hewlett-Packard Openview University Association Plenary Workshop 2001 (HP-OVUA), Berlin, Germany, June 24-27, 2001.

[Pavl00] G. Pavlou, A. Liotta, **C. Bohoris**, D. Griffin, P. Georgatsos, *"Providing Customisable Remote Management Services Using Mobile Agents"*, Proceedings of the Hewlett-Packard Openview University Association Plenary Workshop 2000 (HP-OVUA), Santorini, Greece, June 12-14, 2000.

Thesis Related Projects

[MIAMI] Mobile Intelligent Agents for Managing the Information Infrastructure (MIAMI) project (1998-2000), ACTS 338,

http://www.cordis.lu/infowin/acts/rus/projects/ac338.htm

[MANT] Management Testing & Reconfiguration of IP based networks using mobile software agents (MANTRIP) project (2000-2002), IST-1999-10921,

http://www.solinet-research.com/mantrip/

[VESPER] Virtual Home Environment for Service Personalization and Roaming Users (VESPER) project (2000-2002), IST-1999-10825,

http://www.ee.surrey.ac.uk/CCSR/IST/Vesper/

PART I - THESIS BACKGROUND

Chapter 1

1 Introduction

1.1 Thesis Overview

The success of the Internet during the 90's along with the tremendous popularity of hand-held, mobile communication devices have crucially strengthened the position of the telecommunications industry. Today, this industry is merging the building blocks required to support advanced network applications and services. Network management capabilities constitute an important building block, aiming to ensure the efficient operation of a network infrastructure. More specifically, network performance management capabilities play a central role in this direction, mainly through mechanisms for performance monitoring, threshold checking and configuration of network Quality of Service (QoS) guarantees. In the modern telecommunications environment where change is the only constant, network management has to effectively address a number of requirements for programmable and dynamic operation.

This situation has highlighted a number of limitations of protocol-based network management as exemplified by the widely used Simple Network Management Protocol (SNMP) [Case90]. The approach has been heavily criticized by the research community for its centralized nature, static management capabilities and simple-minded model. As an alternative, approaches based on distributed object frameworks were proposed, allowing for decentralized and scalable network management systems. Early experience with distributed object frameworks as the basis for Telecommunications Management Network (TMN) systems highlighted the main disadvantage of the approach [Pavl96]. Distributed object frameworks lack the required support that would allow programmable network management capabilities. This has led many researchers to study software mobility and mobile agent technology as a means of achieving programmability. While some initial, mostly theoretical work has been done in this direction, progress has been hindered by a lack of standards, inconsistent terminology and limited experience in programmable network management. Given these motivations, the thesis examines the practical usefulness of mobile agents in the context of network performance management.

As a first step, the theoretical background on mobile agents was reviewed and enhanced, in order to lay a basis of consistent terminology, typical mobile agent behaviour and potential usefulness for network management. A significant result of this study was the identification of 'Constrained' mobility, an agent migration strategy that involves a mobile agent autonomously migrating to a single network element where its execution is confined. Constrained mobility is an evolution of an earlier model for mobile code typically referred as Remote Evaluation (REV) [Stam90]. An important aspect of Constrained mobility is that a mobile entity is not restrained to be a remote service, as in the case of REV, but instead acts as an autonomous software agent (e.g. choosing its migration node, intelligently collaborating with other agents to achieve its task, etc.). Constrained mobility allows the easy programmability of network management capabilities through a process of removal of an agent executing at a network element and migration of a replacement agent carrying updated management logic. This logic can be statically placed in an updated mobile agent or it can be dynamically and autonomously loaded by a suitable mobile agent during initialisation.

A practical evaluation and assessment of the usefulness of mobile agents was performed in the context of three case studies involving modern telecommunications applications relying on advanced performance management capabilities. The case studies were, "ATM-based Active VPN management", "QoS Configuration and SLA Auditing of IP-DiffServ Networks" and "QoS Management for the Virtual Home Environment". Constrained mobility was found to be a particularly suitable strategy for network management systems that involve long-term tasks for which programmability is required. The strategy was chosen as the most suitable for all the three case studies of performance management considered. Through a number of measurements of system performance, the overheads incurred by mobile agent-based systems were compared to distributed object and mobile code systems based on CORBA, Java-RMI and Script-MIB. The additional capabilities offered by mobile agent frameworks imply an increase in performance overheads and this was confirmed by the measurement results. Given this weakness of mobile agent approaches, the work of the thesis was directed towards solutions that retain key mobile agent capabilities while achieving significant performance improvements.

An important issue highlighted through the case studies was that modern mobile agent frameworks offer a lot of generic, agent supporting functionality that was inherited by the management systems developed. Most parts of this functionality were typically not required but nevertheless contributed to an increase in performance overheads. Given this motivation, the thesis presents a novel, lightweight platform for Constrained mobility (named, the 'CodeShell') that offers only the minimum supporting functionality required by a typical network management system. This reflects on a mechanism allowing Constrained mobility combined with the flexibility to introduce autonomous behaviour.

Another important aspect revealed through the work on the case studies is that mobile agents and distributed objects, although once thought as rivals, should ideally co-exist in a system that combines the best of both. Mobile agent frameworks allow the development of management systems that support programmability and autonomous, fine-tuned operation while distributed object frameworks result in systems with low performance overheads. As such, real synergy could be achieved in a network management system if stationary entities are provided using static objects that collaborate with mobile agents in both directions. The thesis presents such a novel solution for network performance management that is based on a 'hybrid' approach combining mobile agents and CORBA objects.

A detailed set of performance measurements was taken on network performance management systems in the context of both the 'CodeShell' lightweight platform and the 'hybrid' approach. The measurement results confirm the performance improvements expected and further justify the value of the proposals.

1.2 Research Motivation

In recent years we have witnessed an evolution in network management approaches driven by the need of addressing the requirements of modern networks becoming increasingly large, sophisticated and complex. The initial protocol-based network management solutions, proposed in the early 90's as exemplified by the widely used Simple Network Management Protocol (SNMP) [Case90], are today particularly limiting due to their centralized and static nature. The approach is centralized as it relies on a limited set of capabilities at network nodes while management processing has to be performed at the network management station. Any capabilities at network nodes are fixed, embedded by the manufacturer at the network element construction time.

While a protocol-based approach is specific to a management framework, a generic approach to the client/server model can be achieved through the use of a distributed object framework as proposed by researchers in the mid 90's. These are exemplified by the Common Object Request Broker Architecture (CORBA) [OMG95] proposed by the Object Management Group (OMG) and Java-Remote Method Invocation (Java-RMI) [JRMI] proposed by Sun Microsystems. Distributed object frameworks, allow the creation of decentralized, static systems. Decentralization is achieved by placing required management logic in network nodes and by creating instances of management objects specific to interested clients. Despite this, distributed object frameworks still suffer from a lack of support for programmability, as the management logic located in network nodes is static and cannot be easily altered. The issue of programmability of managed nodes is particularly important for network management systems. The complexity and long standardization cycles associated with TMN systems crucially hindered their success [Pavl96]. Network managers

of such systems had to wait for several years before a standardization cycle was complete and the required management functionality was embedded in network nodes.

In order to address this problem, research turned into a mobile agents approach, representing an attractive solution for programmable systems that can potentially influence significantly the network management area. Back in 1998, emerging agent paradigms and enabling technologies were considered a key for the implementation of flexible and scalable solutions that add a degree of openness to the telecommunications industry. In retrospect, agent technology suffered from different terminology and heterogeneity of technical approaches due to the lack of standards. Furthermore the broad application of agent technologies to specific environments such as network management was still in its infancy. These circumstances triggered in 1997 a large initiative within the European Union involving 14 research projects (grouped as the 'Climate' cluster of projects) [Clim97] researching the applicability of Mobile Agents to telecommunications systems. My research on the role of mobile agents for network management started in 1998 while working within the 'Climate' cluster and specifically within the MIAMI project [MIAMI].

The motivation for this research work is also reflected as "future work" in related PhD theses submitted before or during the early stages of this thesis. A few indicative excerpts from thesis sections on "future work" are as following:

G. Vigna, February 1998 [Vign98b]: *"Certainly, the framework presented in this thesis needs to be incrementally enriched, revised, taking into account experiences, results, and innovations that will be progressively achieved. We need to improve our classification by better analysing the properties and weaknesses of the existing design paradigms. … Finally, we need to further explore the relatively unknown world of applications and problems that can benefit from the adoption of technologies and methodologies based on the notion of code mobility."*

G. Pavlou, March 1998 [Pavl98]: *"The advent of languages like Java that can support code mobility opens up new possibilities for management. The impact of this new paradigm to TMN is a whole new area of interesting research, pointing to "programmable" network elements with very simple native interfaces and an environment that can host mobile software entities."*

T. Papaioannou, February 2000 [Papa00b]: *"We believe the next stage of validation for our philosophical argument would be to undertake a course of research to directly compare Mobile Agents with RM-ODP* (Reference Model of Open Distributed Processing)*."*

A. Liotta, July 2001 [Liot01b]: *"One necessary step towards the realization of active distributed monitoring is its implementation and experimentation on prototype networks or on a real networked system. Real measurements will enlighten the actual behaviour of the proposed approach with regard to overheads, stability and complexity. Measurements on the actual agent*

development time as a function of various scale factors will strengthen even further the motivation for integrating agents in monitoring systems”.

1.3 Objectives

A lot of promise has surrounded the potential impact of mobile agents to the area of network management. The thesis aims to demystify their practical usefulness for network performance management. In this direction the main objectives are:

1. The identification of agent migration strategies for network management tasks, presenting a clear direction of practical exploitation.
2. The evaluation of mobile agents as compared to alternative modern approaches based on static distributed objects.
3. The assessment of the usefulness of mobile agents within the context of case studies for performance management.
4. The proposal of efficient mobile agent-based solutions for network management.

1.4 Thesis Statement

The statement of the thesis can be formulated as follows:

Software mobile agents can provide an easy means for programmability of managed nodes mainly by following a 'Constrained' migration strategy to a single destination node. This can influence significantly the capabilities of a network performance management system operating within the context of currently envisioned network applications. Programmability can be effectively retained in agent systems that are based on execution environments especially formulated for improved network performance.

1.5 Thesis Roadmap

The thesis is composed of two parts. The first involves the current introduction along with the thesis background covered by chapters 2 and 3.

Chapter 2 describes the background topics associated with this interdisciplinary thesis. These include theoretical aspects of mobile agents, existing ideas on software mobility, as well as approaches for network management and mechanisms specific to performance management.

Chapter 3 complements the previous chapter by focusing on the review of work related to the thesis. The state of the art in approaches for code mobility is presented along with the current

network management situation and the emergence of mobile agents in the area. These descriptions further help to put this research work into context and decouple previous research efforts from the additional value of the thesis.

The second thesis part involves the thesis results covered by chapters 4 to 8.

Chapter 4 presents the author's theoretical study and contributions on agent mobility strategies explaining their usefulness for performance management and their role in the wider network management context. A detailed description of the approach and usefulness of the proposed 'Constrained' agent migration strategy aims to highlight the differences and added value compared to similar models for mobile code. Descriptions of additional agent migration strategies present the enhancements over previous proposals, placing them in a specific context of practical exploitation by network management systems. The study of mobility is complemented by a study on autonomous behaviour that highlights the capabilities of mobile agents further from mobile objects and code and in the direction of providing 'intelligent', self-configurable and fault-tolerant network management systems.

Chapter 5 builds upon the basis of the previous chapter, with descriptions of three case studies that allowed a detailed practical assessment of the usefulness of mobile agents. The three case studies presented are "ATM-based Active VPN management", "QoS Configuration and SLA Auditing of IP-DiffServ Networks" and "QoS Management for the Virtual Home Environment", examined by the author while working within the ACTS-MIAMI [MIAMI], IST-MANTRIP [MANT] and IST-VESPER [VESPER] projects respectively. For each, we describe the role of performance management within the specific environment, the mobile agent approach chosen along with the benefits and drawbacks of the use of mobile agents in the system. The author designed and developed suitable mobile agent-based performance management capabilities for each of the case studies and assessed them in detail through a number of experiments and measurements, highlighting the performance issues of mobile agents when compared to distributed objects and mobile code approaches.

Chapters 6 and 7 are motivated by the significant performance overheads identified through the practical case studies and related experiments as the primary concern for mobile agent deployment. As such, in each chapter the author proposes an alternative solution that preserves key benefits of agent mobility and autonomy but also results in an important improvement in network performance overheads.

Finally, Chapter 8 concludes the thesis, summarizing the main results and contributions. It discusses the work from a global viewpoint, indicates the extent at which the objectives were addressed and finally, draws the conclusions and points to future research developments stemming from this thesis.

Chapter 2

2 Background

This chapter provides an overview of the various topics involved in this interdisciplinary thesis. These include theoretical aspects of mobile agents, existing ideas on software mobility, as well as approaches for network management and mechanisms specific to performance management. We should note here that the purpose of this chapter is to introduce the main concepts and issues related to the thesis and identify some of the gaps that motivate the need for further work. A detailed and extensive review of the topics covered is already present in the literature and is, hence, beyond the scope of this thesis.

2.1 Software Agents

2.1.1 Definition and Characteristics

Software agents have been introduced in the early 90's within the artificial intelligence research community, as semi-intelligent computer programs that assist a user with large amounts of complex information within a network environment. One of the pioneering research groups in the area was the MIT media lab headed by Professor Pattie Maes that first built successful prototypes of agents for tasks like personalized information filtering, user assistance, and online trading. During the next years until the mid 90's, research in the field applied a large number of well-established artificial intelligent techniques (neural networks, fuzzy logic, genetic algorithms, etc) into software agents [Chai92]. The sheer number and diversity of these approaches resulted in software agent research loosing its focus and identity. It was time to step back and re-consider what has been done and attempt to place some strong foundations. In this direction we have seen a number of research efforts such as in [Smit94][Oren94][Maes95][Wool95b][Haye95], all attempting to narrow down a definition of a software agent and its characteristics. A notable work by [Fran96] combined this theoretical work and has formalized a popular definition as follows: "An *autonomous agent* is a software entity situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own goals and so as to effect what it senses in the future". The term "software agent" has been widely adopted to reflect on the

above definition and also incorporate the two most commonly described agent types that researchers have used to tackle complex problems, namely Mobile Agents (MA) and collaborative Intelligent Agents (IA). Software agents that may be dispatched from one node in a network and transported to a remote node for execution are commonly referred to as mobile agents. Intelligent agents are typically depended on intense collaboration with other agents often in a generic manner through an agent communication language.

The advantage of mobile agents as compared to static intelligent agents is that the former allow for the easy programmability of remote nodes by migrating and transferring functionality were required. On the other hand, static intelligent agents can collaborate to devise or negotiate flexible solutions for complex application scenarios. A disadvantage of mobile agents as compared to static intelligent agents is that they can often incur significant performance overheads in the network during migration.

Software agents are often more clearly understood through their attributes and behaviour. It is commonly agreed among researchers that every agent exhibits several (but rarely all) of the following characteristics [Wool95a]:

- Autonomous: Agents should operate without the intervention of external entities. They typically have control over their actions and internal state.
- Adaptable: Agents are characterised by their ability to set-up their own goals and strategy to achieve them. They typically acquire and process information on their environment both spatially and temporarily and use this information to influence their future behaviour.
- Goal oriented: Agents should exhibit goal-oriented behaviour such that their performed actions cause beneficial changes to the environment. In most cases an agent terminates after the completion of its goal.
- Communicative/collaborative: No agent has a complete picture of the overall system within which it operates. Each agent is an expert in a specific task and it has to collaborate with other agents in order to solve a given problem.
- Pro-active/Active: Often agents are required to anticipate future situations along with responding to changes within their environment.

Traditional A.I. techniques can easily result in large, processing intensive and complex programs that we often want to avoid in the network environment within which software agents reside. This is why initial research considered agents as semi-intelligent entities, exhibiting only a ‘weighted’ level of computational intelligence through a selection of required characteristics. From this viewpoint, agents have been classified [Wool95b] according to their degree of agency (i.e.

intelligence), as “strong” and “weak” agents. “Strong” agents are those which reason based on mental notions like belief, desire and intention. “Weak” agents (e.g. most mobile agents) do not rely on mental notions and they are more concerned with the distribution of their intelligence, adaptation issues or concurrent execution. The contributions presented throughout this thesis relate to weakly intelligent agents.

2.1.2 Potential Benefits

Software agent researchers have foreseen a number of significant potential benefits of agents. However, these are mostly based on the theoretical study of the properties of the agent and alternative paradigms, weakly supported by limited practical experience and assessment within specific application contexts. These benefits, identified and widely supported by a large number of researchers [Lang98] [Bies98b] [Papa00a] are described below:

- Reduction of network load.

  Distributed systems often rely on communication protocols involving multiple interactions to accomplish a given task. For network management operations this typically results in high levels of management traffic. Mobile agents allow users to package a conversation and dispatch it to a destination host where interactions take place locally. Mobile Agents are also useful when reducing the flow of raw data in the network. By raw data we mean basic information that usually needs to be processed in order to produce a useful meaning. When very large volumes of raw data are kept at remote hosts, that data should be processed in its locality rather than transferred over the network.

- Overcoming network latency.

  In critical real time systems there is a need to respond immediately to important events. Controlling the decisions of such a system remotely through the network can involve significant latencies. These can be unacceptable for some real-time systems or systems involved with on-line processing of information. Mobile agents offer a solution because they can be dispatched from a central controller to act locally and execute the controller’s directions directly.

- Easy encapsulation of protocols.

  When data is exchanged in a distributed system, each host owns the code that implements the protocols needed to properly code outgoing data and interpret incoming data. However, as protocols evolve to accommodate new requirements of efficiency or security, it is cumbersome if not impossible to upgrade protocol code properly. As a

result, protocols often become a legacy problem. Mobile agents, on the other hand, can move to remote hosts to establish "channels" based on proprietary protocols.

- Asynchronous and autonomous execution.

  Mobile devices often rely on expensive or fragile network connections. Tasks requiring a continuously open connection between a mobile device and a fixed network are probably not economically or technically feasible. To solve this problem, tasks can be embedded into mobile agents, which can then be dispatched into the network. After being dispatched, the agents become independent of the process that created them and can operate asynchronously and autonomously. The mobile device can reconnect at a later time to collect the agent.

- Dynamic adaptation.

  Mobile agents can sense their environment and react autonomously to changes. When a number of mobile agents are assigned to work with a common goal they can distribute themselves among hosts in the network to maintain the optimal configuration for solving the problem. In the case of a mobile agent moving across a number of host nodes, it can adapt its future behaviour according to information that it has already collected and stored in its state.

- Dealing with heterogeneous environments.

  Network computing is fundamentally heterogeneous, often from both hardware and software perspectives. Mobile agents are only dependent to their execution environment. Also, mobile agents are not coupled with other objects; instead they collaborate with other local agents and use the functionality they discover in each host.

- Network fault tolerance.

  In a scenario of fixed nodes connected to a network, by placing mobile agents to execute locally, their operation continues even if a network fault occurs that makes remote communication unavailable.

The work described in this thesis intents to demonstrate and assess the competence of mobile agents in providing such advantages within the context of network management. Although the potential benefits from the use of mobile agents are particularly appealing, the transition to agent-based systems was hindered by a number of serious obstacles detailed below.

2.1.3 Associated Issues

During the last decade agent technology was mostly used within the academic and research communities, with no impact to the telecommunications industry. Researchers have identified a number of problems and issues [Kotz99] [Hari95] [Papa00a] associated with software agents as described below:

- Standardization and Interoperability: Inconsistency has greatly hindered the adoption of mobile agent technology with limited standardization so far. Even the few standardization efforts made are yet to be widely adopted. In the direction of standards, the Object Management Group (OMG) has produced the Mobile Agent System Interoperability Facility (MASIF) [MASI98] that crucially addresses the issue of interoperability between mobile agent platforms. In addition, the Foundation for Intelligent Physical Agents (FIPA) has produced specifications for the "intelligent" communication between agents [FIPA01] [FIPA02].

- Security and Safety: Although it is now possible to deploy a mobile agent system that adequately protects a node against malicious agents [Vign98a], numerous challenges remain. These involve the protection of nodes without artificially limiting agent access rights, protecting an agent from malicious nodes as well as protecting groups of nodes that are not under single administrative control.

- Lack of killer application: The current network environment, its usage and related management tasks have not revealed so far an application that can be only achieved through the use of mobile agents.

- Limited practical experience: While a theoretical base for mobile agents exists there is limited work on the application and practical assessment of agent technologies to specific contexts such as network management. This is an important requirement for agent technology to reach maturity.

- Performance Overheads: Mobile agent-based systems can help reduce network latency and bandwidth utilization, but this often comes at the expense of higher utilization of resources at network nodes. Furthermore, attention is needed regarding any agent migration overheads especially in scenarios involving multi-hop mobility.

- Getting ahead of the evolutionary path: It was unlikely that the current centralized client/server approach to management would move directly to mobile agent-based approach. The evolutionary path takes time and it will probably move gracefully from centralized protocols, to distributed object frameworks, followed by mobile code solutions and later by mobile agents.

This thesis tackles mainly the last three issues described, with detailed practical work, evaluation and assessment on the usefulness and drawbacks of mobile agents within the network management context. Furthermore, the thesis presents proposals on 'lightweight' mobile agent solutions that improve performance overheads and smooth the transition into agents through the enhancement of emerging distributed object technologies.

2.1.4 Intelligent Agents

Intelligent agents typically rely on heavy collaboration. This is done in a generic manner using messages described in a special Agent Communication Language (ACL). A definition of the context within which a message has a useful meaning is provided by an agent ontology. The Foundation of Intelligent Physical Agents (FIPA) has recently standardized an ACL that enjoys growing support and is based on earlier results in the form of the Knowledge-based Query Mark-up Language (KQML) [KQML]. This FIPA-ACL [ACL98] has a precisely defined syntax that forms the basis of communication between independently designed and developed software agents.

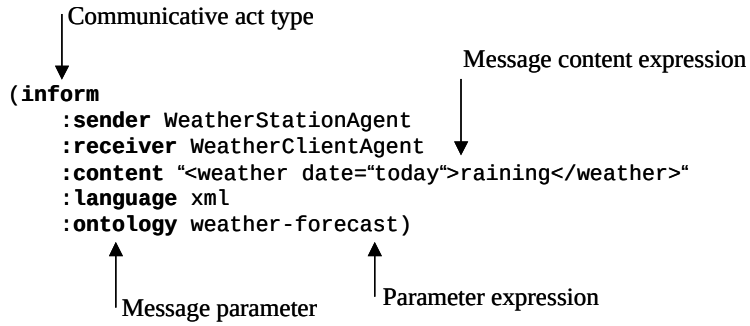


Figure 2-1: An example FIPA ACL message

As defined in the FIPA-ACL specification, an agent can be described as though it has mental attributes of:

- *Belief*, denoting the set of propositions which the agent accepts are currently true. Propositions which are believed false are represented by believing the negation of the proposition.
- *Uncertainty*, denoting the set of propositions which the agent accepts are not known to be certainly true or false, but which are held to be more likely to be true than false. Propositions which are uncertain but more likely to be false are presented by being uncertain of the negation of the proposition.
- *Intention*, denoting a choice, or property, or set of properties of the world which the agent desires to be true and which are not currently believed to be true. An agent which adopts

an intention will form a plan of action to bring about the state of the world indicated by its choice.

FIPA-ACL Messages are based on the speech act theory, which resulted from the linguistic analysis of the human communication. A key idea in the speech act theory is that producing language is an action. The action is performed by a speaker intending to change the mental state of the listener. The structure of a typical FIPA-ACL message can be seen in the figure (Figure 2-1).

An important advantage of deploying intelligent collaborative agents is the use of communication messages to decouple entities from interface dependencies that can be limiting and ineffective when we have complex data processing, involving dynamically changing data (e.g. When contracts are negotiated, etc).

2.1.5 Relationship to Active Networks

The concept of active networking emerged from discussions within the Defence Advanced Research Projects Agency (DARPA) research community between 1994 and 1995 on the future directions of networking systems. The problems identified included the current difficulty of integrating new technologies and standards into the shared network infrastructure as well as the difficulty of accommodating for new services. Several strategies, collectively referred to as “active networking”, emerged to address these problems. Commonly, Active Networks (AN) [Tenn96] allow their users to push customized programs in network nodes for execution. While mobile agents are one AN enabling technology, active networks research also considers a more extreme case in which packets are replaced by what is referred as “capsules” or “active packets”. These contain data and a program fragment that executes within every router or switch along its path. The networks are active in the sense that nodes can perform computations, which modify the packet’s content. Suitable applications for this paradigm include packet filtering, congestion control and service-specific routing [Tenn97].

While the use of Mobile agents in networking has generally been restricted to the management plane, ANs mainly focus on packet processing in the data path, in the user plane and sometimes the control plane. Recently, AN concepts have been applied to the management plane [Kawa00][Schw00][Raz00], resulting in an overlap with mobile agent approaches. The reason is that an active packet can be perceived as a simple mobile agent that executes on every node along its path through the network. Similarly, a migrating mobile agent can be perceived as an active packet containing data and a program. In this direction, Kawamura et al [Kawa00] introduced an active distributed management architecture characterised by a programmable middleware platform (organised in different layers of abstraction), whose active properties are drawn from the

AN and MA paradigms. Furthermore, Raz et al [Raz00] described a prototype system where legacy routers are enhanced with an 'active' engine, a user level execution environment adjunct to the standard forwarding mechanism. In this direction of active networking a number of suitable applications are proposed such as monitoring/control, bottleneck detection and topology detection.

2.1.6 Relationship to Open Signalling

In addition, to active networks a more modest approach to achieving network programmability was proposed by the Open Signalling (OPENSIG) research community. Their proposal is to provide open access to switches and routers by using a set of open, programmable interfaces that model the communication hardware. Network elements are abstracted as distributed objects with a clear distinction between transport, control and management aspects. By opening up the network elements service providers can manipulate the states of the network using distributed object frameworks to construct and manage new network services. As we can see an Active Networks or mobile agents approach is far more dynamic than OPENSIG network programming interfaces.

While, OPENSIG allows programmability, an update in functionality of a large number of network elements places a burden to a network administrator to manually install the required servers (e.g. based on CORBA) at each one. In contrast, an active networks approach based on mobile code or capsules would mean that the network administrator specifies the functionality modules required and these are delivered to the network nodes. In a further enhancement of capabilities a mobile agent-based approach to programmability, can act on behalf of the network administrator, autonomously selecting the appropriate functionality modules and delivering them within suitably equipped agents.

2.1.7 Mobile Agent Platforms

Mobile agent platforms are comprised of two main important building blocks, a naming service similar to that of CORBA as well as an agent execution environment. A naming service for mobile agents caters not only for migrating agents but also for the naming and lookup of the agent execution environments hosting those agents. Because of the dynamic nature of mobile agents, sophisticated mechanisms (e.g. component frameworks mechanisms such as reflection and introspection) are required for the binding and remote communication of agents. Such communication is typically supported by a proprietary Agent Communication and Transfer Protocol (ACTP). ACTPs are based on a number of protocols such as the Java Remote Method Protocol (JRMP), the Internet Inter-ORB Protocol (IIOP) or sockets relying on TCP or UDP transport.

A large number of mobile agent platforms are available today, offering a suitable runtime environment for mobile agents as well as a high-level programming API supporting typical mobile agent capabilities. The work presented in this thesis was done using the Grasshopper mobile agent platform developed by IKV++ [Breu98]. Grasshopper was chosen for a combination of benefits offered such as: extended support for agent capabilities, simplicity in usage and agent programming, good support and documentation, support for the MASIF and FIPA standards, etc. The Grasshopper runtime is comprised of a suitable execution environment for agents as well as a naming service catering for mobile agents and any execution environments present. A Grasshopper naming service is termed a "registry" while an agent execution environment is typically referred to as an "agency". Such facilities are offered by all mobile agent platforms, typically in a non-standardized manner, although some first efforts have appeared on MASIF compliance.

2.2 Technologies for Network Management

2.2.1 Protocol-Based

2.2.1.1 SNMP Management

The Simple Network Management Protocol (SNMP) is currently the most widely adopted approach for the management of data networks. The protocol first appeared in the late 80's and addressed what was then perceived as the most critical feature lacking in network management systems: interoperability between multiple vendors.

SNMP follows a manager/agent model of operation, involving a management station that interacts with agents executing in network elements. In this context, the meaning of the word agent reflects on computational entities whose purpose is to provide a standardized interface for accessing information about the network device on which they reside (i.e. similar to Unix daemons, or Windows services). Each agent stores information in a local information base, named management information base (MIB). Management logic located at the management station processes the 'raw' network information obtained through the remote polling of agents. This fine-grained client / server interaction is sometimes called micro-management and leads to a significant increase in traffic and load at the management station. All remote interactions involve standard messages sent within packets called Protocol Data Units (PDUs). These packets are transferred using the connectionless User Datagram Protocol (UDP). SNMP supports the following five message types:

- Get Request: allows access to the agent in order to obtain managed objects values.

- Get Next Request: similar to Get Request but allowing the retrieval of values corresponding to the next logical identifier in an MIB tree.
- Set Request: allows the modification of the value of a managed object.
- Response: a respond to the Get, Get Next and Set Request PDUs. Contains information about the status of the response (e.g. error codes, etc)
- Trap: allows SNMP agents to report the occurrence of a specific event.

One important factor that contributed to the initial success of SNMP management is the simplicity of the protocol. Recently however, this same fact has come to the centre of a number of limitations and disadvantages:

- Lack of Scalability:
    - There is one agent for every network element a fact that can easily result into problems when the agent serves a number of clients.
    - All management processing is performed in a centralized manner at the network management station (see Figure 2-2). Later efforts in the direction of addressing this problem resulted in the Remote Monitoring (RMON) Management Information Base (MIB) [Wald95] in an attempt to decentralize management tasks by performing some data pre-processing in managed nodes.
    - Polling-based SNMP operation results in high amounts of traffic and network latencies. Large portions of redundant managed object identifiers and the lack of a compression mechanism for transferred data
- Unreliable transport protocol: SNMP relies on the UDP connectionless transport protocol that can be unreliable due to its lack of acknowledgements. This introduces the risk of losing critical management information even for trivial reasons such as buffer overflows in IP routers.
- Security: SNMP versions 1 and 2 adopt a weak security scheme with passwords for access to network elements transferred across the network as clear text in unprotected packets. The latest SNMP version 3 has brought major advancements in such security issues but so far the support of this version by major vendors has been slow.
- Limiting low level semantics: Major shortcomings include the absence of high-level MIBs, limited set of SNMP protocol primitives and the data-oriented nature of the SNMP information model.

- Lack of programmability: SNMP capabilities at network nodes are fixed, embedded by the manufacturer during network element construction. Recently, IETF's Script MIB [Levi99] proposal introduces some level of programmability within the SNMP context through the use of mobile code servers.

2.2.1.2 CMIP Management

The International Standards Organisation (ISO) standardized its own approach to protocol-based management in terms of the Common Management Information Protocol (CMIP). The protocol follows a manager/agent model similar to SNMP with a client network management station remotely polling an agent at the network element. The CMIP protocol is used to implement a number of network management capabilities, referred as the Common Management Information Services (CMIS). In contrast to SNMP, CMIP agents are much more sophisticated and capable, but on the downside they are also more difficult to program and require a greater amount of computational resources. Below we give a summary of the main advantages of CMIP over SNMP:

- CMIP variables not only relay information, but also can be used to perform tasks.
- CMIP is a safer system as it has built in security that supports authorization, access control and security logs.
- CMIP provides powerful capabilities that allow management applications to accomplish more with a single request.
- CMIP provides better reporting of unusual network conditions.

Despite these benefits, CMIP failed to gain popularity mainly due to a combination of long standardization cycles, the need for large amounts of network element resources and its programming complexity.

2.2.1.3 DEN Management

Directory-Enabled Networking (DEN) is an industry-standard for the construction and storage in a central directory of information about users, applications, management services, network resources and the relationships among them. In contrast to SNMP, DEN defines an object-oriented information model. The model is mapped into a directory defined as part of the Lightweight Directory Access Protocol (LDAP). Whenever a network, user or application change takes place a number of affected directories are automatically updated through a process called directory replication. The approach of having central directories of information allows the integration of management information from a variety of sources and provides essential underpinnings for policy-based management.

2.2.2 Distributed Objects-based

2.2.2.1 CORBA

While a protocol-based approach to management is specific to a particular management framework, a generic approach to the client/server model can be provided through the use of a distributed object framework. A distributed object framework, with particular influence on the network management area, is the Common Object Request Broker Architecture (CORBA) [CORB95] proposed by the Object Management Group (OMG).

Any relationship between distributed objects has two sides, a client and a server. The server provides a remote interface, and the client calls a remote interface. In CORBA remote interfaces are specified in a special language called the Interface Definition Language (IDL). Any methods included in IDL interfaces can involve parameters and returns in the form of basic data types (e.g. int, double, char, etc.), arrays and structures only. IDL is a very significant part of CORBA, providing a bridge between different programming languages, operating systems, networks and object systems. The interfaces follow the object-oriented methodology making them easy to maintain. IDL is also mapped in a large number of programming languages that can be used to write CORBA systems.

In the context of distributed objects, the terms client and server define object-level rather than application-level interaction, any application could be a server for some objects and a client of others. In fact, a single object could be the client of an interface provided by a remote object and at the same time implement an interface to be called remotely by other objects. In order to support interaction between remote objects, CORBA deploys an Object Request Broker, a class library that provides the low-level communication mechanisms. Between the ORBs, communication proceeds by means of a shared protocol, the Internet Inter-ORB Protocol (IIOP). IIOP, which is based on the standard TCP/IP Internet protocol, defines how CORBA-compliant ORBs exchange information. In addition, to simple distributed communication capabilities, ORBs can provide a number of optional services defined by the OMG. For instance we can have services for looking up objects by name, maintaining persistent objects, supporting transaction processing, etc.

A typical scenario of distributed communication between CORBA objects is as follows. On the client side, the application owns a reference of a remote CORBA server. The object reference has a stub method, which is a stand-in for the method being called remotely. The stub is actually wired into the ORB, so that calling it invokes the ORB’s connection capabilities, which forwards the invocation to the server. On the server side, the ORB uses skeleton code to translate the

remote invocation into a method call on the local object. The skeleton translates the call and any parameters to their implementation-specific format and calls the method being invoked. When the method returns, the skeleton code transforms results or errors, and sends them back to the client via the ORBs.

In the network management context, distributed object frameworks such as CORBA, allow the creation of decentralized, static systems (see Figure 2-2). Decentralization is achieved in a scalable manner by placing required management logic in network nodes and by creating instances of management objects specific to interested clients. Although distributed object frameworks such as CORBA have succeeded in allowing the development of decentralized systems, they still suffer from a lack of support for programmability, as the management logic located in network nodes is static and cannot be easily altered.

2.2.2.2 Java-RMI

The Java Remote Method Invocation (Java-RMI) is a distributed object framework proposed by Sun Microsystems in the mid 90's. The framework follows a typical distributed objects model, similar to CORBA. Unlike CORBA's support for multiple programming languages, Java-RMI is coupled with the Java programming language. While this is seen by many as a drawback, it is this assumption that enables enhanced interactions between Java-RMI distributed objects. As such, Java-RMI remote interfaces contain methods with parameters and returns that may involve any known Java object. In this manner, Java-RMI enhances the typical model of distributed objects exchanging data with the introduction of object mobility. Object mobility in Java-RMI is provided as a convenience enhancement to the distributed objects approach. As such, Java-RMI does not support the notion of a generic object execution environment, which is a basic requirement of a mobile agent platform. Instead, objects can be exchanged in a context specific manner, as supported by a specific Java-RMI server. All remote communication is facilitated by Java-RMI's own transport protocol, called the Java Remote Method Protocol (JRMP). In the context of network management Java-RMI is used by the Java Management Extensions (JMX) initiative [JMX] of Sun Microsystems.

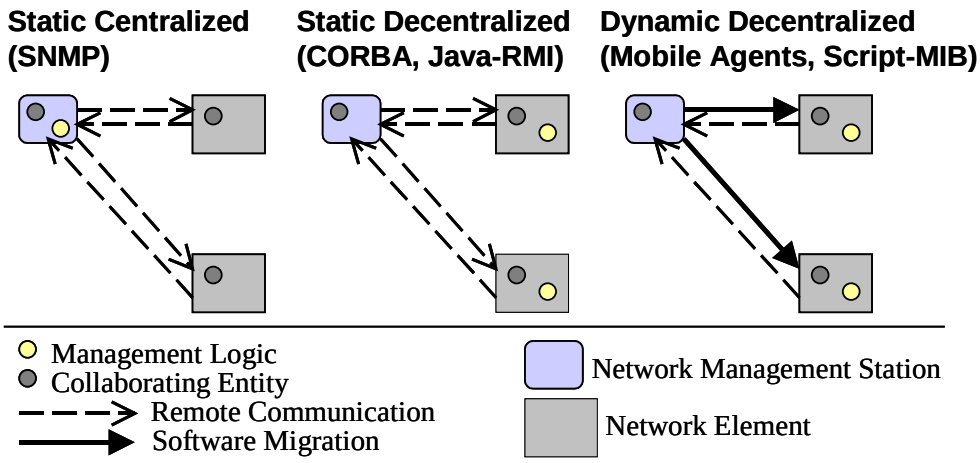


Figure 2-2: Evolution of network management approaches

2.3 Network Architectures

This section describes the main characteristics of architectural frameworks that influenced the case studies of network performance management described later in chapter 5.

2.3.1 Telecommunications Management Network (TMN)

The ITU-T reference model for the TMN represents a management architecture that is tailored to the specific needs of operators of public networks and is aimed at supporting an integrated management of these networks [M3010]. The TMN objectives were as follows:

- It should take only one management network with distributed management functionality to manage different basic networks.
- All five OSI functional areas (FCAPS, i.e. Fault, Configuration, Accounting, Performance and Security management) should be taken into account.
- The entire management pyramid involving Business, Service, Network and Network Element management should be taken into account.
- Manufacturer specific FCAPS concepts should be supported (catering for heterogeneity).
- Interfaces should be defined between the domains of the carrier, between provider and customer, as well as between the different systems for the purposes of management information flow.

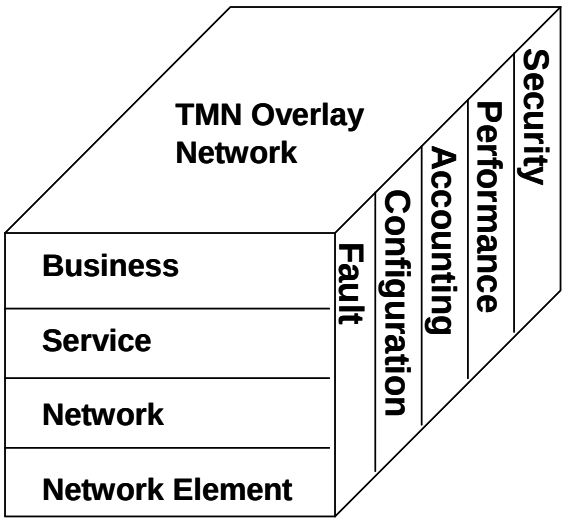


Figure 2-3: TMN management dimensions

The TMN reference model incorporates a logically separate management network (the TMN overlay network) for the interaction between different management-relevant components as shown in Figure 2-3. Its architecture and management activities are based on the so-called functional blocks described below (Figure 2-4):

- Telecommunications Network (TN): Individual sub-network offered by a carrier (e.g. ISDN, PSTN, X.25, etc). A TN is located outside the TMN.
- Network Element (NE): A component that provides TN users with the network functionality needed and with interfaces to TMN (Examples include, switching nodes, routers, multiplexers, etc.)
- Operations System (OS): A component that processes management information for the purposes of controlling or monitoring a TN (i.e. the actual management system).
- Mediation Device (MD), Q-Adaptor: These TMN components support the forwarding of management information between NEs and the OS and are therefore management gateways between the Q3 and Qx interfaces.
- Workstation (WS): A component that enables human users to access the TMN.
- Data Communication Network (DCN), Local Communications Network (LCN): These components allow communication to take place between other TM entities. These are therefore the transport networks for management information.

Additionally, the TMN reference points of interactions between the functional blocks are as follows (Figure 2-4):

- F: Interface to WS.

- X: Interface to other TMNs.
- Qx: Interface for the connection of simple transmission and switching equipment via non-standard protocols.
- Q3: Interface for the connection of complex equipment (or entire switching nodes) and OSs.

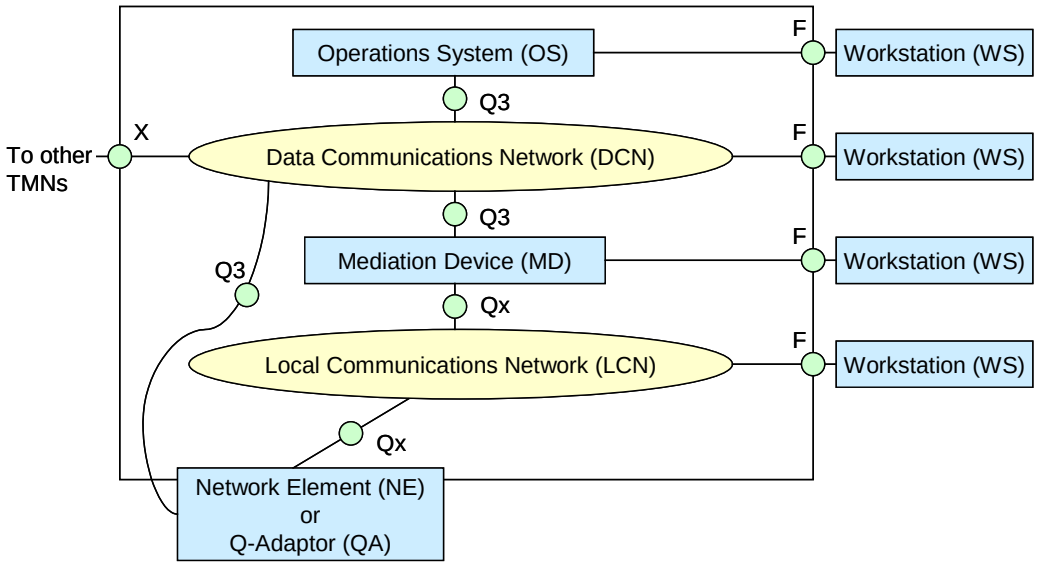


Figure 2-4: The TMN reference model

The TMN standards importantly provided the foundation for a functional network management architecture. A number of mechanisms and techniques were specified in order to deliver a base of management capabilities for all five OSI functional areas. The mobile agent-based performance monitoring capabilities developed during this thesis work are based on relevant TMN standards. In TMN these capabilities are offered in a static manner (i.e. they are fixed within network elements and cannot be changed), a fact the crucially hindered its success. In contrast, the work presented in this thesis attempts to suitably exploit mobile agent attributes in order to deliver functionality for performance monitoring in a programmable and autonomous manner.

2.3.2 Telecommunications Information Networking Architecture (TINA)

The TINA architecture attempts to define an all-encompassing, distributed software architecture for future telecommunication systems. TINA was designed to insure interoperability, portability and reusability of software components and independence from specific technologies. The architecture allows the rapid introduction of new services by sharing the burden of creating and managing a complex system among different business stakeholders, such as consumers, service providers, and connectivity providers [TINA95]. Furthermore, the TINA specifications describe

"how" rather than "what" and can thus complement other functional architectures such as Intelligent Networks (IN) and TMN.

An important building block in TINA is the Distributed Processing Environment (DPE), a generic platform for the deployment and execution of software components. Based on this, two important separations are made. First, TINA considers a separation between the application and the DPE. Second, it considers a further separation between applications into a service specific part and a generic management and control part. According to these separation principles the architecture is divided into the following three sub-architectures:

- Computing Architecture: Defines the concepts and principles for the development of software components and a supporting DPE.
- Service Architecture: Defines the concepts and principles for providing services.
- Network Architecture: Defines the concepts and principles for a generic and technology independent model of setting up connections and managing telecommunications networks.

Another important aspect of the TINA architecture is its business model. This defines the different parties involved in service provisioning and the relationship with each other. A small number of roles are defined in the business model, which reflect the major business separations of a complex telecommunications and information market. Between the different roles, we have reference points comprising of a set of interfaces that describe the interactions taking place between these roles [TINA97]. The five business roles identified are (also see Figure 2-5):

- Consumer: A stakeholder that takes advantage of the services provided in a TINA system.
- Retailer: A stakeholder that serves the consumer business role.
- Broker: Provides other stakeholder with information that enables them to find other stakeholders and services in the TINA system.
- 3rd Party Provider: Supports retailers or other third party providers with services.
- Connectivity Provider: This stakeholder owns (manages) a network (switches, cross-connects, routes and trunks).

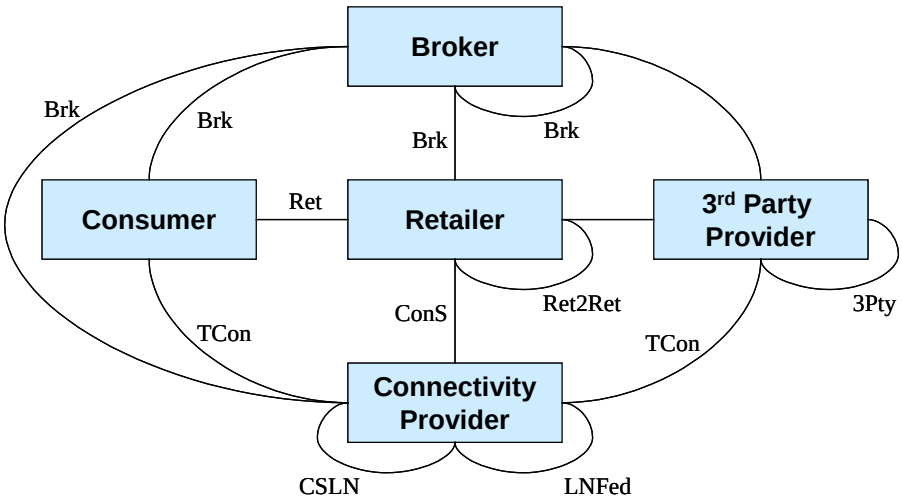


Figure 2-5: The TINA business model

The reference points identified are as follows (also see Figure 2-5):

- Client-server layer network relationship (CSLN): Provides the use of layer networks between business administrative domains performing the connectivity provider business role.
- Layer network federation business relationship (LNFed): A federation relationship between connectivity providers, allowing the provisioning of a connectivity service spanning multiple business administrative domains implementing the connectivity provider role.
- Terminal connection business relationship (TCon): Provides the link management between the connectivity provider business role and the business roles of the consumer or 3rd party provider.
- Connectivity service business relationship (ConS): Provides the network transport services (point-to-point and point-to multi-point) and the business roles (typically the retailer and third party service provider) using the transport connectivity services (on behalf of their customers).
- Retailer-to-Retailer business relationship (Ret2Ret): Re-uses the functionality from the 3Pty and the Ret business relationships considering the fact that the information passed over the reference point may be different, but the actual interactions are not.
- Third party business relationship (3Pty): Interaction with a stakeholder in the third-party service provider business role to provide a broader range of services to its consumers without actually possessing the services.

- Broker business relationship (Bkr): Allows the access and management of broker information by any of other TINA business roles.
- Retailer business relationship (Ret): Used between stakeholders in the consumer business role and stakeholders in the retailer business role.

Within the context of this thesis, the TINA business model was adopted as a basis for the effective separation of roles and responsibilities in the context of the three case studies described later in Chapter 5.

2.3.3 3GPP OSA and Parlay

Today, there is a lot of activity on the preparation of the so-called 3rd Generation communications systems and associated services. The current vision involves the following main ideas:

- Abstraction: allow the homogeneous opening of the network for application development
- Allowing more combinations: An application can combine different network capabilities
- Application convergence: fixed/mobile/IP migration and convergence, application convergence/integration between public and enterprise domain

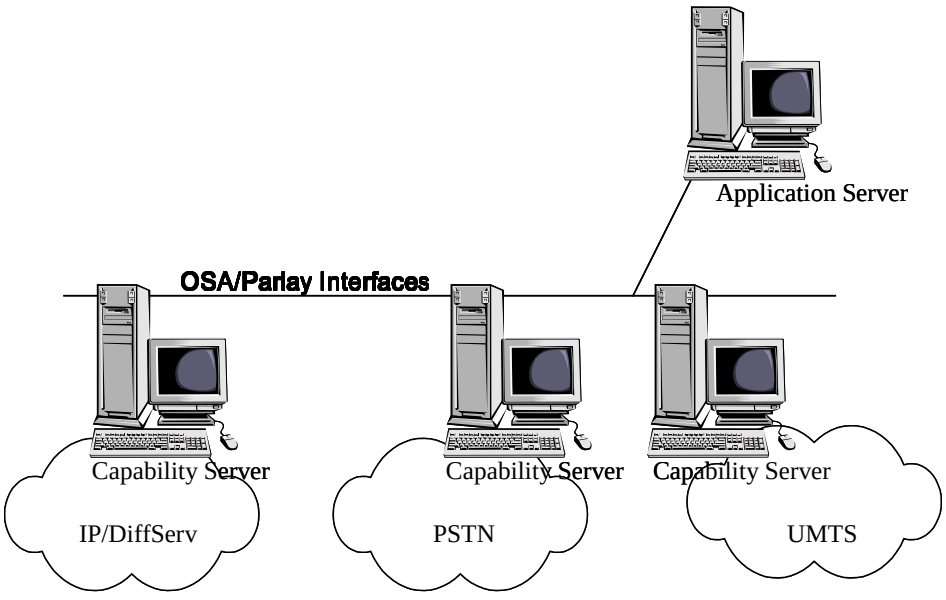


Figure 2-6: The role of OSA/Parlay Interfaces

Currently, large industrial consortia, most importantly the 3rd Generation Partnership Project (3GPP) (with its Open Service Access (OSA) APIs) as well as the Parlay group [PARLAY], are working in a merging direction towards the specification of open interfaces for services that can operate across multiple networking platform environments. The combination of OSA/Parlay, crucially offers functionality that allows connection establishment and management for both

mobile and fixed underlying network infrastructures. The approach provided is guaranteed to be secure, independent of vendor specific solutions and also independent of programming language by use of generic Object-Oriented techniques.

The 3GPP efforts on OSA/Parlay provide an important attempt to open up network capabilities of a network provider to service providers. The work is still in progress and in particular regarding network management capabilities there are many plans that still need to be delivered. The influence and effect of OSA/Parlay specifications in relation to this work on mobile agent-based performance management are described in Chapter 5 within the context of the case study on the Virtual Home Environment (VHE).

2.4 Performance Management

Performance management aims to provide awareness of network conditions and ensure that the network performs well. A number of mechanisms and techniques are typically used, often in a collaborating manner, in order to achieve this goal. Performance monitoring, involving the systematic monitoring of network resources is often considered a key component. Also often important are QoS management capabilities that allow the configuration of qualitative or quantitative guarantees of network performance. The most important mechanisms and approaches used to deliver these capabilities are described in the sections that follow.

2.4.1 Metric Monitoring and Summarization

Metric Monitoring [X739] and summarization [X738] functions were standardized by ITU-T in the context of the TMN specification work. The specifications describe important mechanisms that were applied in the work of this thesis, as the basic functionality expected from a performance monitoring system. Metric monitoring involves the observation of a counter or gauge attribute, converted into a derived gauge attribute after every observation. Within this context the time between two successive observations is termed “granularity period”. If a counter is observed, the metric monitoring process derives a gauge value equal to the difference between successive observations of the counter. If a gauge is observed, the derived gauge value is equal to the observed attribute at the time of observation. A moving average or other statistical smoothing algorithm can be applied on the derived gauges observed in order to eliminate issues associated with random bursts. A derived gauge can be associated with a number of upper or lower thresholds checked using a mechanism called “Severity indicating gauge-threshold”, illustrated in Figure 2-7. In this figure we can see the value of the derived gauge as well as two thresholds, an upper threshold 1 and a lower threshold 2. Each threshold has an associated level value, as well as a clear value. If the threshold level is crossed by the derived gauge, a threshold notification is

send. Any ripple and subsequent crossing of the threshold level does not result in further notifications sent until the derived gauge crosses the threshold clear value.

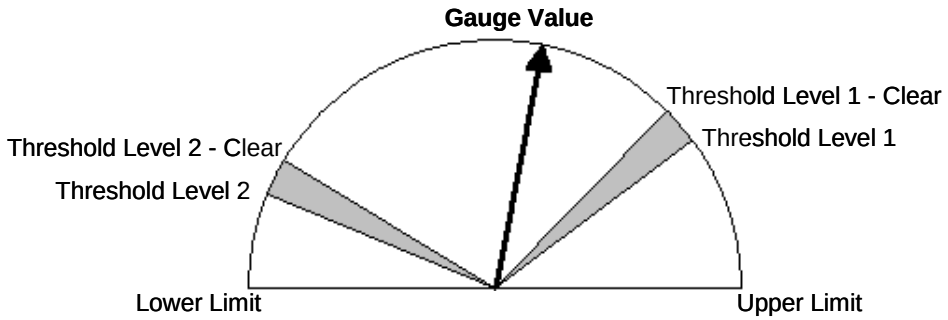


Figure 2-7: Severity Indicating gauge-threshold

Additionally, the summarization process allows requests of a number of derived gauge attributes that are reported periodically in a scheduled manner. This function is particularly important for gathering historical performance data for capacity planning, trend analysis and logging. Finally, It is important to note that both Metric Monitoring and Summarization capabilities are not linked to any specific network technology and thus have a wide scope. As such they were conveniently used as the basis of performance monitoring capabilities within all three case studies detailed in Chapter 5.

2.4.2 IP Networks

2.4.2.1 Performance Monitoring

In the IP world there are typically two ways to gather performance monitoring information:

- Passive Measurements: Information is gained as a result of the analysis of the traffic passing in a network element (e.g. for measurements of used bandwidth, loss, etc).
- Active Measurements: Information is gained by inserting (low-impact) test streams (e.g. for measurements of delay, jitter, etc).

There are typically a number of different performance parameters we are interested to obtain with passive and active measurements [Paxs98]. For example, in a router, utilization can be measured by packet forwarding rate, the processor load, percentage of dropped frames on each interface and number of packets being held in a queue. Such a network element is best monitored by examining its forwarding rate versus its forwarding capability in a given network scenario, because they typically can process frames more quickly than their associated links can deliver them. A high processor load tells you that the device has little idle time. This usually is not a problem unless the network element becomes busy 100% of the time and cannot continue to process network traffic. In this case the high utilization of the device may result in packets being discarded, forcing the

source system to retransmit them. If the router can process the packets but needs to queue them, this can also affect the overall network performance. The device may be waiting on a heavily utilized network link, a link with many errors, or another network device that may be experiencing performance problems. Measuring the link utilization is a critical task for performance management. Typically, link utilization is the amount of bits per second sent and bits per second received, divided by the total available bandwidth (also measured in bits per second). A significant manifestation of over-utilization on network devices and links is a noticeable decrease in the level of service to users. To measure the level of service, one needs to determine the following.

- Total Response Time: The amount of time it takes a datum to enter the network and be processed and for a response to leave the network. The IETF IPPM working group has provided a detailed description regarding the round-trip [Alme99b] as well as the one-way delays [Alme99a].
- Rejection Rate: The percentage of time the network cannot transfer information because of lack of resources and performance. Most devices provide the information on the number of frames that have been rejected on a specific interface.
- Availability: The percentage of time the network is accessible for use. This is often measured as mean time between failure (MTBF). Availability can be measured on a theoretical level, and most device vendors can provide those numbers. On a practical level most devices and links can provide the information on how long they have been operational.

Today, the IETF-proposed SNMP protocol is the most popular technology used to collect management information from network elements. This typically involves the remote polling for the values of objects defined in a Management Information Base (MIB) supported by the network element. Widely supported objects for performance management at the interface, IP, TCP or UDP level are specified in MIBs defined by the IETF in [McCl91] [McCl94]. Today, there are also attempts for alternative technologies to be supported at the network element level, most notably CORBA. In contrast to a specific technology support from a network element, a possible inclusion in the future of a Java Virtual Machine (JVM) would open a window of opportunity for the support of many other technologies, including mobile agents, at the network element level.

2.4.2.2 QoS Management

The aim of QoS management is to provide guarantees of network performance. Today, there are two prominent approaches to QoS management for IP networks namely, Integrated Services (IntServ) and Differentiated Services (DiffServ).

The Integrated Services model is based on the idea of reserving resources within each router involved in the transfer of a specific user packet stream called a flow. The Resource Reservation Protocol (RSVP) is a signalling protocol used by applications in order to reserve resources. A typical resource reservation scenario using RSVP is initiated by the sender node sending a PATH message to the receiver containing the characteristics of the traffic. When this message reaches the receiver responds with a RESV message requesting resources for the flow. Every intermediate router along the path receives the request and if accepted the router reserves link bandwidth and buffer space for the flow and stores relevant state information for this. RSVP is one of four components required by the Integrated Services model as described below:

- Signalling Protocol (e.g. RSVP): Requests the reservation of resources for a specific flow.
- Admission Control Routine: Decides whether a request for resource reservation can be granted.
- Classifier: Classifies packets and places them in the appropriate queue.
- Packet Scheduler: Schedules the packet delivery according to QoS requirements.

The Integrated Services model manages to provide the following two service QoS classes:

- Guaranteed Service: For applications requiring a fixed delay guarantee.
- Controlled-load Service: For applications requiring a reliable and enhanced best effort service.

Unfortunately a number of problems have been associated with the Integrated Services model mainly related to scalability and the increase of state information in routers proportionally with the number of flows, along with the high processing overheads in routers equipped with the RSVP, admission control, classification and packet scheduling components.

In an effort to address the issues associated with Integrated Services, the Differentiated Services model can be characterized as a relative-priority scheme and provides a very different solution to QoS management for the Internet. Instead of considering resource reservation in routers the approach proposes the appropriate marking of IP datagrams with QoS information and their preferential treatment by routers along the path. Differentiated Services defines the layout of the Type of Service (TOS) field (also referred as Differentiated Services or DS field) of the IP header as well as a base set of packet forwarding treatments called per-hop behaviours (PHBs). A customer that requires Differentiated Services must have a service level agreement (SLA) with a network provider. The SLA contains all the information that needs to be negotiated such as the service classes supported and the amount of traffic allowed in each class. When an SLA has been established the customer can mark the DS fields of individual packets, which are then classified,

policed and possibly shaped at the ingress routers of the network provider according to the SLA. When an IP datagram enters one domain from another domain, its DS field may be re-marked as determined by the SLA between the two domains.

The Differentiated Services approach can provide a number of services such as:

- Premium Service: For applications requiring low delay and low jitter.
- Assured Service: For applications requiring a reliable and enhanced best effort service.
- Olympic Service: Provides strict, quantitative guarantees of QoS in three tiers of services, gold, silver and bronze, with decreasing quality.

Since a service is allocated in the granularity of a class rather than per flow as in IntServ, the amount of state information in routers is proportional to a limited number of classes rather than the number of flows. Thus, the Differentiated Services approach is much more scalable. The Differentiated Services approach was used along with suitable performance monitoring facilities with the context of the "QoS Configuration and SLA Auditing of IP-DiffServ Networks" case study detailed later in Chapter 5.

2.4.3 ATM Networks

Asynchronous Transfer Mode (ATM) is a connection-oriented packet switching protocol. Information is carried in fixed size cells along virtual channels, contained in virtual paths. ATM is designed to support various kinds of connections, voice/video/data etc. using statistical multiplexing of sources to achieve efficient resource utilisation.

Every incoming call is characterised by its type – Constant / Variable / Available / Unspecified Bit Rate (CBR / VBR / ABR / UBR) and the Quality of Service (QoS) it requires. The QoS of a connection consists of parameters like the Mean Cell Rate (MCR), Peak Cell Rate (PCR), Cell Transfer Delay (CTD), Cell Delay Variance (CDV), Cell Loss Ratio (CLR), Cell Error Ratio (CER) etc. defined for CBR and VBR services. The parameters CTD, CDV and CLR are the most important for real time connections in which delays are interruptions in service and losses may not be recoverable using re-transmissions. Real time connections are usually CBR or VBR depending on the coding of the source.

In the connection set-up phase a route is determined, with the network guaranteeing (this guarantee is of a statistical nature) the QoS required by the connection along the complete path. The network has to provide this QoS throughout the duration of the connection along the complete path but is allowed to drop cells if the source violates the QoS contract. To this end, performance management becomes necessary (i.e., measurement and control of the QoS provided). Performance management crucially supports tasks related to congestion control, traffic

management and network planning. Monitoring of QoS parameters of a virtual channel (or path) is also useful for QoS auditing purposes, helping to ensure that the agreed QoS guarantees are actually delivered.

Performance monitoring aspects of ATM networks are specified in the Operations and Management (OAM) standard for ATM, in the I.610 recommendation of the ITU-T [I610]. There are two vertical OAM levels defined for the ATM layer, the virtual path level and the virtual channel level. OAM flows are provided by cells dedicated to VPCs (or VCCs) and follow the same path as user cells. These flows can be further classified into two horizontal levels, end-to-end flows (from a VP (or VC) endpoint to another VP (or VC) endpoint) and segment flows (defined as single or multiple inter-connected VP (or VC) links). Intermediate nodes can monitor existing flows and insert new OAM cells but cannot terminate flows which are not their own. The OAM functions defined for the ATM layer are fault management, continuity check, loopback and performance monitoring.

Performance monitoring is usually performed by inserting end-to-end or segment monitoring cells at the VP or the VC level. These cells monitor a block of user cells (of size 128, 256, 512, 1024). The main objective of this monitoring is to detect error blocks and loss / mis-insertion of cells. There are forward monitoring cells to measure and backward reporting cells to report the measured values to the source. The various fields defined for these cells include,

- A sequence number to identify the cells.
- The total number of user cells sent (a modulo 64K counter) and total number of user cells with high Cell Loss Priority (CLP = 1).
- A block error detection code (even parity Bit Interleaved Parity – 16)
- An optional time stamp for delay measurements.
- For backward monitoring cells – total number of user cells received, total number of CLP=1 user cells and the block error result on the received cells.

The method provides only limited information regarding the delays and losses, which are critical to real time services. The standard does not specify how these values are used for pro-active or re-active control actions required if QoS guarantees are not being met. Distributed performance management system is necessary for that purpose, containing objects equipped with performance management algorithms. This system could use OAM cells to get basic performance information. Alternatively, such a system can use a different approach of obtaining this information, for example by inserting fixed cells and measuring delays or polling the information provided by an ATM switch. Regarding this, the SNMP protocol is today widely supported in ATM switches and the IETF has done considerable work for the standardization of ATM-related MIBs [Tesi99].

Chapter 3

3 Related Work

This chapter complements the previous one by focusing more closely on the review of work related to the thesis. As already stated, the thesis investigates the role of mobile agent technology for network performance management. The information presented here involves the exploitation of mobile code and agent approaches in the context of network management. The descriptions follow a number of different viewpoints, including related paradigms, techniques, enabling technologies and applications.

3.1 Code Mobility in Network Management

3.1.1 Paradigms

There are three important and widely referred mobile code paradigms introduced by [Bald97] describing in general terms, ways of exploiting code mobility. The first of these paradigms is called Code on Demand (COD) (Figure 3-1). In this paradigm a client downloads (pulls) required code and initialises it in order to perform a task. The code is downloaded from a "code server" where software components are stored. In the COD paradigm the client owns the resources needed by a task but lacks the logic required to perform it.

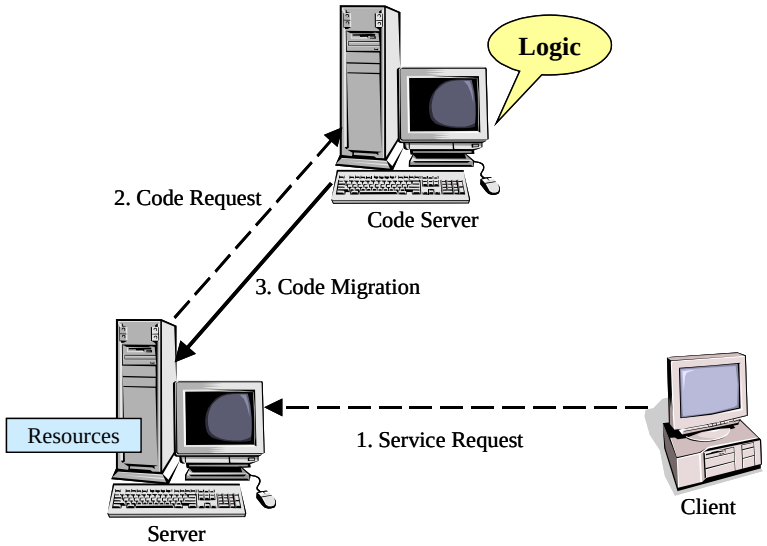


Figure 3-1: Code on Demand (COD)

The second paradigm (Figure 3-2) is called Remote Evaluation (REV) and it was first introduced as a concept in the work of [Stam90]. In this paradigm the client uploads (pushes) the code containing the required logic along with initial parameters to a remote server for execution. Hence, in REV the client owns the logic required to perform a task but lacks the resources need.

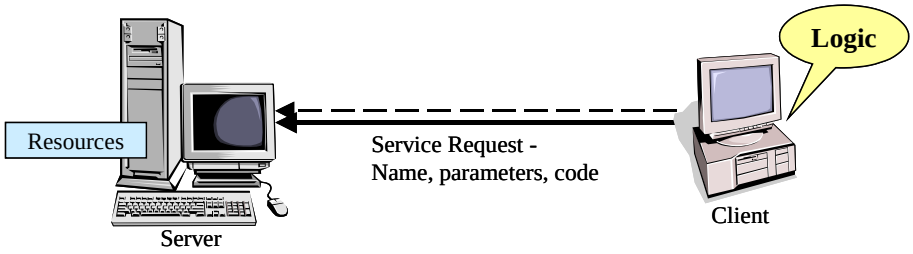


Figure 3-2: Remote Evaluation (REV)

The third case is termed the Mobile Agent (MA) paradigm. Within this context an agent is just an Execution Unit (EU) containing the logic to perform a task. While it executes in a network, it can decide to migrate autonomously to a remote node in order to have local access to required resources. Hence, in the MA paradigm the agent owns the logic required to perform a task but lacks the resources needed. The transfer of agents between different nodes is supported by two different migration mechanisms [Fugg97]. Migration can be either proactive or reactive. In proactive migration the time and destination for migration are determined autonomously by the migrating agent. In reactive migration the transfer is triggered by a different object that has some kind of relationship with the agent to be migrated.

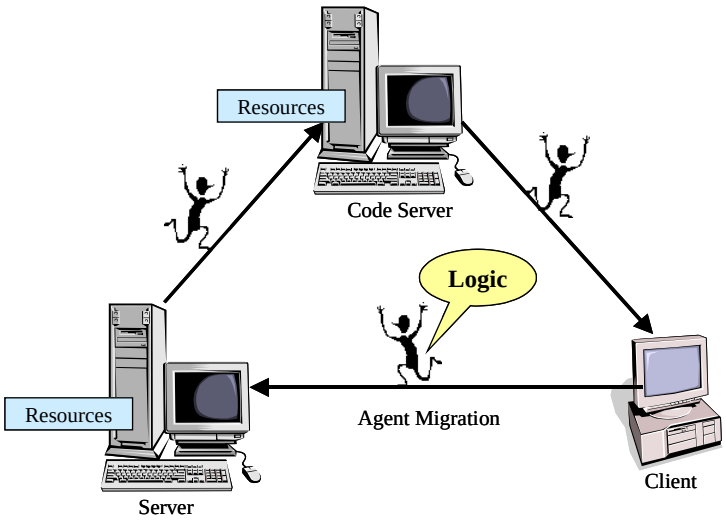


Figure 3-3: Mobile Agent (MA) paradigm

The above paradigms are derived from the analysis of the characteristics of current Mobile Code Languages (MCLs). The idea behind these languages is to overcome the limitations of the client/server paradigm by embodying the notion of code mobility. Java [Java95] is an example of

an MCL that supports directly the COD paradigm in the form of Java Applets. The Mole [Stra96] project and TACOMA [Joha95] are two examples of systems supporting the REV paradigm with procedures that can be sent to a remote host for execution. In Telescript [Whit94], Agent-TCL [Gray95] and Grasshopper [Breu98] the MA paradigm is naturally supported using a special agent process that can suspend its execution, migrate and resume when it reaches its destination.

The ability of an MCL to migrate code is divided in two different levels [Carz97]. In Strong Mobility the MCL allows EUs to move their code and execution state (in the form of the program execution stack) to a remote node. In Weak Mobility the MCL allows the transfer of code only without the execution state. Strong MCLs are a minority because of the challenging problems posed in defining and implementing the semantics of strong mobility. Telescript and Agent-TCL are two examples of such languages. Weak mobility on the other hand is more common, with Java, Mole, TACOMA and Grasshopper being good examples.

These paradigms and categorizations of code mobility have provided an important starting base for this thesis. Many important aspects of the work presented above need to be researched taking into account the autonomous attributes of software agents within the network management context. The work of this thesis goes a step further by considering the mobility strategies of autonomous agents in the network management context along with guidelines for their suitability and associated benefits.

3.1.2 Management by Delegation

Management by delegation (MbD) [Yemi91][Gold96] constitutes a first clear effort of exploiting mobility in order to decentralise network management operations. The MbD approach is based on delegation-agents, programs that can be linked and executed under local or remote control. Delegation-agent programs can be written in arbitrary languages, interpreted or even compiled. This allows for better handling of time critical tasks such as real-time monitoring and control of network resources.

A typical system following the MbD approach includes the following features:

- Elastic Server: A program that can be modified, extended and/or contracted during its execution. Elastic servers are pre-instantiated in network nodes, providing the environment that supports delegated-agents arriving from a remote node.
- Delegating Client: Responsible for the transfer of a delegated-agent to an elastic server and also the remote controlling of its execution.
- Delegated agent: A piece of code containing management logic that can be transferred to an elastic server for execution.

- Remote Delegation Protocol (RDP): An application-layer protocol used by delegating clients to transfer code to an elastic server and remotely control its execution. The supported operations include: delegate/delete, instantiate/terminate, suspend/resume and get state/set state.

A typical scenario of delegation can be seen in Figure 3-4. A delegating process D transfers a delegated-agent program DA to an elastic server (ES). For example, D can be a manager process sending a performance monitoring program to an ATM switch. In effect, the manager process D attempts to delegate an agent DA with the execution of a task in the elastic server ES. If DA is accepted by the elastic server, D receives a handle that can later be used to instantiate a DA process inside the environment of ES.

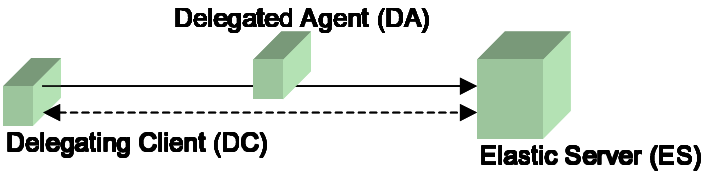


Figure 3-4: Management by Delegation

MbD can inter-operate, extend and decentralise current network management protocols, thus current management applications can also benefit from its use. For example, MbD can be used to dynamically extend the capabilities of an SNMP agent. An important feature is that delegated-agents execute asynchronously with respect to the delegating client. This allows the network management station to perform other tasks in the meantime, introducing a higher degree of parallelism in the management architecture. Some network management standards bodies, namely the IETF and the ITU-T, have been working on integrating the MbD model into their management frameworks [Scho97] in order to increase the scalability and flexibility of their proposals. The approach followed was to define management functions as scripts, with the term used to emphasise that management functions will be highly customisable. Scripts are delegated to operate in remote nodes, with their transfer and execution controlled by means of the respective management framework. The ITU-T proposal was named the CMIP Command Sequencer, while the IETF standardised the SNMP Script-MIB.

The MbD approach more than a decade ago provided a foundation for exploitation of mobility for programmability in management. This thesis takes this into account and considers enhancements of the approach, as motivated by later developments in agent, object-oriented and component technologies, along with the requirements of modern services and network environment.

3.1.3 SNMP Script-MIB

Technologies based on software mobility have failed to gain popularity, mainly due to the lack of standards and fears of associated security risks. An important effort towards mobility-based network management is the Script-MIB [Levi99][Scho00] proposal of the IETF. Script-MIB follows a "Management by Delegation" approach based on mobile code that is sent to a remote node for execution. The technology defines an SNMP-compliant Management Information Base for the delegation of management functions. In particular, it provides the following capabilities:

- Transfer of management scripts to a distributed manager.
- Initiating, suspending, resuming and terminating management scripts.
- Transfer arguments for management scripts.
- Monitor and control running management scripts.
- Transfer the results produced by running management scripts.

Script-MIB assumes that management functions are defined in the form of executable code (scripts) that can be installed in network nodes. The MIB supports arbitrary programming languages and makes no assumptions about code formats. It also allows the delegation of compiled native code, if an implementation is able to execute native code under the control of the Script-MIB. This is an important difference compared to the CMIP Command Sequencer approach that chose to define a special script language (Systems Management Script Language, SMSL) in order to facilitate interoperability at the script level.

The technology supports two different ways to transfer management scripts to a distributed manager. The first approach, termed the "push model", requires that the manager pushes the script to the distributed manager. The second approach termed the "pull model", first requires the manager to inform the distributed manager of the location of a particular script. This is followed by the retrieval of the script from the distributed manager.

The delegation protocol proposed by the IETF is the Script-MIB Extensibility protocol (SMX) [Scho99]. The protocol can be used to separate language specific runtime systems from the runtime system independent Script-MIB implementations. Figure 3-5 shows the process and communication model underlying the SMX protocol.

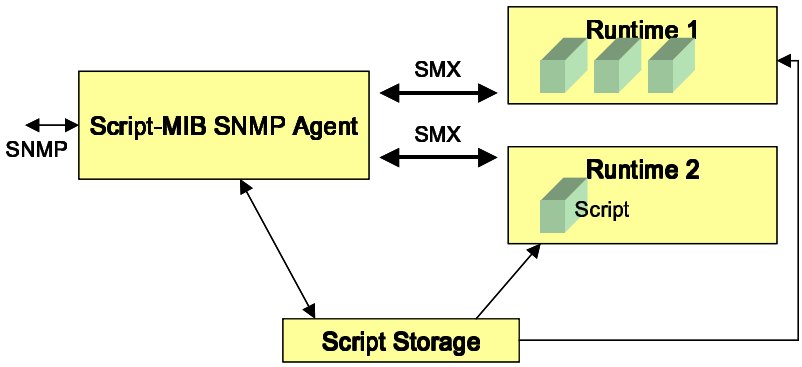


Figure 3-5: SMX process and communication model

The language and runtime system independent SNMP agent implementing the Script MIB communicates with one ore more runtime systems via the SMX protocol. A runtime system may be able to execute one or multiple scripts simultaneously (multi-threading). The SMX protocol supports multi-threading, but it does not require multi-threaded runtime systems. The SMX protocol uses a local storage device (usually implemented on top of the local file system) to transfer scripts from the SNMP agent to the runtime systems. The SNMP agent has read and write access to the script storage device while the runtime systems only need read access. The SMX protocol passes the location of a script in the local storage device to the runtime engines. It is then the responsibility of the runtime engines to load the script from the specified location.

The Script-MIB standard has seen its first implementation recently in the form of the Jasmin platform [Jasmin]. In later chapters of this thesis we examine in detail the limitations of Script-MIB and its MbD approach along with a comparative experimental assessment involving among others the Jasmin platform.

3.2 Mobile Agent-based Network Management

3.2.1 Mobile Agents, Objects and Code

A lot of research on software mobility during the 90's concentrated on mobile code (e.g. [Papa00b][Picc98][Hall97][Gold96]), neglecting many interesting aspects of object and agent mobility. The notions of code, object and agent mobility can be distinguished as follows:

- Code mobility: Involves a piece of code that is transferred to a remote node where it is initialised into an object and begins its execution.
- Object mobility: Involves an executing object that is terminated, serialized and transferred to a remote node where it is re-initiated for execution.

- Agent mobility: Involves an executing autonomous object that is terminated, serialized and transferred to remote node where it is re-initiated and resumes execution.

In order to further illustrate the above differences Table 3-1 presents a number of alternative technologies and their respective support for software mobility.

Table 3-1: Mobility of agents, code and objects

Transferred Entity	**Description**	**Example Relevant Technologies**
Mobile Code	Involves code that is transferred to a remote node where it is initialised into an object and begins its execution.	Script-MIB, Java Applets
Mobile Object	Involves an executing object that is terminated, serialized and transferred to a remote node where it is re-initialised for execution.	Java-RMI, Microsoft .Net Remoting
Mobile Agent	Involves an executing autonomous object that is suspended, terminated, serialized and transferred to a remote node where it is re-initialised to resume execution.	IKV Grasshopper, IBM Aglets

3.2.2 Proposals on Mobility

In recent years a number of researchers focused on the exploitation of mobility. The models of mobility investigated can be summarized in terms of single-hop and multi-hop agent mobility. In single-hop mobility an agent visits a single targeted node, while in multi-hop mobility an agent visits and operates sequentially at a number of nodes. Regarding single-hop mobility many researchers have overlooked the combined benefits associated with agent autonomy, and thus have produced results that reflect more on mobile code or mobile object approaches. For instance [Gava01][Rubi99][Saha99][Zapf99][Isma99][Puli00] consider agents that carry management logic to a targeted node offering a performance improvement over standard SNMP polling where the same management processing is done at the network management station. Agent behavior regarding autonomous management operation, adaptability to changes in the agent environment, or self-configuration are not considered here. Assessments of the agent approach were limited to practical comparisons with SNMP, neglecting several emerging distributed objects and mobile code approaches for network management. Furthermore, many researchers have proposed single-hop mobility of agents where the mobile agent performing the task returns to the network

management station for the purpose of delivering the results gathered. Instead, this task should be performed remotely, with significantly less network overheads that would otherwise be incurred by the unnecessary transfer of management logic to the network management station. This definition and application of single-hop mobility is clearly inappropriate and unjustifiable. The same observations stated above also apply for the case of multi-hop mobility where we currently see limited autonomy and often unnecessary migrations of mobile agent logic. What is really the importance of single-hop and multi-hop agent mobility in network management? Do we need single-hop or multi-hop agents in network management or could we do simply with mobile code, mobile objects or even static distributed objects? These are some important questions regarding agent mobility that require further work in order to provide a solid answer and it is the aim of this thesis to contribute in this direction.

3.2.3 Proposals on Autonomy

Agent autonomy has been typically exploited in order to provide adaptable, fine-tuned and fault tolerant network management solutions that minimize the need for user interaction.

Autonomy in the context of fault-tolerant operation is examined in the work of Cheikhrouhou, et al [Chei00a], investigating solutions that allow a graceful adaptation of network management systems to unreliable agents. When an unreliable agent is detected, the management tasks it was performing are re-distributed amongst other still reliable agents. The agents of the management system periodically run a number of tests on each other, comparing their beliefs. If the two do not much then either the tester or the tested agent is unreliable. System level diagnosis techniques are then used to deduce the reliable entities (the same approach is proposed in [Marc99]). Another proposal presented in [Cari98] considers fault tolerant operation relating to the network environment. In order to allow mobile agent response to critical conditions the authors propose mobile-agent systems equipped with a network-awareness infrastructure. Mechanisms for providing such an infrastructure are presented along with the potential of using the mobile agents themselves and other communicated objects as carriers of network status information within such an infrastructure. A similar approach to network-awareness in mobile agent systems is followed in Sumatra [Acha97], an extension to the Java programming language for the support of resource-aware mobile programs that can adapt to changing network conditions.

Regarding the fine-tuned operation of a network management system Abdu et al [Abdu99], have described an adaptive model for initial optimal configuration of management agents according to user/network requirements as well as subsequent reconfiguration (i.e. re-location) of these agents according to changing conditions. The problem of determining the optimal number and location of agents is solved using an enumeration-based algorithm aiming at optimality. By aiming at

location optimality this algorithm is characterized by exponential complexity, making it viable for small-scale systems only. Liotta [Liot01b] [Liot02a] has also worked on this problem and has proposed a simpler algorithm that approximates an optimal solution and offers a more scalable approach.

In another direction the exploitation of multi-agent systems was proposed through the formation of agent swarms with ant-like behaviour [Whit98a]. In this case, each agent is 'chemically' inspired and proposes 'chemical' interaction as the principal mechanism for inter-swarm communication. 'Chemical' messages have two attributes, a label and a concentration with the latter defining a 'weight' that leads the process of behaviour selection. Agents within a given swarm have behaviour that is inspired by the foraging activities of ants with each agent capable of simple actions, while no single agent has knowledge of the global goal. The creation of chemical trails is proposed as the primary mechanism used for distributed problem solving arising from the self-organization of swarms of agents. While the approach has been applied to a few network management tasks (e.g. for fault management [Whit98c]) there is no thorough evaluation of swarm systems including their performance aspects and as such it is not clear whether they provide an efficient and desirable solution to network management tasks.

3.2.4 Applications

In our days, there is an ongoing research on the use of mobile agents in network management. Mobile agents are particularly useful for tasks involving distributed information retrieval and processing. In this direction, they have been successfully deployed in complex network management systems covering most functional areas, as collectively presented in a number of mobile agent surveys [Bies98a] [Chei98] [Mage96] [Kaha97] [Hayz99] [Wren99]. Some notable applications of mobile agents are described below.

3.2.4.1 Performance Management

Performance monitoring involves the processing of a large amount of performance information extracted from the network. Gavalas et al [Gava99a] proposed that instead of remotely polling network elements, a mobile agent can be dispatched to perform its task locally and return to the network management station to deliver any reports gathered. The author has also considered multi-hop agents for performance management and investigated a hierarchical approach where the network is divided in many domains, each the responsibility of a single agent. Experimental measurements of the mobile agent-based performance monitoring system showed superior performance compared to SNMP polling [Gava99b][Gava99c][Gava00] especially when the health function involved depends on a large number of resource values. Better performance is identified in terms of reduced network management traffic, decentralization of processing and

better response times unaffected of network latencies. Furthermore, experimental results on the hierarchical approach to performance monitoring aimed to identify an efficient relation between the number of agents deployed, and the number of network nodes assigned to each agent. The agents of this system do not exhibit any autonomy and are merely perceived as distributors of management logic that would otherwise (i.e. in the SNMP case) execute in the network management station. In another notable work on performance management by Wijata et al [Wija00], stationary agents are considered, residing in managed nodes and exhibiting autonomous, adaptable monitoring behaviour based on the performance information gathered. The agents use the KQML [KQML] agent communication language to wrap existing monitoring and measurement tools, allowing agents to collaborate via a common agent protocol. The approach was chosen to address a number of performance management requirements of large scale fixed networks. Static intelligent agents were used by [Vayi00] in order to provide flexible ATM resource management capabilities. The proposed architecture is based on intelligent collaborating agents located in the service and network provider domains. Every time there is a service request, the intelligent agents of a service provider inform the agents of a number of network providers. The latter gather a report of the current availability of resources along with charging information and place a bid to the service provider. In this way, the proposal promotes competition between network providers and allows the service provider to intelligently select the most suitable offering. The network provider selected further relies on intelligent agents in order to dynamically configure QoS parameters according to specific service needs.

3.2.4.2 Configuration Management

Configuration management involves a number of requirements mainly regarding the modification of a number of attributes of network nodes, as well as the installation of any required software. Typically, a network manager has to perform such tasks manually. Instead, an architecture has been proposed by Raza et al [Raza98] that exploits mobile agents in order to allow for “plug-n-play” style capabilities of network nodes.

Cheikhrouhou et al [Chei00b] has demonstrated an agent-based approach to the automatic provision of Permanent Virtual Channels (PVC) in ATM networks. This work presents agents able to acquire new capabilities without interrupting their operation, in order to dynamically upgrade the functionality of an ATM switch. This task does not involve a migrating agent but a static agent that supports interfaces that allow the download of management modules. Feng et al [Feng01] have also studied the issue of uninterrupted software upgrades and propose a framework for the dynamic replacement of management modules. In contrast, these proposals involve migrating agents equipped with the updated modules, received by a software swapping object.

3.2.4.3 Fault Management

In fault management a key aspect is fault detection, typically a process of building a specialised model of a network. Mobile agents [Elda98] can decentralise this process by locally checking for over-utilization of resources, required for the creation of a model of over-utilised, error prone nodes. If the constraints used describe violations of what is considered normal behaviour of a network element then the agent performs a fault detection function. A mobile agent can also access other local resources of a node and use them to perform enhanced tests or execute a recovery routine. The constraints used for this task can involve complex fault detection and correlation algorithms.

El Darieby, et al [Elda99], has demonstrated the use of intelligent mobile agents to automate fault management tasks. The system involves a mobile agent that visits a list of nodes that recently triggered fault alarms. There, it autonomously diagnoses the alarms and determines the cause of faults using a built-in expert system. Finally the agent returns to its origin for reporting.

Thottan, et al [Thot99], has presented a system for the prediction of network failures at the router level, achieved using a mobile agent with sensor capabilities. The agent is deployed near a managed router, obtains relevant MIB data and provides temporally and spatially correlated predictive alarms. The time correlated abnormal changes in the individual MIB variables are spatially correlated using a combining scheme. When tested the agent successfully predicted seven out of nine faults with a prediction time in the order of minutes.

Another proposed approach is based on societies of small, biologically inspired and relatively simple agents that need to cooperate to deliver the intelligence needed for the diagnosis of network faults [Whit98a]. A number of types of such tiny agents are injected in the network, each addressing one aspect of the problem and using observations to confirm or disprove a specific hypothesis. The solution to the problem emerges through the integration of the hypotheses results of each agent type [Whit98b][Whit98c].

3.2.5 Mobile Agent Platform Evaluations

Recent research has produced a number of thorough evaluations of modern mobile agent platforms highlighting their relative advantages and suitability. Altmann et al [Altm01] evaluated 12 current mobile agent platforms in terms of security features, MASIF and FIPA standards compliance, development efficiency, runtime stability and network performance. The best all-round performer was found to be the Grasshopper mobile agent platform, followed by Jumping Beans, Aglets and Voyager. While this comparison concentrates more on features and places a small weight on performance criteria, the work of Silva et al [Silv00] focuses on performance overheads associated with a number of typical operations. Eight mobile agent platforms were put

into tests and measurements of the elapsed time and network traffic were taken. The best performing platform was found to be James [James], while Grasshopper produced average performance results.

In addition to these evaluations the author of this thesis has worked twice within a collaborative team of researchers working on the evaluation and selection of a mobile agent platform suitable for management tasks. These efforts took place within the contexts of the MIAMI [MIAMI] and MANTRIP [MANT] projects, with all evaluation results presented in [Guth98] and [Mich00] respectively. In both cases the Grasshopper platform was selected as the best all-round solution. Grasshopper offers average performance overheads, it is MASIF and FIPA compliant, offers a stable runtime environment, and well-designed agent APIs.

Below we present some mobile agent-platform information gathered during the selection stage of the MIAMI project [Guth98] towards which the author actively contributed and participated.

- **April:** April (Agent PRocess Interaction Language) [APRIL] is a process-oriented language especially designed for implementing intelligent network applications. April is also the name of the corresponding execution platform. April was developed at Imperial College, London as part of the ESPRIT project Imagine. The April system is written in C and the agents are programmed in the April language. The agent transport and communication is based on a proprietary protocol based on TCP/IP.
- **Aglets Software Developer Kit:** The Aglets Software Developer Kit (ASDK) [AGLETS] was developed at the IBM Research Laboratory in Japan. The first version was released in 1996. Since April 1998 the ASDK version 1.0 is available as a first formal release of the platform with an IBM International License Agreement for Non-Warranted Programs. The ASDK is entirely written in Java. The Aglets transport and communication is based on the proprietary Agent Transfer Protocol (ATP), which is modeled over the HTTP protocol. The protocol capabilities are accessed through the Agent Transfer and Communication Interface (ATCI), which allows an abstraction from the underlying transport protocol.
- **D'Agents:** The D'Agents platform [DAGE] is a modified version of the Tool Command Language interpreter (TCL), which aims at providing a basic support for mobility. This support is achieved via a modification of the TCL kernel which offers a command to freeze the execution of a TCL script, catch the execution state, move it to another computer, and then resume the execution at the next instruction of the script. The platform is written in C and the agents are written in TCL. For transport and communication purposes D'Agents uses a proprietary protocol over TCP/IP.

- **Grasshopper:** Grasshopper [GRAS] is the first available mobile agent platform, which is compliant to the MASIF and FIPA standards. It is being continuously developed since 1997 by IKV++ and offered as a commercial product or with a free license for non-commercial/educational use. The entire Grasshopper platform is implemented in Java. Grasshopper supports a number of transport and communication protocols. Along with the default proprietary protocol based on TCP/IP, we have support for RMI/JRMP as well as CORBA/IIOP.
- **Voyager:** Voyager [VOYA] is a Java-based and agent-enhanced Object Request Broker (ORB) developed by Recursion Software Inc. Voyager is a commercial product although a free license allows non-commercial use of the Voyager core technology. Voyager is entirely programmed in Java. The agent transport and communication is based on a proprietary ORB on top of TCP/IP.

3.2.6 Mobile Agent Systems Evaluations

A number of experimental measurements available deal with an evaluation of a mobile agent approach in comparison to SNMP remote polling. The results presented in [Saha99] [Isma99] [Zapf99] showed that mobile agents can achieve significant performance improvements over SNMP polling. Despite this, the results in [Saha99] show that there are boundaries to the performance improvements of agents relating to the number of managed objects filtered as well as the number of network nodes for which a single agent is responsible to migrate in sequence and perform its task. This work was furthered by Gavalas et al [Gava99a] by providing additional measurement results in favour of mobile agents as compared to SNMP polling and examining the optimum number of network elements that are most efficiently the responsibility of single mobile agent. Unfortunately evaluations by [Gava99a][Saha99] wrongly involve the migration of an agent to the network management station just for the purpose of delivering a report (i.e. mobile agent overheads reported would actually be less if they were properly applied). Furthermore, all these evaluations are limited to a comparison of mobile agents with SNMP, neglecting emerging distributed objects and mobile code approaches for network management. More recently Glitho et al [Glit02] provided an experimental evaluation of a client/server and mobile agent approach to distributed information retrieval. The measurement results lead to the conclusion that the mobile agent approach offers better performance, although this performance improvement may not be necessary and a cleverly designed client/server approach may be well sufficient. In addition, Simoes et al [Simo02] has presented a thorough evaluation of different mobile agent models of distribution (including Constrained Mobility introduced in [Boho00a]). The results showed that for many cases, performance gains are determined mainly by distribution rather than locality (i.e. local client/server transactions), which only provides marginal improvements. The measurement

results also showed that from a performance perspective data compression is hardly relevant under good network conditions. Finally, this evaluation showed that constrained mobility offers better performance and reduced traffic compared to multi-hop agent migration.

In another direction researchers have produced performance evaluations based on analytical or simulated results that aim to highlight performance issues of mobile agents, often at large scale. Pioneering work in this direction was presented in [Bald97] and [Picc98] comparing paradigms for mobile code with non-autonomous, so called 'agents', reflecting on software programs with migrating capabilities. Management tasks carried out on a number of network elements involving multiple remote interactions have been examined. Analytical models of the overall traffic and the traffic overheads affecting the network management station were computed for the client/server, Code on Demand, Remote Evaluation and MA paradigms. In [Rubi99] a simulation-based comparison of mobile agents and SNMP showed the potential of mobile agents to offer a significant performance improvement. This was achieved when the number of managed network elements ranged between two identified limits. A lower limit, involving the number of messages that pass through backbone links and an upper limit related to the incremental size of a mobile agent. In parallel, Liotta et al. [Liot99] presented an analytical evaluation of mobile agent-based monitoring systems for a combination of schemes based on monitoring models, mobile agent configuration and deployment patterns. The modeling and analysis reported in this paper are restricted to the case of hierarchical networks such as telecommunications networks. Results on traffic show that mobile agents typically offer improved performance, although they do not always scale better than remote polling. Nevertheless, in very-small scale monitoring problems and for relatively short monitoring tasks, centralized polling tends to be preferable. Moreover, results on delay demonstrate that mobile agent solutions improve both scalability and performance in most circumstances. This work later evolved with a variety of simulation-based results presented in [Liot01a]. These showed that a system based on mobile agents out-performs remote polling in many typical situations. It has also helped to identify conditions in which agents cannot be recommended. Mobile agents were particularly effective in reducing the communication and processing bottleneck located at the monitoring station, while their weakness was mainly related to migration overheads.

3.3 Conclusions

During the early to mid 90's theoretical studies on the possible models of deployment of mobile code provided an important first foundation for research on software mobility. Further work on code mobility, mainly in the form of the Management by Delegation proposals, showed the potential role of software mobility in network management. An agent-based approach adds the

important attribute of autonomy alongside with mobility, opening new opportunities for research. In this direction, a number of proposals exploiting agent mobility and autonomy have been devised to address the requirements of particular network management scenarios. Despite this, in the area of network management we have seen a lack of clear direction of exploitation of mobile agents, evident through the heterogeneity of technical solutions proposed and the use of inconsistent terminology. The chapter that follows attempts to clarify the potential role of mobile agents in the area, with the proposal of strategies for mobile agent-based network management, addressing typical network management requirements.

PART II – THESIS CONTRIBUTIONS

Chapter 4

4 Agent Mobility and Autonomy

The chapter presents a theoretical study on agent mobility strategies explaining their usefulness for performance management and their role in the wider network management context. A detailed description of the approach and usefulness of the proposed 'Constrained' agent migration strategy aims to highlight the differences and added value compared to similar models for mobile code. Descriptions of additional agent migration strategies present the enhancements over previous proposals placing them in a specific context of practical exploitation by network management systems. The author's work on mobility is complemented by a study on autonomous behaviour that highlights the capabilities of mobile agents further from mobile objects and code and in the direction of providing 'intelligent', self-configurable and fault-tolerant network management systems.

4.1 Constrained Mobility

The advent of mobile agent technologies provides ground for the evolution of one of the most elementary forms of code mobility defined as Remote Evaluation (REV), as we have described earlier in chapter 3. In REV an application in the client role can dynamically enhance the server's capabilities by sending code to the server. Subsequently, clients can remotely initiate the execution of this code that is allowed to access the resources collocated within the server. Therefore, this approach can be seen as an extension of the client-server paradigm whereby a client in addition to the name of the service requested and the input parameters can also send the code that implements the service. Hence the client owns the code needed to perform a service, while the server offers both the computational resources required to execute the service and access to its local resources.

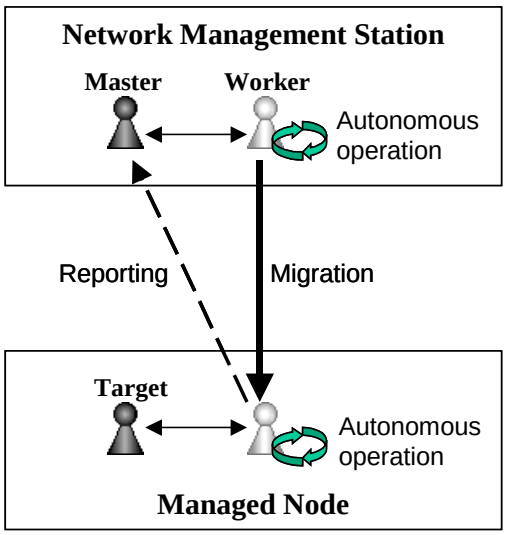


Figure 4-1: Constrained mobility for mobile agent-based network management

A natural evolution of the REV model involves the migration of an entity not restrained to be a remote service but which can also act as a fully autonomous software agent. Unlike mobile code or objects, an autonomous mobile agent can decide by itself the appropriate node of migration or autonomously equip itself with the appropriate network management functionality modules before migrating. Continuing its operation at the managed node the agent further demonstrates its autonomous capabilities in the form of autonomous management operation, intelligent adaptation to the environment and intelligent collaboration, as we will discuss in more detail later in this chapter. We term this strategy 'Constrained' mobility since an agent, upon its creation at a client site, intends to migrate to a single network element where its execution is confined [Boho00c][Boho00b][Boho00a]. In contrast with previous work on general migration models for mobile code, constrained mobility is a strategy specific to mobile agent-based network management, involving a minimum of three agents and incorporating single-hop mobility as well as autonomous operation. In its simplest form, constrained mobility involves one mobile agent and two stationary agents with the following roles (Figure 4-1):

- Master Agent: Stationary agent that initiates a management task and handles interactions with the user or client application.
- Worker Agent: A mobile agent initiated by the Master and equipped with the required management logic, migrating and executing its task in the appropriate remote node. Any reports or notifications generated are remotely transferred from the Worker to the Master agent.
- Target Agent: Stationary agent that pre-exists at the targeted network element, allowing Workers to access a number of required network element resources.

The single-hop migration involved in constrained mobility means that any concerns on mobile agent migration overheads are minimized. Constrained mobility is not suitable for short-term repetitive tasks as a frequently repeated single-hop agent migration followed by autonomous execution and termination can result in significant migration overheads. The performance concerns relating to the single-hop migration of constrained mobility become less significant as the time spent by the agent in the targeted network element increases. Ideally, for many network management tasks we would like to send autonomous agents in a managed network element and leave them to execute over a period of hours, days or even weeks. The system autonomously handles any problems, or changes in conditions encountered so that the need for user intervention is minimized. In addition we would like a network management system that is programmable, in the sense that when network management functionality is updated or requirements change, the system can program itself by replacing the previously executing agent with the updated or customized version. This is exactly the scenario that agent constrained mobility serves best.

In summary, constrained mobility is suited for long-term network management tasks for which programmability and autonomy are desirable. Such tasks are particularly common in network management (e.g. performance monitoring, fault correlation, intrusion detection, etc.) and thus constrained mobility serves a particularly important role. The following section presents the application of constrained mobility for mobile agent-based performance monitoring.

4.1.1 A System for Performance Monitoring

The performance monitoring system developed monitors the edge nodes involved in a user's connectivity path, following an approach that separates the management logic from network technology specific modules that are loaded dynamically, as and when required. The functionality for performance monitoring we developed follows a generic design approach to constrained mobility involving the three agent roles as presented below (see Figure 4-2):

- Master Agent: Stationary agent responsible for the interactions between the user and the system. The Master interacts in both directions with the user, initialising and controlling the performance monitoring process and sending performance monitoring notifications and reports that may trigger an adaptation action.
- Worker Agent: A mobile agent autonomously equipped with the required management logic and supporting context appropriate to the underlying network technology. Following a constrained mobility strategy the agent subsequently migrates to the targeted node for execution. The Worker agent for performance monitoring creates a number of monitors of required performance parameters and migrates with them to the targeted network element to perform its task.

- Target Agent: Stationary agent at the targeted network element allowing the monitors to access a number of required resources. The Target allows passive and active performance measurements. For passive measurements the Target 'wraps' the underlying network technology allowing access to 'raw' resources of the network element (for measurements of used bandwidth, loss, etc). For active measurements the Target provides an "echo" facility that remotely returns a test stream to the sender upon which a measurement can be taken (for measurements of delay, jitter, etc).

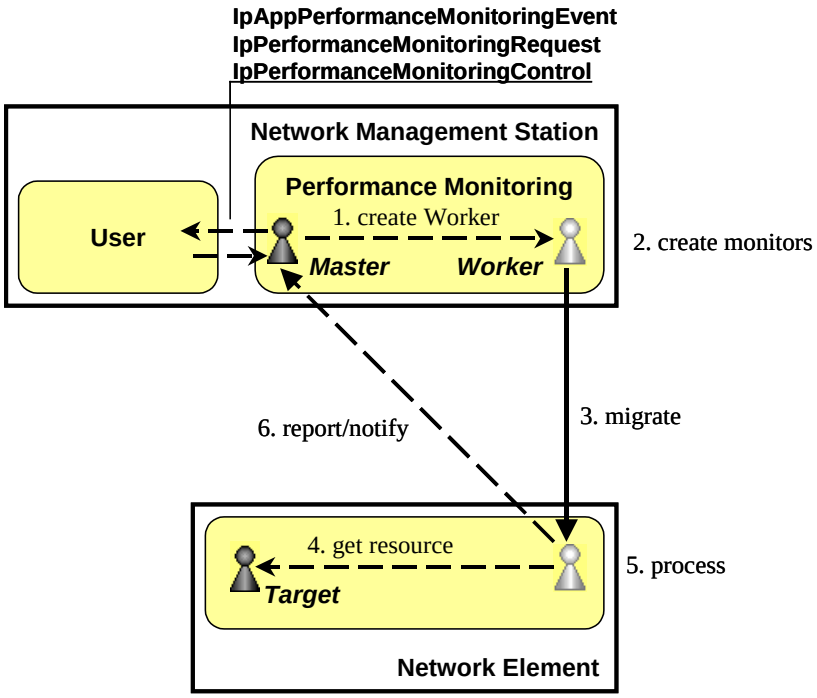


Figure 4-2: A performance monitoring system following a constrained mobility strategy

The user of this system controls the performance monitoring operation and receives network performance events in a generic manner. These external interfaces and data types involved in the communication with the user are compliant with the design guidelines of the Parlay group (see Figure 4-3). This was done as a complement to current Parlay work (Connectivity Manager APIs version 2.1, [PARLAY]), which although still in progress, enjoys a strong industry support and has been selected by 3GPP as the basis for their OSA APIs. In order to gain wider acceptance it is important for any network management component dealing with different networks or services to provide a generic set of interfaces that are in line with current standardization efforts.

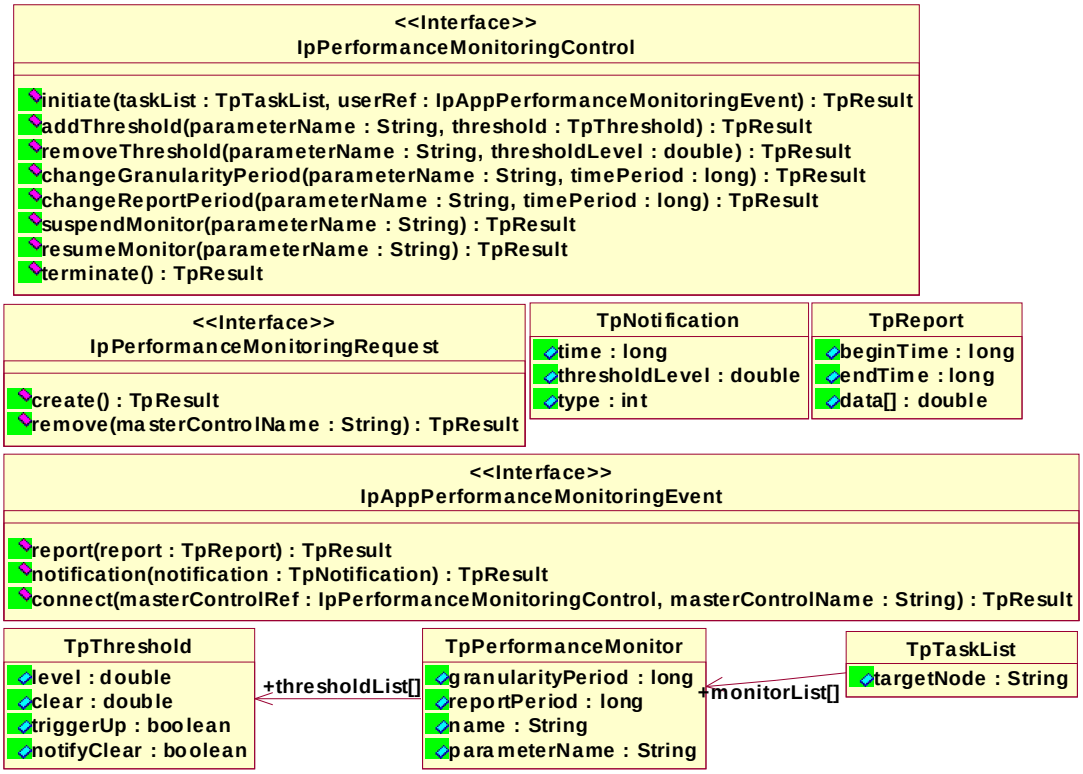


Figure 4-3: Parlay compliant interfaces for performance monitoring

Using the API shown in Figure 4-3, a user request for performance monitoring of a node is initially passed to the Master agent, which in turn creates a suitable mobile Worker agent. The Worker agent, based on the given performance parameters, loads the associated Java class files at runtime. Upon completion of this process, the Worker migrates to the targeted node where it contacts the Target agent that pre-exists in the node and gives access to 'raw' performance information. The Worker initiates the operation of a number of monitors each responsible for the monitoring of a single performance parameter and running on a dedicated execution thread. Each monitor initiates its task with periodic requests for 'raw' performance information (typically involving counter type values) provided by the Target. Based on this information a monitor performs metric monitoring for a number of performance parameters (e.g. Used Bandwidth, Loss, Delay, Jitter, etc), checks the thresholds set and gathers the information produced in order to generate reports. When a monitor generates a performance event (report or notification) this is passed from the Worker remotely to the Master agent and finally to the user or client component. The remote reports of the results gathered are generated on a scheduled basis (e.g. every 30 minutes) while notifications are sent on the fly every time a performance threshold is triggered.

The functionality included in the monitors carried by the Worker agent is based on the Metric Monitoring and Summarization Open Systems Interconnection (OSI) Systems Management Functions (SMFs) (see [X738][X739] and section 2.4.1). While such performance monitoring

functionality is fixed in OSI Systems Management (OSI-SM), the logic included inside the Worker may be customized by the user e.g. to provide a different model for triggering notifications, based on semantic knowledge of the monitored resources. The important customisation aspects are described next.

4.1.2 Customization of the Management Logic

Network elements tend to provide a set of standardized as well as proprietary objects that can be accessed remotely to support management functionality. These are fixed and cannot be altered or extended. In that sense a network element can be characterized as a black box with pre-programmed management capabilities. With the evolution of the telecommunications industry, we see today support for distributed object architectures at the network element level (e.g. [CEMF]) and in the future, eventually, support for mobile agent technologies will also be included. The key benefit is that mobile agents implementing customized functionality could migrate and execute there, augmenting dynamically the elements' capabilities. We examine here how the functionality of a mobile agent can be easily customized in this system.

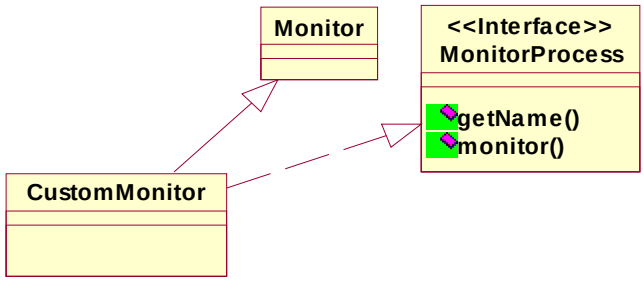


Figure 4-4: Allowing programmable performance monitoring logic

Customized functionality can be provided in an object that inherits the functionality of the standard Monitor provided in the system. In Figure 4-4 we see an example CustomMonitor class that can extend the standard monitoring functionality by inheriting from the standard Monitor class provided by the system. In addition by implementing the MonitorProcess interface as shown in this figure the CustomMonitor can also override the monitoring behaviour of the standard monitor in order to introduce its own customized approach.

The standard monitor provided by the system already supports autonomous customisation as any performance parameter involved is selected and loaded dynamically at runtime. This allows the monitor to keep a generic view of its network management task and use a suitable performance parameter implementing a customized measurement approach reflecting on a specific network technology or different measurement methodologies. In this system new performance parameters can be written by extending the PerformanceParameter class as shown in the examples of Figure 4-5. The system does not need previous knowledge of any performance parameter modules, but

instead any of these can be loaded dynamically using capabilities of the JavaBeans component framework [JAVAB].

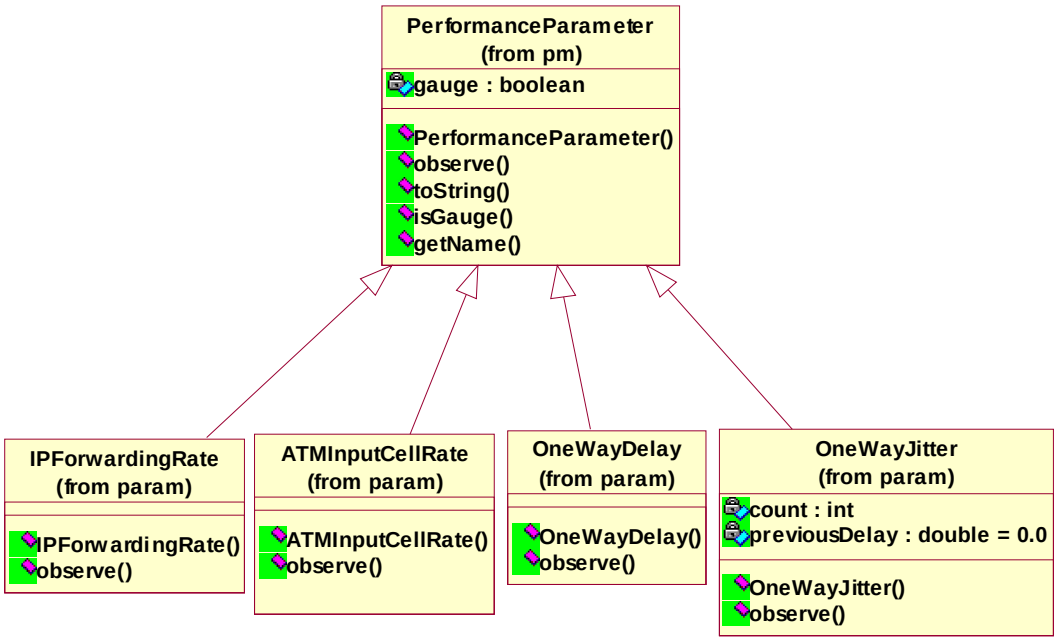


Figure 4-5: Context specific performance parameters

4.2 Weak Mobility and Strong Mobility

The study on multi-hop agent mobility based on network management requirements, previous work on mobile code and an analysis of mobile agent attributes resulted in refined 'Weak' and 'Strong' mobility definitions [Boho00b]. Agent-based network management following a weak and strong mobility strategy involves an agent that migrates into a number of targeted nodes for autonomous execution. The important difference between the two is that in strong mobility the agent carries with it state information that can influence its current or future behaviour. Figure 4-6 depicts the sequence of operation and agent configuration involved, in terms of the abstract Master, Worker and Target agent roles, as described earlier. These strategies for mobile agent-based network management present some important differences from the general categorizations of weak and strong mobility of mobile code as presented by Fuggetta et al in [Fugg97]. In this earlier work, models of software mobility relate to 'agents' that are merely software components with the ability to move between different execution environments. This is in contrast with the viewpoint of this thesis and the intelligent distributed systems community, considering the difference between an agent and an object in terms of autonomous operation.

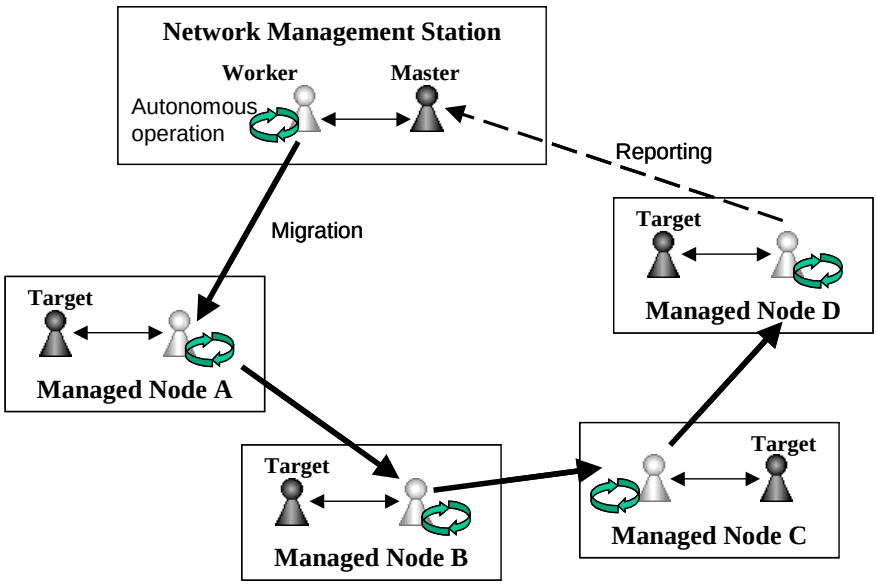


Figure 4-6: Weak and strong mobility for mobile agent-based network management

Another important difference with the views of this thesis is that, Fuggetta et al bind strong mobility with a transferred state that reflects the run-time image of a component transferred as a whole, including its execution state (i.e., program counter, call stack, etc.). There is one important problem in the context of network management associated with this view. The transfer of such a run-time image results in a very large migration overhead, not suited for network management systems that in principle should perform their task with the minimum possible impact on the managed network. The increased low level complexity and large performance costs have in fact led popular mobile code languages such as Java and the .NET to refrain from supporting strong mobility in this manner. Despite this, the effect resulting from strong mobility, (i.e. involving agents that migrate and exhibit behaviour based on previous state) can be efficiently supported through the use of class attributes reflecting the agent state. Such attributes, preserved between migrations, designate the agent state required for associated actions to be followed.

In another direction, proposals appearing in recent research work (e.g. PhD thesis by Gavalas [Gava01], [Saha99], etc) describe an operational model involving an agent that returns to the network management station for reporting and termination, as shown in the left part of Figure 4-7. This is in contrast with the view of this thesis that an agent should only migrate if a task needs to be executed at its destination (and the logic to perform the task is not already there of course). In the author's view, an agent should never migrate in order to deliver a report but unfortunately this is a frequent mistake made by designers/developers of mobile agent systems. In the approach proposed by this thesis and shown in the right part of Figure 4-7, the agent could save the overheads of this migration by remotely sending reports to the network management station. In addition, for repetitive tasks the agent can continue its operation by visiting the nodes backwards, performing a task or collecting information and finally sending a remote report at its last visit. The

operation can be repeated as many times until a termination decision is made (i.e. either autonomously or through manager intervention).

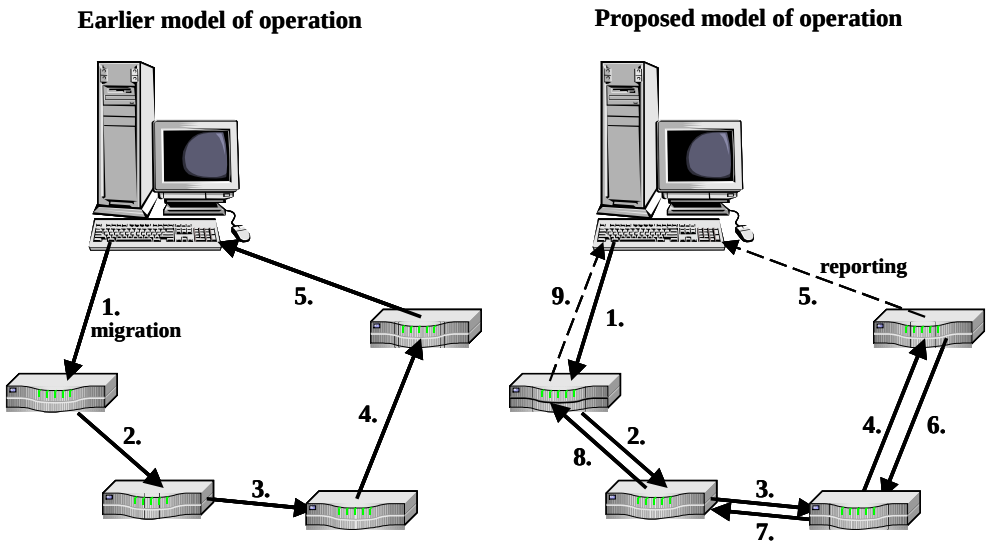


Figure 4-7: Differences in operational modes relating to earlier approaches proposals

In order to deploy weak or strong mobility efficiently the network management tasks executed at each node should require a short time to complete in order to ensure responsiveness. This is important as in weak and strong mobility a single cycle of the system's operation typically involves the execution in all of the intended nodes. Weak and strong mobility are thus suited for short-term tasks, for which programmability and autonomy are desired. Due to the multiple migrations involved, the size of a mobile agent operating in this manner should be kept small.

Some theoretical work and simulation results on multi-hop agent migration relating to future applications and environments have shown positive results for the use of multi-hop mobility, as shown in the thesis of Liotta [Liot01b]. Further to this and despite efforts from the author of this thesis and recent research, the results on the practical impact of weak and strong mobility on current and emerging network management scenarios have not been particularly encouraging. For instance, let's consider a typical short-term task of QoS configuration of network nodes involved in a user's connectivity path. The logic to perform this task is suitably operating in a static manner, residing in each network node. A remote request for QoS configuration arriving at each node is much more efficient than any migration overheads incurred by even the smallest agent. Further, if programmability is required for this QoS configuration logic, constrained mobility can be suitably deployed autonomously by the system re-configuring or updating itself.

4.3 Autonomous Agent Behavior

Compared to mobile code and objects, autonomy represents an important attribute introduced by mobile agents. The field of artificial intelligence has produced a large number of alternative approaches introducing 'intelligent' software behaviour (e.g. neural networks, genetic algorithms, swarm intelligence, etc). No single approach has yet managed to separate from the rest; some can tackle a specific problem better, although in many cases a relative advantage may not be obvious. The same situation applies for agent autonomy, influenced by the same approaches and hurdles of artificial intelligence. Traditional A.I. techniques can easily result in large, processing intensive and complex programs that we often want to avoid in the network environment within which software agents reside. In these lines pioneering researchers of software agents (e.g. [Lash94]) accurately characterized agents as "semi-intelligent" entities, exhibiting only a 'weighted' level of computational intelligence, aiming to keep performance overheads within reasonable levels.

Each agent of a network management system can exhibit autonomous behaviour in the form of autonomous management operation, intelligent adaptation to the environment and intelligent cooperation. We describe these in detail and for the context of performance management in the sections that follow. Within this context and scope the author has successfully applied an approach that follows, as appropriate, the three forms of autonomous behaviour in combination with a constrained agent mobility strategy. This has resulted in performance management systems with distinguishable benefits compared to mobile code and mobile object approaches, as we will present in more detail later in chapter 5.

4.3.1 Autonomous Management Operation

This involves autonomous operations applied to a management task combined with the decision logic that triggers them. These are tightly coupled with a specific network management task (e.g. pro-active threshold notifications in performance management, autonomous (re-) configuration of the parameters of a specific management task, etc). In the following sections we describe such autonomous operations in the context of performance monitoring.

4.3.1.1 Initial Task Configuration

A performance management system often serves client applications with a role that focuses on service management issues. Such client components have limited capabilities of assisting the initial configuration process of performance management. Ideally, in such cases a performance management component should have the autonomous logic to configure most of its initialisation parameters. The system presented in this thesis exhibits this autonomous behaviour based on the following required external input:

- The required QoS provided by the client component: This is a high level indicator of QoS specified according to 3GPP's 'conversational', 'streaming', 'interactive' or 'background' service QoS classes. This input is the only requirement that performance management places on the client and it is crucial for the appropriate configuration of network monitors. As the client typically focuses on service management tasks depending on network QoS, it is natural to be already aware of Service QoS information. In this sense the requirement of providing the service QoS to performance management is not an additional burden to the client.
- Network Type and available resources: Different network technologies require different modules of performance parameters implemented to access technology specific 'raw' resources and perform simple calculations producing information that is further processed generically by the metric monitoring logic. The performance management component separates specific and generic parts of its functionality so that the same generic part can load different specific modules as required and cooperate with them. In addition, some configuration management information should be obtained in order to configure appropriately the metric monitoring process of the selected performance parameters. For example if the traffic utilization on a network interface needs to be monitored, then the interface speed needs to be known in order to set appropriate thresholds.

4.3.1.2 Proactive Monitoring Behaviour

There exist some scenarios where a client of performance management would like to be notified of a tendency of network performance before the critical event of a threshold crossing occurs. For instance, a video transfer service using the performance monitoring capabilities may want to be pro-actively and gradually informed of deteriorating network conditions in order to smoothly adapt the quality of the transmitted video (e.g. by changing coding aspects of the video transmission). In order to provide such a capability, the performance management component is equipped with autonomous logic that identifies performance tendencies that designate a likely imminent threshold crossing. If such a case is identified, a special pro-active notification is sent to the client notifying of the event detected as well as the proximity to the threshold level. In addition, when eventually the threshold level is exceeded, the system will autonomously re-configure a new threshold in a way that the current traffic levels are always bound within an upper and a lower threshold.

4.3.1.3 Selection of Migration Destination

The performance management system has the responsibility to instruct the appropriate migration of an agent equipped with management logic. The decision is based on performance and

configuration management information. Initially, performance management needs to consider all known hosts in a sub-network equipped with a suitable agent execution environment. Today, network elements do not yet provide support for an agent execution environment but a network of interconnected workstations, forming a Data Communication Network (DCN) in TMN terms, can be conveniently set-up for this purpose along with the managed network. Given the knowledge of a number of alternative nodes to migrate, the selection of the most suitable node is made based on a comparison of performance. The most efficient node may be selected as a result of a number of parameters such as delay between this and the managed network element, memory utilization, number of agents currently executing, etc.

4.3.2 Intelligent Adaptation to the Environment

An adaptable and fault tolerant agent should take into account the dynamic environment within which it operates and autonomously deal with changing conditions (e.g. remote communication failures, required updates of management modules, unavailability of collaborating agents, etc). In such critical circumstances, a typical object-based network management system is highly dependent on the intervention of a network administrator or client application. An agent-based system on the other hand should attempt to resolve any problem by itself and be prepared to take fallback measures. The author has created the taxonomy presented below in order to analyse the issue in a generic manner that can be applied to different network management contexts.

An agent-based network management system contains software entities that depend on an environment that consists of the following:

- Collaborators: Local software entities with which collaboration is required in order to achieve a management task.
- Correspondents: Remote software entities with which communication of information is required.
- User: The user of the management system, either a person or a client application.
- Managed Nodes: The network nodes associated with a connectivity path and require some management tasks. They provide the computational and management resources required.

For any agent involved in a network management task, there exist 6 basic environment-related requirements as described below:

- Proximity: Management logic should execute as close to the required resources as possible, ideally logic and resources should be co-located.

- Availability of Collaborator: A management agent typically depends on the collaboration with other agents or objects.
- Availability of Correspondent: The operation of a management agent typically involves remote communication with other agents or objects.
- Resource Availability: Management logic depends on the availability of a required resource (fixed within a specific network element).
- Availability of Logic: A management agent crucially depends on a module containing the required management logic.
- Availability of Operational Resources: This includes any resources at a network node required for the execution of an agent task.

A goal-oriented, adaptable, autonomous and fault tolerant agent should take care of addressing all these requirements taking also into account that in a dynamic environment some requirements may not be met. In such a case a normal program would report failure and terminate, the user should rectify any problems and restart the system with the expected conditions. An agent system on the other hand should attempt to resolve any problem by itself, find a fallback solution and give up only when the task is impossible within reason. There are a set of generic problems, solutions, fall back solutions and dead ends that should apply for the management agents of a number of network management systems. These are examined below, in relation to the 6 requirements presented:

Proximity:

1. **Migration Denied:**
- *Solution:* Migrate to a nearby alternative node.
- *Fallback:* Operate from node of creation.
- *Side effect:* Inform managing entity, periodically re-attempt.

2. **Runtime Environment Unavailable:**
- *Solution:* Migrate to a nearby alternative node.
- *Fallback:* Operate from node of origin.
- *Side effect:* Inform managing entity, periodically re-attempt.

3. **Network Node Inaccessible:**
- *Solution:* None.
- *Fallback:* None.
- *Side effect:* Inform managing entity, periodically re-attempt.

Availability of Collaborator:

1. **Collaborator Unavailable:**
   - *Solution:* Request for a collaborator (creation and migration of a suitable collaborator can follow).
   - *Fallback:* None.
   - *Side effect:* Inform managing entity.

Availability of Correspondent:

1. **Correspondent Unavailable:**
   - *Solution:* Request for a correspondent (creation and migration of a suitable correspondent can follow).
   - *Fallback:* None.
   - *Side effect:* Inform user.
2. **Remote Communication Failure:**
   - *Solution:* None.
   - *Fallback:* Continue operation, retain information intended for communication.
   - *Side effect:* Periodically re-attempt.

Resource Availability:

1. **Resource Unavailable:**
   - *Solution:* None.
   - *Fallback:* None.
   - *Side effect:* Inform managing entity, periodically re-attempt.

Availability of Logic:

1. **Logic Unavailable:**
   - *Solution:* Request for management logic modules.
   - *Fallback:* None.
   - *Side effect:* Inform managing entity.
2. **Logic Update Required:**
   - *Solution:* Suspend and request for management logic modules.
   - *Fallback:* Continue operation as is.
   - *Side effect:* Inform managing entity, periodically re-attempt.

Availability of Operational Resources:

1. **Critical Resource Utilization:**
   - *Solution:* Release some resources.

- *Fallback:* Migrate to a nearby alternative node.
- *Side effect:* Inform managing entity, if fall back is followed periodically check resources for a migration back.

We now examine how these required adaptation capabilities could be incorporated within a network management agent. A useful approach to follow has been presented by Maes [Maes91] and is based on the definition of an agent as a set of modules with discrete responsibilities, termed 'competence modules'. Let's consider a Worker agent responsible for a certain network management task. Such an agent may consist of three competence modules:

- A Carrier Module: Responsible for the initial migration or any subsequent re-location of the agent.
- A Management Module: Involves the conventional and autonomous aspects of a specific network management task.
- An Environment Module: In collaboration with the two other modules takes decisions to adapt within a given environment and also executes the related actions (this is the module handling the solutions, fall backs and side effects mentioned above). The actions taken are based on the analysis of accordingly updated agent state. For instance, an agent attribute reflecting state and represented as an array of constants at some point may contain the values:

  LOGIC_UPDATE_REQUIRED,

  MONITORS_SUSPENDED,

  WAITING_UPDATE_RESULT

  The updated management logic module is eventually sent to the Worker and it is subsequently initialised accordingly so that the monitor's operation is resumed. At this point the previous state values are removed and replaced (e.g. with MONITORS_RUNNING).

4.3.3 Intelligent Collaboration

Involves the use of an Agent Communication Language (ACL) (e.g. the FIPA ACL [FIPA]) for the collaboration between agents. The main advantage of the approach involves the use of communication messages to decouple agents from interface dependencies that can be limiting and ineffective, especially when complex processing of dynamically changing data is involved (e.g. when contracts are negotiated). Within the performance monitoring system presented in this thesis there were only simple collaboration requirements between the Master, Worker and Target agents.

In this respect this benefit is not highlighted in the system and the functionality can be realised effectively by using carefully defined interfaces of communication.

4.4 Conclusions

This chapter presented the strategies of constrained, weak and strong mobility specifically for mobile agent-based network management. For each case, we presented the mode of operation, suitability and associated benefits. The difference between the strategies considered in this thesis and earlier proposals mainly relating to mobile code was also identified. Most importantly, we assert that by deploying an agent constrained mobility strategy we can suitably and efficiently address the requirements of the large majority of network management tasks for programmable and autonomous operation. The thesis proposals on agent autonomous behaviour are an important complement to our mobility strategies. The approaches described provide a ‘weighted’ level of computational intelligence aiming for autonomous behavior with acceptable performance overheads. The thesis proposals on autonomous mobile agents for performance monitoring following a constrained mobility strategy are examined and assessed through practical research work, presented in chapter 5 that follows.

Chapter 5

5 Case Studies and Technology Assessment

This chapter builds upon the basis of chapter 4 with descriptions of three case studies that allowed a detailed practical evaluation of the usefulness of mobile agents. The three case studies presented are titled “ATM-based Active VPN management”, “QoS Configuration and SLA Auditing of IP-DiffServ Networks” and “QoS Management for the Virtual Home Environment”. The descriptions provided cover the role of performance management within the specific environment, the mobile agent approach chosen along with the benefits and drawbacks of the use of mobile agents in the system. The author of this thesis suitably designed and implemented the performance management capabilities required in each of the three case studies. The resulting work is thoroughly assessed through a number of experiments and measurements that highlight the performance overheads of mobile agents as compared to distributed objects and mobile code approaches.

5.1 ATM-based Active VPN management

This case study involves a mobile agent based system for “active” management of a VPN. The client of this VPN (see Figure 5-1) is a group of collaborating Service and Content Providers forming a Virtual Enterprise (VE). The VE makes use of the VPN offerings of a VPN provider termed Active Virtual Pipe Provider (AVPP) with a business role similar to a Telecommunications Information Networking Architecture (TINA) Retailer. The AVPP itself makes use of the connectivity services provided by a number of Network Providers (NPs). The NP domain of the case study comprises configuration, performance and fault management functionality.

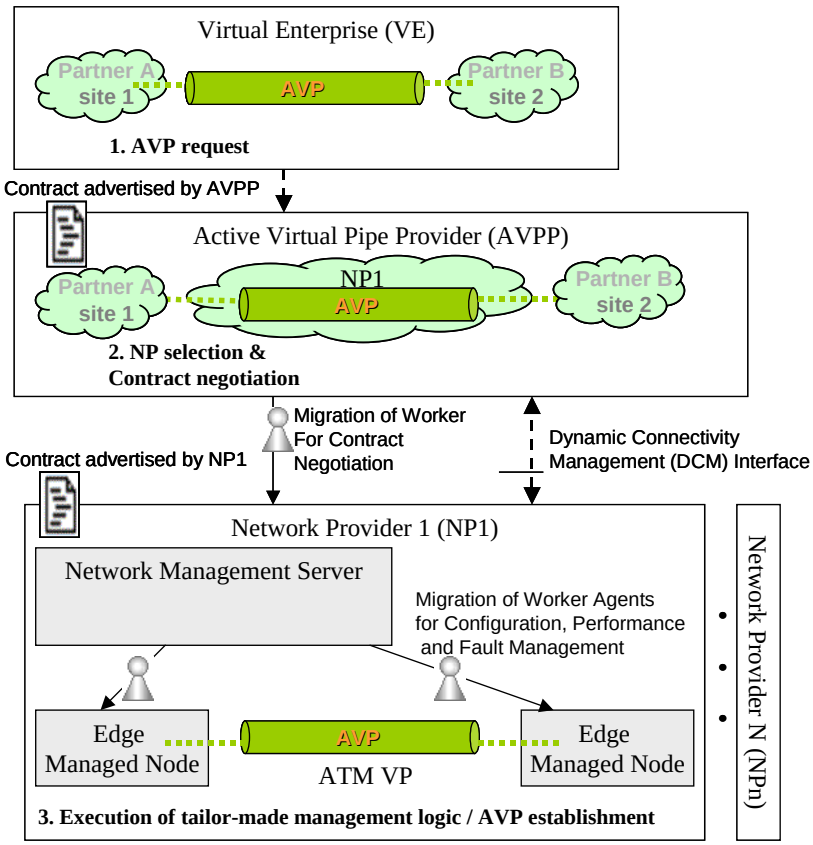


Figure 5-1: Creation of an Active Virtual Pipe (AVP)

The VPN provided by the AVPP is not static as it would typically be the case today. Instead, it exploits mobile agents to provide a programmable VPN, tailor made to the changing needs of the VE [Grif00][Gali02][Boho00a]. Initially the VE client contacts the AVPP and requests connectivity between two remote sites. Contract negotiation is the first consideration and this is handled by a mobile agent carrying the VE's request to a selected NP and negotiating locally the contract parameters. Agent mobility allows us to decentralize this task by sending the required negotiation logic to execute locally at a specific NP. Agent autonomy is manifested in the form of intelligent collaboration between the negotiating agents. This is done in a generic manner through the use of ontology-based messages written in FIPA-ACL, in conjunction with Extensible Markup Language (XML) descriptions. The advantage of this approach is the 'loose' coupling between the AVPP and the NP that can significantly simplify their interactions and speed up any collaboration between them on the introduction of new services. If the contract negotiation phase is successful, a Dynamic Connectivity Management (DCM) interface is created for the VE client, allowing access to VPN management capabilities.

In their basic form, the management capabilities offered by this system are quite similar to the facilities in TMN or TINA and can be realized through distributed objects or mobile code. However, there is one distinguishing advantage in our mobile agent-based approach, reflecting on the way the DCM capabilities can be customized using autonomous mobile agents. Customisation

can be handled solely by the agent system in accordance to VE requirements, or in relation to tailor made management logic provided by the VE. These customized agents are deployed at the network provider's domain, autonomously making any decisions specific to their local environment. This diminishes the need for intervention from the VE and allows the VE to concentrate in the business exploitation of the VPN management facilities.

As mentioned above, one customisation option available is handled solely by the agent system in accordance to VE requirements. This is done by Worker agents that autonomously equip themselves at runtime (e.g. using component related techniques such as reflection, introspection, etc) and migrate once to the targeted node in a constrained mobility manner. The agent has knowledge of a number of available management modules, and selects the ones appropriate to the specific context. A migrating agent carries a tailor made mix of only the required capabilities and parameters down to the targeted network element for execution. As an example, we can consider a Worker agent for performance monitoring required in the context of a video–conferencing service offered by the VE over an IP network. Because the service is dependent on prompt delivery of data, the agent decides on supporting the delay and jitter performance parameters and will then equip itself with delay and jitter-related modules developed following the IETF's guidelines for IP performance metrics. Additionally, out of the various management algorithms used to process information it selects a uniformly weighted moving average smoothing algorithm, allowing it to quickly identify performance trends and notify promptly when conditions deteriorate.

In the second customisation option available a VE can enhance the system with its own tailor made management logic. For instance, regarding performance management capabilities, customized management logic could be deployed to monitor connectivity resources and dynamically re-negotiate parameters such as bandwidth in the case of under or over-utilization. In this way, it is possible that the capacity of connections can be configured dynamically by the VE in accordance with the current business activities. In the case of severe performance deterioration, or a fault, a VE is given the capability to request for the re-establishment of connectivity with a different NP or take other appropriate action.

The work on mobile agent-based performance management within the context of this case study was performed while involved in the ACTS-MIAMI project [MIAMI] between 1998 and 2000. Mobile agents allowed us to provide dynamically managed VPN services supporting the dynamic extranet required by a VE. The system suitably exploits autonomous software agents that follow a constrained migration strategy in order to offer a programmable connectivity service (i.e. an active virtual pipe).

5.2 QoS Configuration and SLA Auditing of IP-DiffServ Networks

This case study considers a scenario of a service provider that establishes a Service Level Agreement (SLA) with a network provider in order to access its network capabilities. The service provider uses the network capabilities in order to effectively provide and differentiate its service offerings. The capabilities offered allow it to control the establishment and release of connections according to the SLA and also audit the QoS parameters of established connections. A connection can be unidirectional or bi-directional and associated with qualitative of quantitative QoS assurances between two end points during a specified time period. Further to QoS configuration, SLA auditing allows the service provider to ensure that the QoS assurances specified in the SLA are delivered by the network.

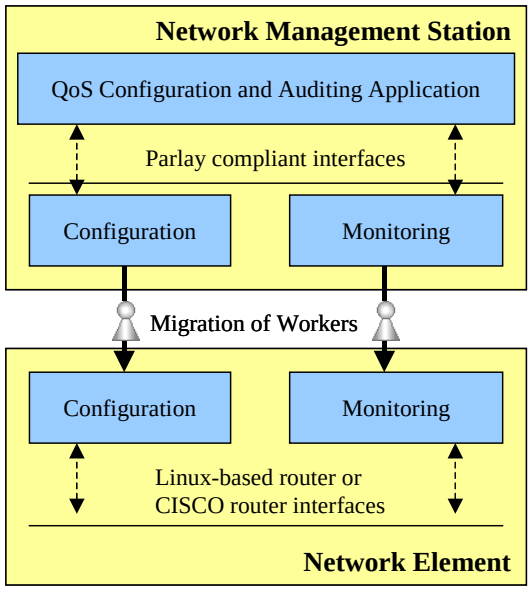


Figure 5-2: A DiffServ QoS configuration and SLA system

A typical scenario of operation starts with a client of the service provider requesting to use a service. The service provider retrieves a set of "class of service" templates supplied by the network provider and selects the appropriate one based on service requirements. The template is filled with QoS configuration instructions involving aspects such as QoS parameter values (e.g. delay, loss, etc), the excess traffic treatment actions, the end points of the required connection, the direction of traffic flow and time requirements. The completed template is then passed to the network provider for validation and connection establishment.

The Auditing application allows a service provider to monitor the QoS parameters of network connections and check their conformance to the SLA. In order to achieve this goal, the system relies on performance monitoring Worker agents that migrate in a constrained mobility manner to the edge nodes of a connection and perform their task. Any reports or notifications are remotely

passed from the Worker agents to the SLA auditing application at the network management station for analysis.

The use of "constrained" mobile agents allowed us to build a system that benefits from autonomous and tailor made management capabilities [Boho00c]. A client may request to use a service from various nodes within a network provider's domain and each time Worker agents are responsible to migrate where required. Before migration, these Workers configure themselves with the appropriate management modules based on the network technology involved (i.e. in our case Linux-based routers or CISCO routers) and the requirements imposed by the SLA. In addition, the service provider is given the capability to customize or extend the standard performance monitoring Workers of the system in order to better address service requirements. For instance, a service provider offering an "audio-conferencing" service can customize performance monitoring Worker agents to provide pro-active notifications based on the tendencies of network conditions, allowing the prompt re-configuration of audio transmission parameters to the appropriate levels of quality. Once more, such capabilities of customisation of management logic at network nodes either by a user, or autonomously by the system, are not supported through alternative distributed object approaches and highlight the usefulness of "constrained" mobile agents.

We should finally note that the work on mobile agent-based performance management within the context of this case study was performed while involved in the IST-MANTRIP project [MANT] between 2000 and 2001.

5.3 QoS Management for the Virtual Home Environment

The preparation of the way for the so called 3rd generation mobile network infrastructure is complemented by an increased interest of the industry in new advanced services that will "intelligently" cooperate with their environment in order to support features not possible so far. Within this vision of "intelligent", distributed software systems for telecommunications applications, an important emerging concept is that of the Virtual Home Environment (VHE) [3GPP00]. A VHE manages a number of services with the aim of consistently providing personalized service aspects to the user, irrespective of the terminal, network and geographic location involved (see Figure 5-3). In order to fulfil this goal, a VHE depends on a sophisticated adaptation process. Initially, when the user requests a service, a VHE adaptation component gathers information about the environment involved in service provisioning. This information reflects on a number of parameters describing the network (for adaptation to network performance conditions and Quality of Service (QoS) offered), terminal (for adaptation to terminal capabilities) and geographic location (for location-based adaptation of service content).

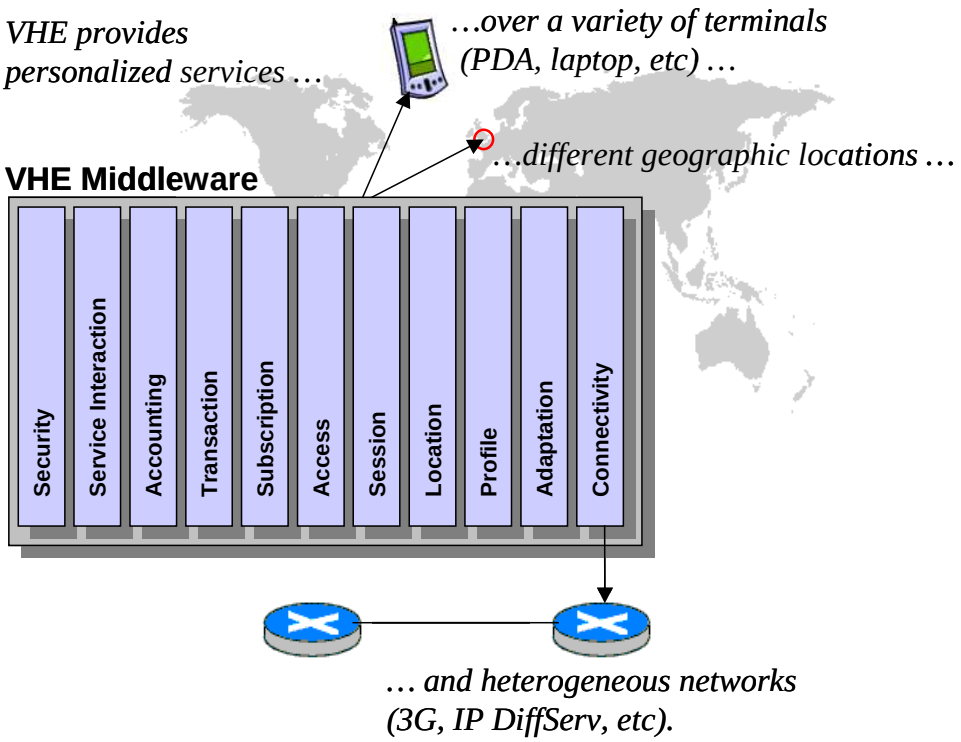


Figure 5-3: The VHE Environment

In order to trigger an adaptation action the VHE adaptation component performs a careful analysis of this information in relation to the user's preferences, kept in a VHE user profile. Following this initial adaptation process and during service usage, the VHE adaptation component depends on real time notifications of any changes in the environment.

Most importantly, a network performance management system, part of a VHE connectivity component, crucially supports VHE adaptation by keeping it informed of changing network conditions that are analysed and may trigger an adaptation action. This real time adaptation process is particularly important for the VHE as it ensures service continuity and correct operation. Performance management involves two different aspects, namely QoS management and performance monitoring. The QoS management part configures connectivity so that the service traffic is delivered with QoS assurances. The performance monitoring part monitors the network performance conditions affecting the connection and informs the VHE of network QoS changes that may require a service adaptation action. This is particularly important when only weak QoS guarantees can be delivered, e.g. qualitative as opposed to quantitative ones, or when only best-effort connectivity is supported.

The work within this case study was performed between 2000-2002 within the context of the IST-VESPER project [VESPER], doing research on architectural and design issues of a VHE system and investigating the role of mobile agents within the dynamic VHE environment [Boho03]

[Liot02b] [Yew01]. In the following sub-sections we examine these research aspects as related to network performance management.

5.3.1 Deployment Requirements

A VHE system typically exhibits the following important characteristics:

1. It is a large-scale system, as it serves a large number of users and needs to control and manage several networks for this.
2. It is a heavily dynamic system, with users, service types, locations, management and control requirements, access terminals and networks changing as a typical part of its operation.
3. It is able to operate over a heterogeneous network environment.

Table 5-1: VHE requirements on connectivity management

Requirement	Description
Universality	A user can request to use a service in a VHE manner from anywhere at anytime. In order to accommodate this requirement, suitable management functionality should be available for execution at 'any' network element that might be involved in a user's connectivity path.
Dynamic, programmable management functionality	The requirements on management functionality vary depending on the service the user chooses. For example, a video conferencing service has different management requirements compared to an on-line calendar service (this is a shared agenda used concurrently by many users). In addition, a key target of a competitive VHE provider will be the rapid introduction of new VHE services. In this respect, management flexibility should be provided to allow for efficient addition and customisation of the available functionality in order to accommodate different service needs.
Network technology transparency	The user may be connected using a heterogeneous network environment. Management logic should translate a user view of requirements from the network into an abstract network view of these requirements and then map them into the specifics of different underlying network technologies.

This dynamic nature of the VHE gives rise to a number of crucial requirements on VHE performance management (see Table 5-1). An important requirement stems from the fact that a user may unexpectedly request a VHE service from any geographic location. In a decentralized

manner, VHE performance management functionality should operate at any "non-provisioned" location to dynamically configure and manage the user's connectivity path. Additionally, a user may rely on heterogeneous network infrastructures (e.g. Universal Mobile Telecommunications System (UMTS) or Internet Protocol (IP) based networks). The involved network is dynamically determined by a VHE performance management component that decides on the appropriate management logic required (e.g. on network specific performance parameters, measurement methodologies). Finally, the user may select from a number of diverse services with different management requirements. VHE performance management should be able to configure its operation based on the requirements of the specific service (e.g. a network delay sensitive video transmission service or a packet loss sensitive software download service). A natural way for a VHE performance management component to address these requirements is by dynamically deploying tailor-made entities in the appropriate network nodes. Among the various alternative management approaches available today, mobile agent technology has the potential to provide a preferred engineering approach to the realization of a dynamic performance management component for the VHE. In the following section we move on to examine how several management approaches available today cope with the VHE requirements.

5.3.2 Benefits of Agent Mobility

As mentioned before, SNMP is currently the most popular management technology and has proven to be very effective for basic management tasks (e.g. monitoring of local area networks). Despite this, scalability problems due to the centralized nature of SNMP make it inappropriate for the management of a large-scale VHE. In addition, any upgrade/customisation of functionality in network elements requires introduction (re-) compilation and activation of SNMP code and this clearly cannot satisfy the requirement of a VHE for dynamic, programmable management.

In comparison with SNMP, distributed object systems provide a more scalable approach to managing networks, capable of fulfilling the VHE requirement for scalable operation. However, in a similar fashion to SNMP, distributed object frameworks still rely on fixed management functionality that cannot be customized or upgraded without system re-installation, a fact that fails the VHE requirement for programmability. The lack of support for programmability in distributed object approaches also clashes with the 'Universality' requirement of the VHE. For example, in order to accommodate a user that may unexpectedly request a service from/to a "non-provisioned" location, we would need management functionality satisfying the VHE requirements in the edge network elements.

By exploiting software mobility, a mobile code approach in the form of the IETF's Script-MIB and mobile agents both fulfil the VHE requirements for programmability and 'Universality' by

dynamically deploying software entities where and when needed. However, the Script-MIB approach is specific to the SNMP management framework and thus does not fulfil the VHE requirement for network technology transparency. On the other hand mobile agents provide a generic approach. Agents can be configured to autonomously acquire the appropriate context (e.g. support for measurement of performance parameters specific to a particular network infrastructure) for cooperation with the underlying network technology identified and thus fulfilling the VHE requirement for network technology transparency.

Based on the above discussion, we see that the mobile agent approach conveniently fulfils all three stated VHE performance management requirements summarized in Table 5-1. In the following section we describe a scenario of VHE service adaptation supported by an agent constrained mobility-based performance monitoring system.

5.3.3 VHE Service Adaptation Scenario

Let's now consider a VHE service involving the transmission of a video stream from a video content server to the user's terminal. Upon initiation the service is responsible for informing the VHE adaptation component of the specific service parameters supported for adaptation (e.g. Bit-rate of video transmission, frame size, etc.). Subsequently the adaptation component initialises the VHE components responsible for network, location and terminal aspects. VHE performance management component gets information about the specific network involved and creates workers equipped with the appropriate performance monitoring modules. This is done as described previously in Section 4.3.1.1 by autonomously configuring Worker agents based on network capabilities and the high level QoS requested through the service. These customized mobile Worker agents subsequently migrate to the corresponding 'edge' nodes of the user's connectivity path in order to perform their tasks.

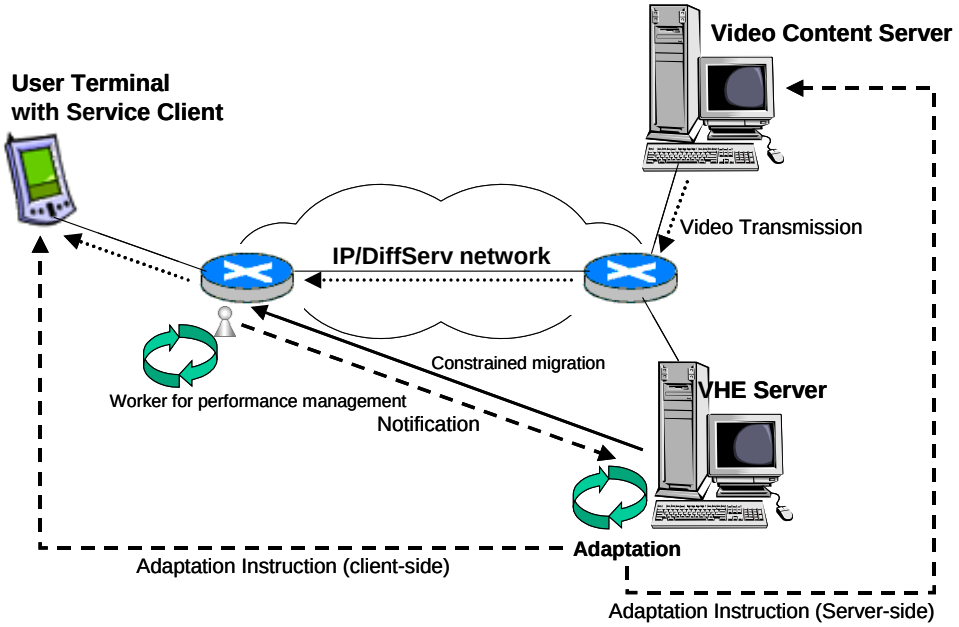


Figure 5-4: Supporting VHE service adaptation

Regarding network adaptation, the network performance monitoring system sends a notification to the adaptation component informing it that network conditions affecting the service have changed. During service usage the adaptation component analyses the effect of any arriving network performance notification and decides whether service adaptation is required (e.g. ask the video server to reduce the bit rate of transmission in case of deteriorating network conditions).

5.4 Experimental Assessment

5.4.1 Introduction

A significant drawback relating to all three case studies of mobile agent-based performance management was in terms of the performance overheads of the resulting systems. Most importantly the practical experienced gained led to the realization ([Boho00c]) that modern mobile agent platforms like Grasshopper offer a lot of general-purpose agent capabilities that are not typically required in the network management context. This realization was much later adopted by other researchers and as an example we present below a quote by Gian Pietro Picco, a recognized expert in mobile code research that states in an August 2002 article [Kotz02]: “Most mobile agents systems (termed as “mobile agent platforms” within this thesis) try to solve ten problems at the same time. They tend to be monolithic. People who want to only use one slice of the system have to install the whole thing”.

The functionality that supports these advanced agent capabilities nevertheless is present in the agent execution environments or inherited by the agent systems and results in a significant

increase in performance overheads. First, we have general-purpose agent utilities, e.g. for cloning, distributed logging, etc. As an example, the Grasshopper (v2.2.3) abstract MobileAgent class, sub-classed by all mobile agents in a system contains a total of 35 agent-related methods. Secondly, we have functionality required to support multi-hop agent mobility and this impacts significantly the complexity of the distributed communication model as well as the agent lookup and binding facilities of the platform. Typically, advanced techniques of component frameworks such as reflection and introspection are necessary in order to facilitate the binding and communication of mobile agents that may autonomously migrate to several nodes during their execution. In the following sections we present in detail the performance overheads of the mobile agent system involved in our case studies as compared to distributed object and mobile code approaches offering similar functionality.

5.4.2 Methodology and Environment

The experimental work presented in this section aims to provide a detailed comparison of different network management approaches. The reference mobile agent-based performance monitoring system used, contained a common denominator of functionality available between the systems of the three case studies (i.e. metric monitoring, summarization, reporting, threshold notifications). In order to allow comparisons the author developed performance monitoring systems with similar functionality based on distributed objects (using Java-RMI and CORBA in Sun Microsystems's JDK v1.3.1), mobile code (using Jasmin Script-MIB v1.0) and mobile agents (using IKV++ Grasshopper v2.2).

While preparing the comparative experiments, the author's first concern was to obtain measurements that highlight the performance overheads incurred by each of the systems. In this direction, the aim was to identify those system operations that indicate a possible network performance bottleneck. An important operation with impact on the network involves the scheduled remote transfer of performance reports and notifications from the managed network element to the network management station (e.g. as in Figure 4-1). The measurements consider reports of gradually increasing size, containing 25, 50, 75 and 100 real 'double' numbers. A performance report involves the TpReport data type (see Figure 4-3) containing a list of all the performance monitoring information produced. These numbers reflect on the information gathered during the latest cycle of the performance monitoring process. Similarly, a performance notification involves the remote transfer of an object of the TpNotification data type seen in Figure 4-3.

Another operation that crucially impacts the network is related to software migration. The measurements consider the migration of a Grasshopper and Script-MIB mobile Worker entity carrying management logic to the managed network element.

For these two operations we have taken the following network-related measurements:

- Traffic measurements: Taken using the tcpdump utility with the sizes reported reflecting on the total payload at TCP level. As an exception for the Jasmin Script-MIB system the sizes reported reflect on the total payload at UDP level.
- Response times measurements: Taken using the System.currentTimeMillis() method included in the API of Sun's JDK. The measured values of response times reported represent the "steady-state" running costs, excluding the initial setup costs. For each reporting operation involving a specific report size as well as a notification operation, measurements where repeated 100 times, with values presented in graphs representing the average value arising from these measurements.

In addition to network considerations it is important to highlight the requirements of each system on the resources of a managed node. In the measurements the author has considered the memory usage at the managed node resulting from the execution of each performance monitoring system and its supporting platform. The used memory reported was calculated as a result of the difference in measurements of the total memory allocated by the Java Virtual Machine and its free memory. These two values were obtained using the totalMemory, and freeMemory methods of the JDK's java.lang.Runtime class. As an exception for the case of Jasmin Script-MIB because the platform support and executing systems are incorporated with the native SNMP agent, the memory usage reported was obtained using the UNIX top utility.

From a different perspective and in an effort to assess the 'development' overheads associated with each system we present results on the number of lines of source code required for each performance monitoring system. In this direction, every effort was made to keep the same programming style and comments, with all counts reported taken using the UNIX wc utility.

Finally, we should report that the runtime and measurement environment involved a dedicated testbed of Linux workstations with homogeneous features (Redhat 7.1 (kernel 2.4.2), Pentium Celeron 466MHz and 64MB of RAM), connected to a 100Mbps Ethernet network.

5.4.3 Results

5.4.3.1 Remote Data Transfers

The experimental results regarding the traffic incurred in the network for reporting and notification operations can be seen in Figure 5-5 and Figure 5-6 respectively.

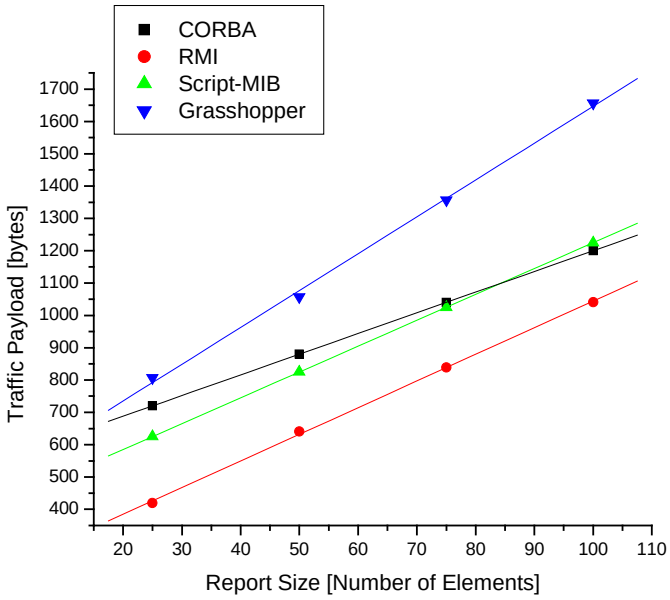


Figure 5-5: Report traffic associated with four alternative technologies

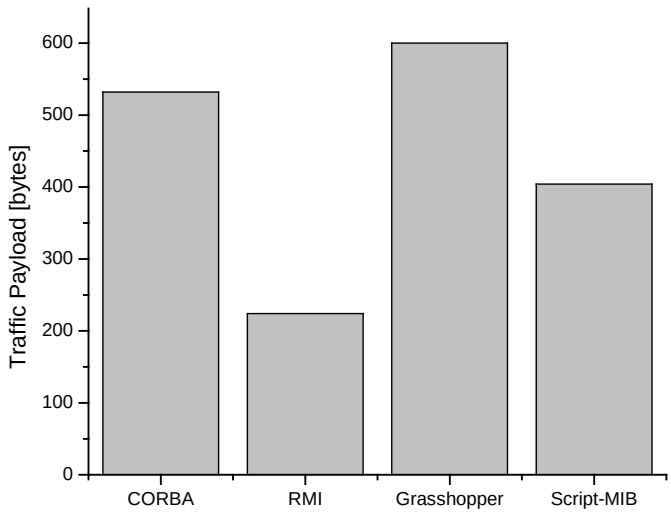


Figure 5-6: Notification traffic associated with four alternative technologies

From these figures we can observe that for a small amount of data (a notification or a report of 25 elements) the Grasshopper system performs competitively incurring around 15% more traffic than

CORBA. The steep slope of the line crossing through the Grasshopper measurement points shows that the mobile agent approach does not scale well for increasingly large reports. As such, for a report of 100 elements it incurs 38% more traffic than CORBA. The Jasmin Script-MIB approach produced moderate performance results and is outperformed by CORBA only for large reports containing more than about 75 elements. The slope of the CORBA line indicates that the approach scales better than all the others in the comparison. Despite this, the significantly lower performance overheads of Java-RMI within this report sizes range show that CORBA eventually outperforms Java-RMI only for reports containing more than 100 elements.

The experimental results on the response times for the reporting and notification operations can be seen in Figure 5-7 and Figure 5-8 respectively.

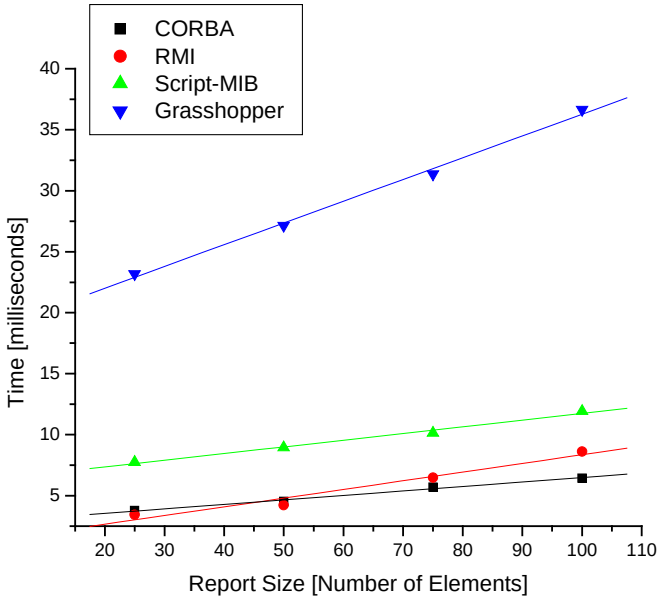


Figure 5-7: Report times associated with four alternative technologies

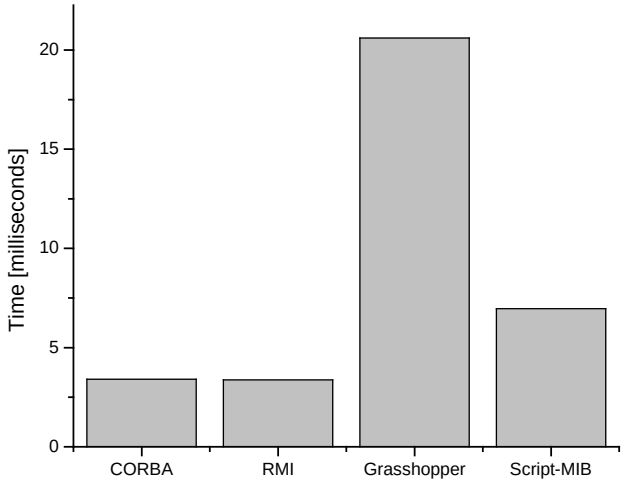


Figure 5-8: Notification times associated with four alternative technologies

From these figures we can see that for a small report of 25 elements or a notification, Grasshopper is about 4 times slower than Jasmin Script MIB. The two distributed object systems, CORBA and Java-RMI offer in most cases comparable performance that is about half the time needed by Jasmin Script-MIB. From the slope of the Grasshopper line we can see that the approach has the worst scalability for increasingly large reports, while CORBA on the other hand offers the best scalability and manages to outperform clearly Java-RMI for reports of 100 elements.

As mentioned in the methodology, for each reporting operation involving a specific report size as well as a notification operation, response time measurements where repeated 100 times, with values presented in the previous graphs representing the average value arising from these measurements. Figure 5-9 below, presents the uncertainties related to the response time measurements for the operation of remote reporting. Each of the four charts in the figure shows the results of the statistical analysis of the data gathered for each system in the comparison. The boxes include the 25-75% range boundaries, the mean values (a small square) and the median values (a line). Outside the boxes we have the error bars as well as the 5-95% range boundaries delimited by whiskers. The data presented in Figure 5-9 are indicative of the response times uncertainties presented throughout this thesis, arising from factors related with the experimental testbed used (e.g. fluctuations in network utilization, CPU utilization, etc). The results in this figure sufficiently validate the conclusions drawn previously, showing a minimum overlap due to uncertainties between the four systems in the comparison.

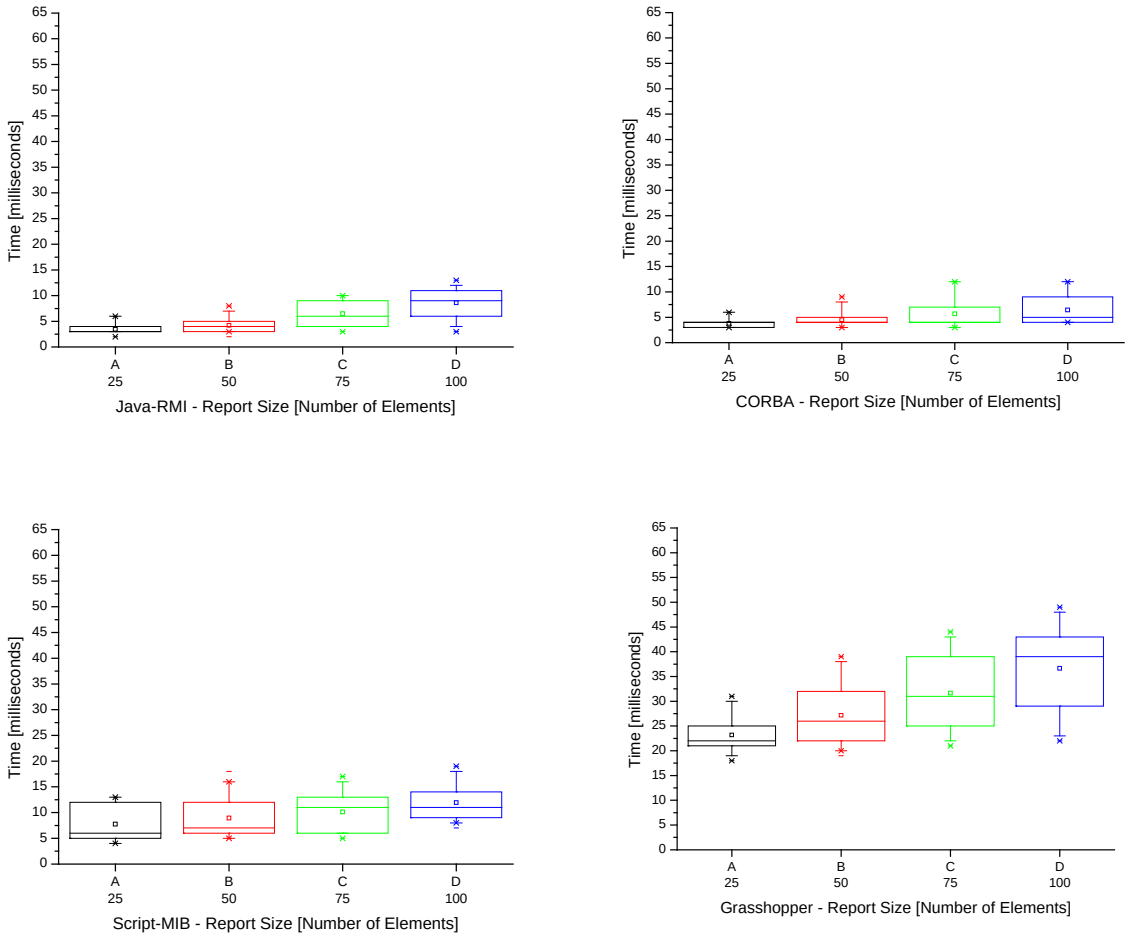


Figure 5-9: Statistical graphs on response times uncertainties

The above results on traffic and response times associated with remote transfer of data for the four different approaches reveal a number of interesting findings. We can see that Grasshopper, a typical, all-round mobile agent platform ([Guth98], [Mich00], [Silv00]), exhibits significantly higher performance overheads than the distributed object and mobile code approaches of our comparison. These are commonly attributed to the much more dynamic communication mechanisms required (i.e. using capabilities of component frameworks) in order to accommodate the remote communication of autonomously migrating, multi-hop agents. In Script-MIB and distributed object approaches, the end-points of remote communication are fixed between distributed entities allowing the approaches to optimise the operation and achieve significantly better performance results.

5.4.3.2 Software Migration

The experimental results regarding the traffic and times associated with the migration of a Grasshopper mobile agent and a Jasmin mobile code script can be seen in Figure 5-10 and Figure 5-11.

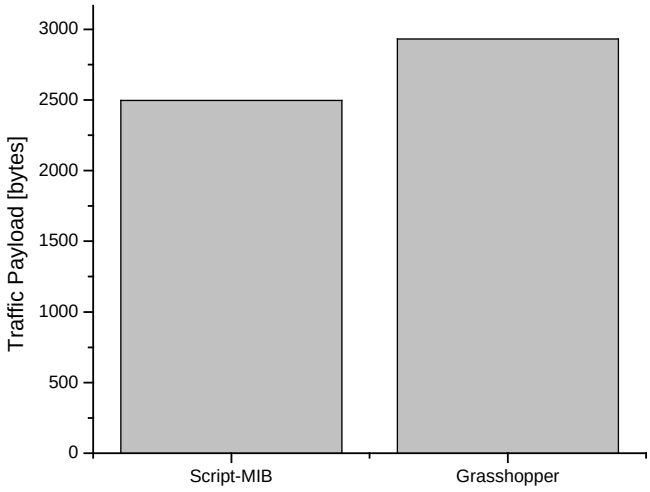


Figure 5-10: Migration traffic incurred by Script-MIB and Grasshopper Workers

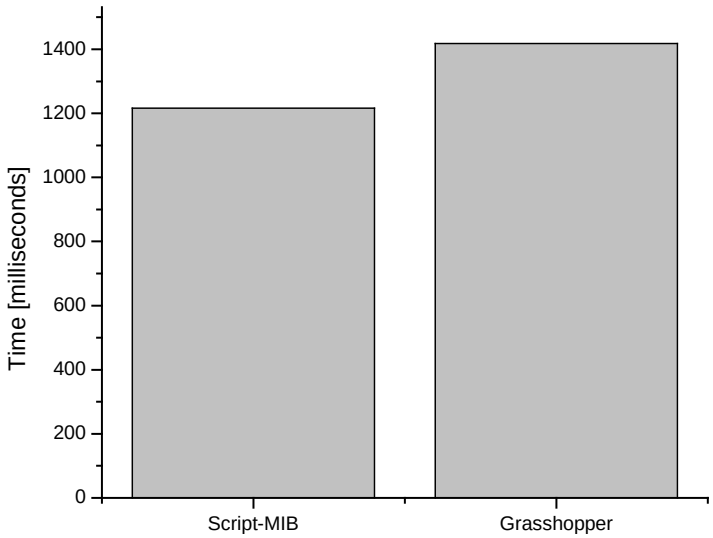


Figure 5-11: Migration times for Script-MIB and Grasshopper

Regarding software migration, the mobile agent Worker needed 1,418 ms to migrate while it incurred 2,932 bytes of traffic. The Jasmin Script-MIB Worker performed slightly better requiring 1,216 ms and incurring 2,496 bytes of traffic. For comparison, typically the creation of a distributed object through a factory requires less than 15 ms to complete and incurs around 500 bytes of traffic [Pavl98]. In this respect a mobile agent Worker takes for its setup about 100 times longer and incurs about 25 times more traffic compared to a distributed Worker server. As such, software migration is a particularly costly operation. This is an important concern while designing a network management system that involves software migration. The results show that the amount

of migrations involved with the system's operation should be kept to the minimum possible. Migrations are justifiable only when there is a task to be performed at a remote managed node for which the required logic does not already exist there. Regarding mobile agent-based network management, constrained mobility respects this requirement through its "migrate-once" strategy. Additionally, the size of migrating entities should be kept to the minimum possible. Again, constrained mobility helps in this direction, since upon creation at a network management station and before migration, Worker agents autonomously equip themselves with only the required functionality modules for their particular task. While executing at a network node, Worker agents communicate information through remote data transfer only.

5.4.3.3 Memory Usage at a Managed Node

Regarding the resource requirements at a managed network element we have measured the memory usage required for the execution of the performance monitoring system and its supporting platform. The memory usage results can be seen in Figure 5-12.

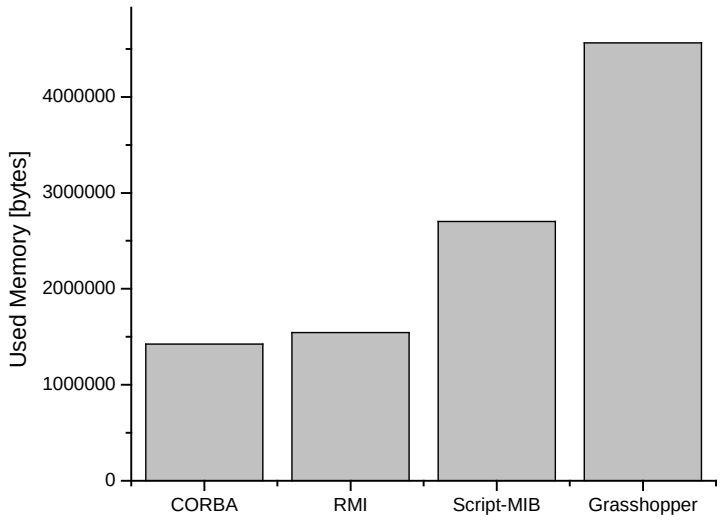


Figure 5-12: Memory usage at a managed node associated with four alternative technologies

The CORBA and Java-RMI based systems have low memory utilization requirements at similar levels. While Script-MIB produced moderate results, a further increase in capabilities available in the Grasshopper mobile agent approach results in significantly higher memory usage. Memory requirements within managed nodes are important since network devices typically offer a modest amount of computational resources. Despite this, in many cases, the use of a DCN network consisting of significantly more powerful workstations dedicated to the management of network nodes means that the high memory requirements of Grasshopper can be considered acceptable in such a case.

5.4.3.4 Software Development Metrics

In addition to the performance characteristics of each system software development metrics were considered as an indication of the complexity of working with each technology. Figure 5-13 depicts the results on the source code size required for the development of each system. In more detail, in Table 5-2 we report on "Generated" code representing platform-specific supporting code (e.g. object skeletons and stubs) that is bundled along with the performance monitoring code written by a programmer.

Table 5-2: Software metrics associated with four alternative technologies

	Grasshopper	Java-RMI	CORBA	Jasmin
Generated Compiled Code [Kbytes]	0	60	136	0
Written Compiled Code [Kbytes]	86	84	92	138
Total Compiled Code [Kbytes]	86	144	228	138
Generated Source code [Lines]	0	2064	2460	0
Written Source Code [Lines]	2363	2540	2583	2972

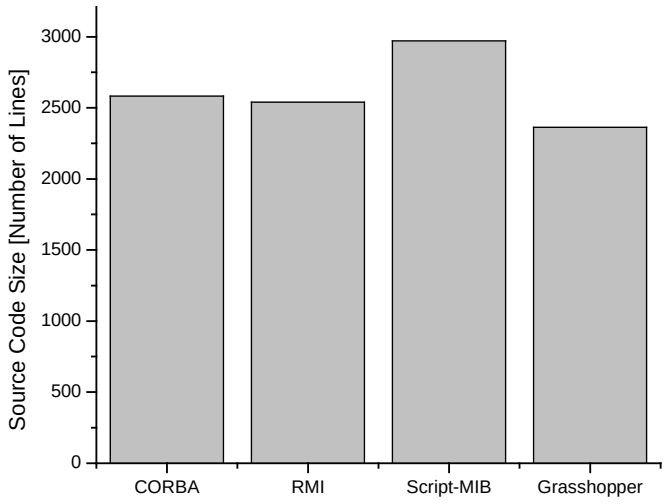


Figure 5-13: Software development metrics

The high level of abstraction in the Grasshopper APIs resulted in the smallest written source code in our comparison. As it is often the case with modern mobile agent platforms, Grasshopper offers good documentation, conveniently designed APIs and a stable execution environment that

significantly simplified the development cycle. In the other side of the spectrum, the Jasmin Script-MIB platform required the most code, it is currently poorly documented and unstable. Furthermore, CORBA and Java-RMI both rely on a large amount of platform-specific code contributing significantly to the total compiled code of the system. Both platforms offer a stable and well-documented environment for developing and running distributed applications.

5.5 Conclusions

The three case studies examined demonstrated that the combination of mobility and autonomy of constrained mobile agents can effectively address modern network management requirements for programmable and dynamic operation. Customized management logic is assembled autonomously by the agent system or provided by the user and subsequently migrates within an agent allowing in both cases the easy programmability of network nodes. Autonomous self-configuration is particularly important in highly dynamic environments, such as in the VHE case, where network management functionality needs to be deployed by the VHE system itself in a tailor-made fashion according to the network technology and service requirements, when and where needed. The combined benefits of easy programmability and autonomous, self-configurable operation distinguish mobile agents from distributed object and mobile code approaches. Despite this, the more advanced capabilities of mobile agents are typically associated with additional performance overheads that may not always be acceptable. Modern mobile agent platforms aim to be general purpose, all round solutions by aggregating agent-related functionality, most of which, was never required within the network management context. All that was identified as necessary for the creation of our network management systems was the required facilities for constrained mobility. This provided the motivation for the work on a lightweight approach to constrained mobility, combining the minimum required mobile agent functionality, the ability to deliver the same benefits of programmability and autonomy, while also reducing the performance costs. The approach is presented in Chapter 6 that follows.

Chapter 6

6 A Lightweight Approach to Constrained Mobility

This chapter presents an approach to mobile agent-based network management based on a specially formulated platform for constrained mobility that suitably enhances the capabilities of a distributed object framework. The platform follows a minimalist approach to provide only the required facilities supporting the constrained mobility strategy in an effort to minimize network performance overheads. The resulting prototype for constrained mobility provides an efficient solution in the direction of evolution of network management approaches from current distributed objects towards full-featured mobile agent solutions.

6.1 Introduction

The work on a lightweight platform for constrained mobility was motivated by the combination of the most prominent advantages and disadvantages from mobile agent-based network management that were discussed in chapters 4 and 5. Mobile agents allow for the easy programmability of network elements and the autonomous, self-configurable operation of a network management system. Unfortunately as seen in Chapter 5, modern mobile agent platforms provide a lot of general-purpose capabilities that are typically unused for network management purposes but nevertheless contribute to increased performance overheads. In the beginning of 2000, work commenced on the first platform to provide only the minimal required support for the development of “constrained” mobile agents. The platform facilities exploit and augment the capabilities of a distributed object framework for remote transfer of data and object binding. This approach is in contrast with other mobile agent platforms that implement their own lower level communication protocols and naming facilities. As the prototype platform work envisioned a “Java everywhere” environment the Java-RMI distributed object framework was chosen, as a well-integrated and optimised framework for Java-only systems. Despite this, the proposed approach has a wider scope and can be realized with any distributed object framework.

6.2 Proposed Approach

The final, integrated systems of the case studies presented in Chapter 5 consisted of a large number of agents in the Master, Worker and Target roles, with each sub-system operating using a constrained mobility strategy. It was soon realized that the same benefits of these mobile agent-based systems could be delivered more efficiently based on a 'lightweight' platform that only combines two required characteristics. A mechanism responsible for the single-hop migration involved in constrained mobility along with the flexibility to introduce autonomous behaviour. In order to support constrained mobility the following two platform facilities are required:

- A platform mechanism that caters for the sequence of suspension, termination, serialization, migration, de-serialization and resumption of agents. These are the steps associated with the migration of a Worker to a targeted node.
- A naming service that caters for agents that migrate once to a targeted node.

Based on these required facilities, the work resulted in a 'lightweight' platform for constrained mobility named, the 'CodeShell' [Boho00c] (see Appendix A for CodeShell UML diagrams and Appendix B for CodeShell Source Code). The platform exploits and enhances capabilities for static distributed objects provided by Java-RMI as well as components capabilities of the JavaBeans framework [JAVAB]. Java-RMI was exploited in order to provide:

- A generic remote interface for the transfer of a serialized agent in byte code form. This is realized by a component referred as the CodeShell Communication Service (CCS).
- An agent-naming component, referred as the CodeShell Naming Service (CNS), ensuring the appropriate naming, binding and lookup of migrating agents (by enhancing the Java-RMI registry capabilities).
- A distributed object that serves as a generic agent execution environment and also encapsulates the above services.

Regarding the JavaBeans framework, reflection capabilities conveniently allowed us to dynamically initialise and resume any specific mobile agent in a generic manner.

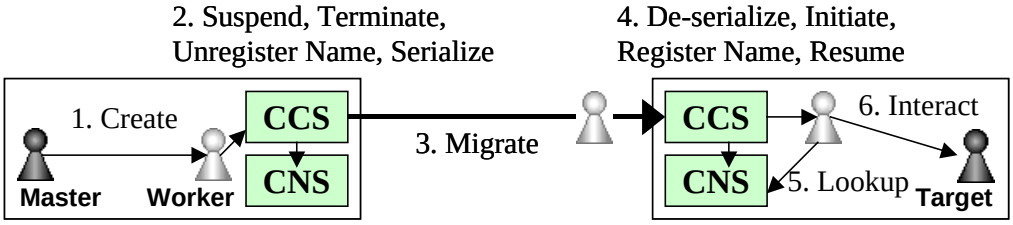


Figure 6-1: Constrained mobility between two remote CodeShell execution environments

A typical scenario of operation commences with the Master agent creating a Worker. The agent configures itself with the required parameters and management modules and contacts the CCS through the platform's external API. The CCS follows a sequence of terminating and deregistering the current agent reference from the naming service. Through the naming service the destination CCS is located and the serialized Worker is transferred to it as an array of bytes. Upon arrival it is de-serialized, initiated, registered and let to resume its operation based on its current state. In the CodeShell class diagram of Appendix A we can see that this state is represented by the "`state: int[]`" Agent attribute, allowing a number of state indicators to be added. The analysis of the indicators appearing in the agent state at any given time allow it to make an autonomous decision on the appropriate management action or adaptation to the environment, as described in Section 4.3. Upon resumption at the destination node, the Worker agent again makes use of the platform's external API in order to access CNS and perform a lookup for a pre-existing Target agent. When a Target reference is obtained, the Worker can start collaborating with it in order to perform its task. Furthermore, the CNS facilities are used by the Master and Worker agents in order to obtain the required references that allow the remote exchange of information between them.

The capabilities inherited by all CodeShell agents are limited to the essentials (see Appendix A, CodeShell Class Diagram). Further to the typical initiation/termination methods, agents support a method that advertises their supported services as well as a method that allows generic inter-agent communication through the exchange of ontology-based messages. This is an alternative to the capability of agent communication through Java-RMI remote interfaces and can be of particular importance for scenarios with interactions involving complex and dynamically changing data (i.e. pre-defined interfaces of communication become ineffective). Agents also carry a unique identifying name and a method (called "`live()`") to override the behaviour of a specific agent. Agents communicate with the platform for migration and agent lookup capabilities.

By limiting the capabilities supported by the agent platform as well as those inherited by all agents, we have built on top of the CodeShell platform a performance monitoring system that incurs only a fraction of the performance overheads associated with a typical all-round mobile agent platform. These findings are detailed in the following section presenting a thorough experimental assessment on the CodeShell based performance monitoring system.

6.3 Assessment

6.3.1 Methodology and Environment

The methodology and environment used for the experimental assessment follows the descriptions given in part 5.4.2. In these lines, traffic and response time measurements involve the remote transfer of reports of increased size and the migration of a Worker for performance monitoring. The memory usage at a managed node was measured as an indicator of computational resource requirements. Finally, source code sizes are compared as an indication of development overheads. The graphs and descriptions presented in the following sub-sections focus on the CodeShell platform. For comparison purposes data presented in Chapter 5 on the performance of Java-RMI, CORBA, Grasshopper and Jasmin Script-MIB are reproduced here.

6.3.2 Experimental Results

6.3.2.1 Remote Data Transfers

The CodeShell platform has been designed to allow the flexibility for agents to communicate through Java-RMI interfaces. As such, regarding remote communication the platform achieves results that are identical to a plain Java-RMI system. Figure 6-2 shows that the CodeShell, Java-RMI, CORBA and Script-MIB solutions offer the best response times at similar levels.

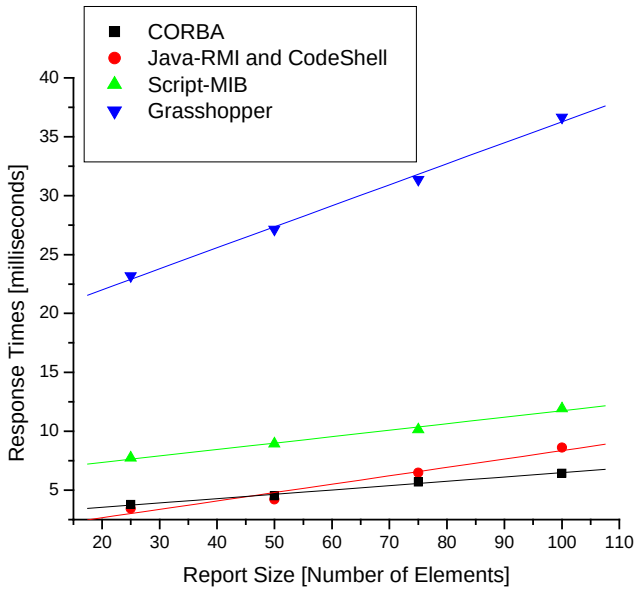


Figure 6-2: Comparison of CodeShell reporting response times

The CodeShell platform is at least 10 times faster than Grasshopper. This gap increases with report size, showing that the CodeShell is a much more scalable solution than Grasshopper when it comes to large amounts of data. Regarding the traffic associated with the reporting operation the Codeshell and Java-RMI based systems clearly incur the least amounts of traffic within our measurement range (Figure 6-3).

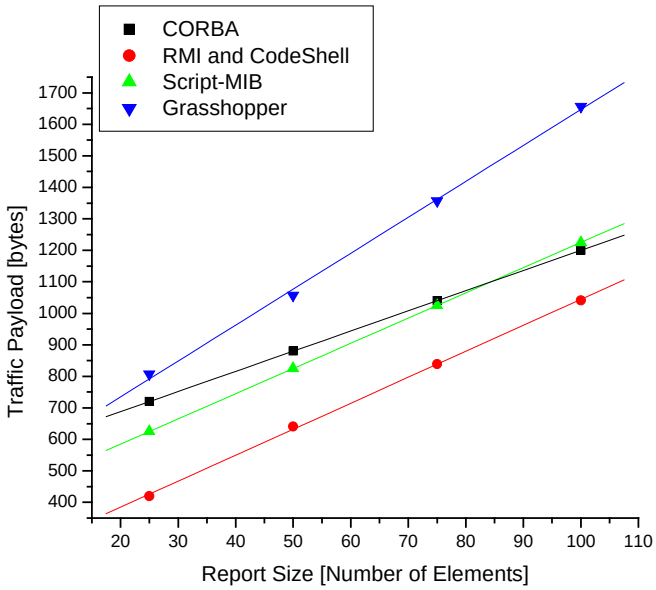


Figure 6-3: Comparison of CodeShell reporting traffic

From the slopes of the CORBA and CodeShell lines in this figure we would expect CORBA to eventually surpass the CodeShell for reports containing more than about 150 elements. The CodeShell based system incurs almost half the traffic of the Grasshopper mobile agent solution.

6.3.2.2 Software Migration

The CodeShell measurement results regarding the required time and traffic associated with the migration of a Worker entity can be seen in Figure 6-4 and Figure 6-5 respectively.

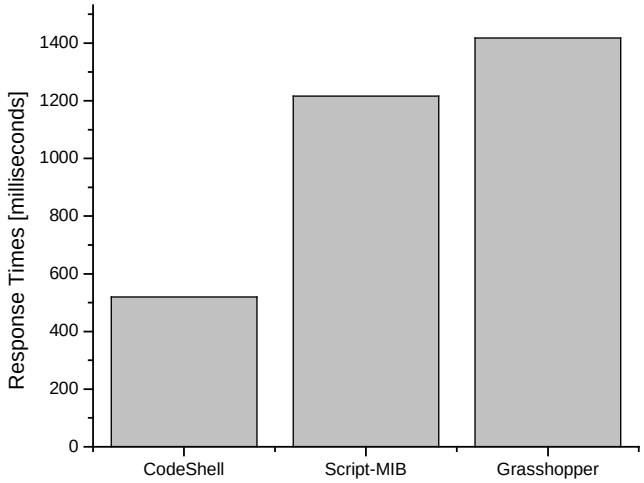


Figure 6-4: Comparison of migration time incurred by CodeShell Worker

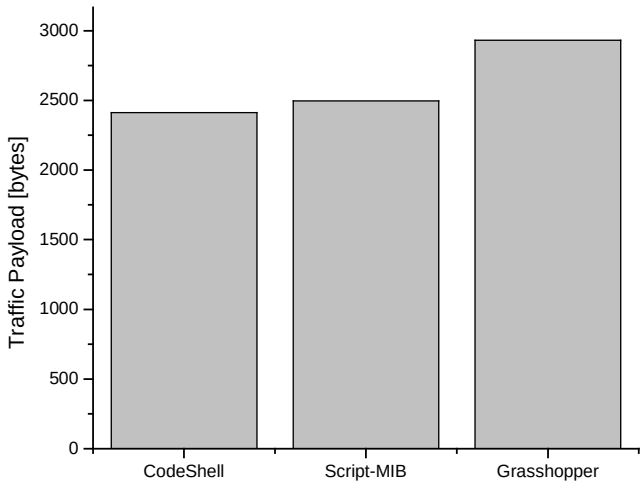


Figure 6-5: Comparison of migration traffic incurred by CodeShell Worker

The CodeShell worker required 520 msec to migrate and proved to be more than 2 times faster than the Script-MIB and Grasshopper workers that took 1216 msec and 1418 msec respectively. Regarding traffic, the CodeShell solution provides the best performance with Jasmin Script-MIB producing a slightly worst result. As we have seen in the figures depicting the remote reporting results, Java-RMI provides very efficient remote transfer facilities that are suitably exploited by the CodeShell platform in order to achieve good agent migration performance. While the Grasshopper worker incurred 2932 bytes of traffic, the CodeShell worker incurred only 2411 bytes, which makes for a significant 521 bytes difference.

6.3.2.3 Memory Usage at Managed Node

Regarding the resource requirements at a managed network element we have measured the memory usage required for the execution of the CodeShell based performance monitoring system and its supporting platform. The memory usage results can be seen in Figure 6-6.

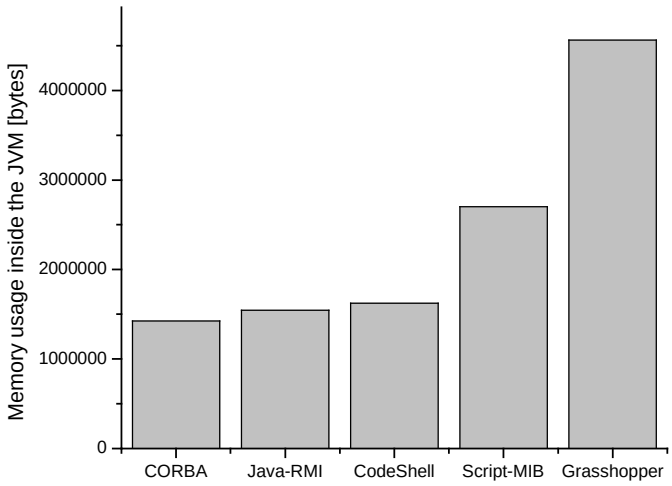


Figure 6-6: Comparison of CodeShell of JVM memory usage

The CodeShell, CORBA and Java-RMI based systems have low memory utilization requirements at similar levels. Compared to Java-RMI the CodeShell management entities and platform facilities for the support of constrained mobility produced a small additional overhead (about (1.62Mbyte - 1.54Mbyte) 0.08Mbyte more). Furthermore the CodeShell memory usage at 1.62MByte compares favourably to the 2.7Mbytes and 4.56Mbytes required by the Script-MIB and Grasshopper solutions respectively.

6.3.2.4 Software Development Metrics

The results on the source code size required for the development of a performance monitoring system for the CodeShell platform can be seen in Figure 6-7.

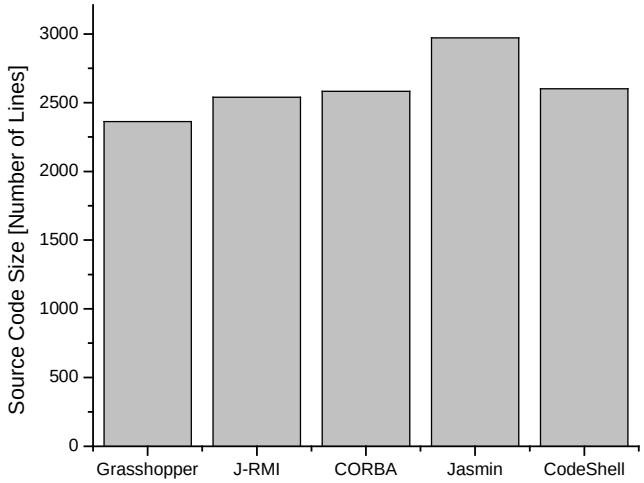


Figure 6-7: Comparison of CodeShell source code size

The CodeShell, Java-RMI and CORBA based systems required a similar number of lines of source code (2562, 2540 and 2583 respectively). While the CodeShell requires 199 more lines than Grasshopper, it still greatly outperforms the Script-MIB approach that requires 2972 lines (i.e. a significant 410 lines difference). The CodeShell effectively combines the advantages of a stable platform taking advantage of the widely used and tested distributed object facilities of Java-RMI, it requires a modest amount of platform specific code and suitably exposes the APIs necessary for the development of constrained mobile agents.

6.4 Conclusions

This chapter presents a lightweight approach to the facilities required for the realization of constrained mobile agents and a thorough evaluation of the constrained mobility strategy for mobile agent-based network management. The author implemented the CodeShell platform for constrained mobility in order to establish whether it was possible to realize management systems based on constrained mobility that at the same time achieve performance levels comparable to the ones of systems based on the most popular static distributed object technologies. The aim was directed at quantifying the performance gain achievable by giving up on general-purpose mobile agent technologies and retaining only the most basic facilities required by constrained mobility. The results presented herein suggest that constrained mobility can be easily integrated in network management systems. Moreover, using constrained mobility it is still possible to achieve performance and scalability typical of static distributed object technologies. Other management functions such as configuration and fault management can similarly benefit from constrained mobility. It is the author's belief that constrained mobility is the most suitable strategy that can find concrete application in network management. While mobile agent frameworks were initially

thought as rivals to static distributed object frameworks, the two approaches need to coexist. The approach presented in this chapter enhances the capabilities of a distributed object framework to support constrained mobility but in some cases the possibility of having a distributed object and a mobile agent framework inter-work may be desirable. In such a case, real synergy could be achieved if stationary agents could be provided using typical static objects, with method invocations being possible between mobile agents and static objects in both directions. Such an environment could combine the advantages of both frameworks and provide a convenient path for the upgrade of capabilities of existing distributed object systems. This proposal is thoroughly examined in the Chapter that follows.

Chapter 7

7 A Hybrid Approach based on Mobile Agents and Distributed Objects

This chapter presents an approach to mobile agent-based network management relying on a specially formulated environment that suitably combines the capabilities of a mobile agent and a distributed object framework. The resulting environment exploits the capabilities offered by the mobile agent framework for autonomous and programmable operation and the small remote communication overheads that typically characterize a distributed object framework. The resulting 'hybrid' environment of collaborating mobile agents and static objects can provide an effective approach for the enhancement of existing distributed object-based network management systems with mobile agent capabilities.

7.1 Introduction

The author's involvement within the case studies presented in Chapter 5 made clear that the rivalry between mobile agents and distributed object alternatives for network management would show no clear winner. On the one hand modern mobile agent frameworks allow us to develop programmable and autonomous network management solutions associated with significant performance overheads. On the other hand systems based on distributed object frameworks offer great performance but depend on fixed, pre-existing server objects at managed nodes. Any network management system involves two types of collaborating software entities, those that operate statically in their network element of creation and those that would ideally be mobile. As such, real synergy could be achieved in a network management system if stationary entities were provided using static objects that collaborate with mobile agents, in a 'hybrid' system that combines the best of both. While chapter 6 presents a lightweight platform realized by enhancing the capabilities of a distributed object framework to support constrained mobility, the proposal of this chapter suitably combines the capabilities of two existing frameworks, one for distributed objects and one for mobile agents. Some preliminary work on the integration of mobile agents and CORBA considered mostly architectural issues [Chat00][Guia01]. The work presented here goes further by concentrating on issues of system design and a performance evaluation of the hybrid approach [Boho02]. While the following section presents the integration of CORBA distributed

objects with Grasshopper mobile agents, the proposed approach of collaborating entities still applies in a general context.

7.2 Proposed Approach

Let's consider a network of managed nodes, each hosting a number of CORBA servers containing an executing piece of management logic (Figure 7-1). While CORBA servers written in compiled languages are tied to a specific hardware architecture and operating system, a number of CORBA servers are written in a language such as Java, allowing them to execute in any managed node running a suitable Virtual Machine (VM) and the relevant ORB.

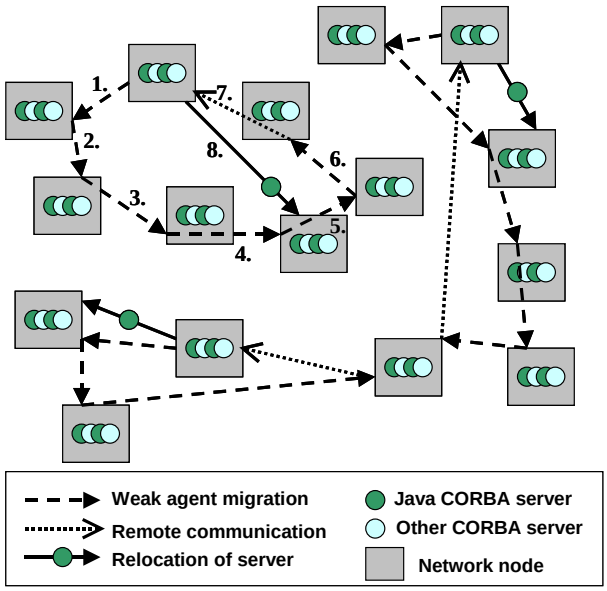


Figure 7-1: A system of dynamically positioned Java-based CORBA servers

For the efficient operation of the system, we would like to have these Java-based CORBA servers placed dynamically in managed nodes. This can be achieved with a mobile agent carrying a CORBA server to the appropriate managed node for execution (Figure 7-2, step 1). In [Liot02a], an algorithm is presented that allows an agent to decide based on an analysis of routing tables on the most efficient node for its migration and execution. The decision mechanisms used in [Liot02a] are a result of a theoretical study, assessed through a set of simulation results confirming the usefulness of the process. In the work presented here we enhance those mechanisms by considering a combination of efficiency parameters and present their realization in a system based on current mobile agent and CORBA technologies. Since the state of network nodes can change with time, we would like to be informed if there is a more efficient node for execution. For this reason, the mobile agent along with the CORBA server can carry the code of a second smaller mobile agent (Figure 7-2, step 1). This small agent can periodically visit (in a weak mobility manner - Figure 7-2, step A) a number of alternative nodes and collect local

performance information that will allow it to possibly instruct (Figure 7-2, step B) a relocation of the CORBA server in a more efficient node of execution (Figure 7-2, step 3).

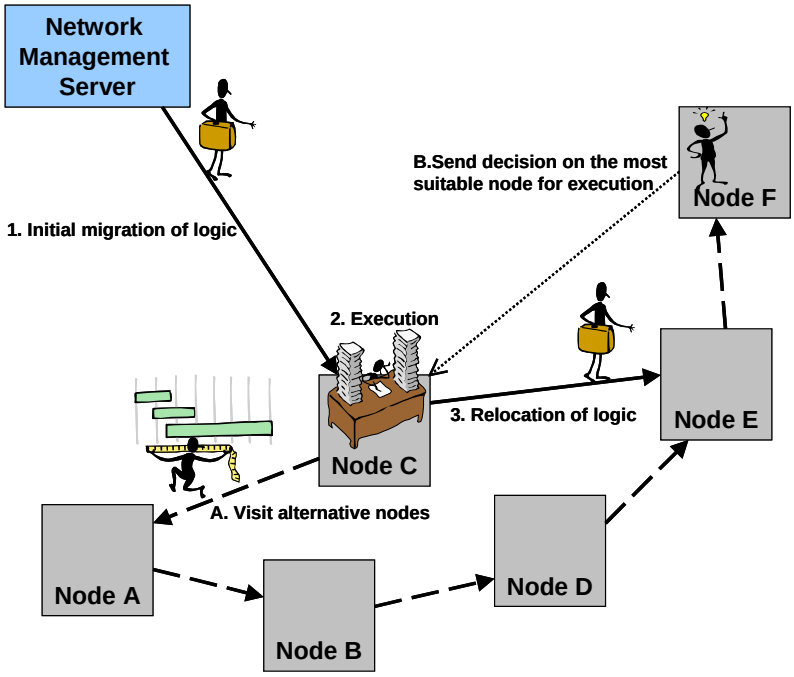


Figure 7-2: A typical scenario of operation and re-location of a Java-based CORBA server

The above discussion allows us to identify that two mobile software entities are required by this approach as well a static object containing the required management logic. In addition, other collaborating static entities can be available in order to formulate a complete management system. In the following section we will demonstrate how the proposed approach can be realized in the context of a system for network performance monitoring.

7.3 A Prototype Hybrid System for Performance Monitoring

The performance monitoring system was specifically developed to support the monitoring of various performance parameters important in an Intranet environment. Based on the approach presented in the previous section we have developed a prototype system for network performance monitoring using a combined CORBA and mobile agents environment. CORBA distributed objects support was provided by Sun's Java mapping of CORBA included in JDK 1.3.1. Mobile agents support was provided by IKV's Grasshopper agent platform version 2.2.1. The performance monitoring system consists of two mobile agents and three CORBA objects as described below (Figure 7-3):

- CORBA Monitor: A CORBA object responsible for performing performance monitoring. It contains all the required management logic for this task involving a combination of metric monitoring and summarization functionality [X739][X738].

- CORBA Master: A CORBA object responsible for initiating and controlling the performance-monitoring task. The Master directly sends to the Monitor any requests for reconfiguration of the performance monitoring process. In addition it receives from the Monitor any performance reports or notifications generated.
- Position Agent: A mobile agent responsible for visiting a number of alternative nodes in the network collecting information on their conditions. Based on this, it can make a decision and instruct for the relocation of the performance monitoring task on a more efficient node of execution.
- Worker Agent: A mobile agent migrating along with the code of the Position Agent and the Monitor into a targeted node. Upon arrival it initiates the Monitor object and passes a reference of it to the Master. It is also responsible for periodically creating a Position agent as well as relocating the monitoring task to alternative nodes of execution.
- CORBA Target: A CORBA object available at each network node allowing the Monitor to access raw performance information. The Target allows the Monitor to take passive performance measurements (e.g. Used bandwidth, loss, etc) by wrapping the underlying SNMP support as well as active measurements by providing an 'echo' facility (e.g. for delay, jitter, etc).

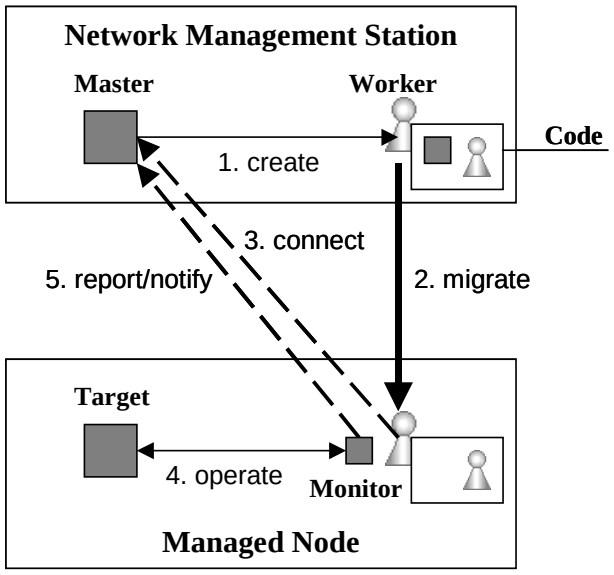


Figure 7-3: Initiation of the Hybrid system for performance monitoring

A typical scenario of system operation commences with a request for performance monitoring from a user of the system creating a CORBA Master. The Master is responsible to create a new Worker agent and pass to it any parameters required for the performance monitoring task (Figure 7-3, step 1). The Worker, containing the code of a CORBA Monitor and a Position agent,

subsequently migrates to the chosen network node. Upon arrival it initiates a CORBA Monitor and passes its reference to the Master so that the two CORBA objects can collaborate directly. The Monitor initiates its task with periodic requests for 'raw' performance information (counter type values) provided by the CORBA Target. Based on this information, the Monitor performs metric monitoring for a number of performance parameters (e.g. Used Bandwidth, Loss, etc), checks the thresholds set and gathers the information produced in order to generate reports. During this task, the Monitor remotely sends to the Master such reports on performance conditions in a scheduled manner (e.g. every 1 hour) as well as notifications in real time when a performance threshold is triggered.

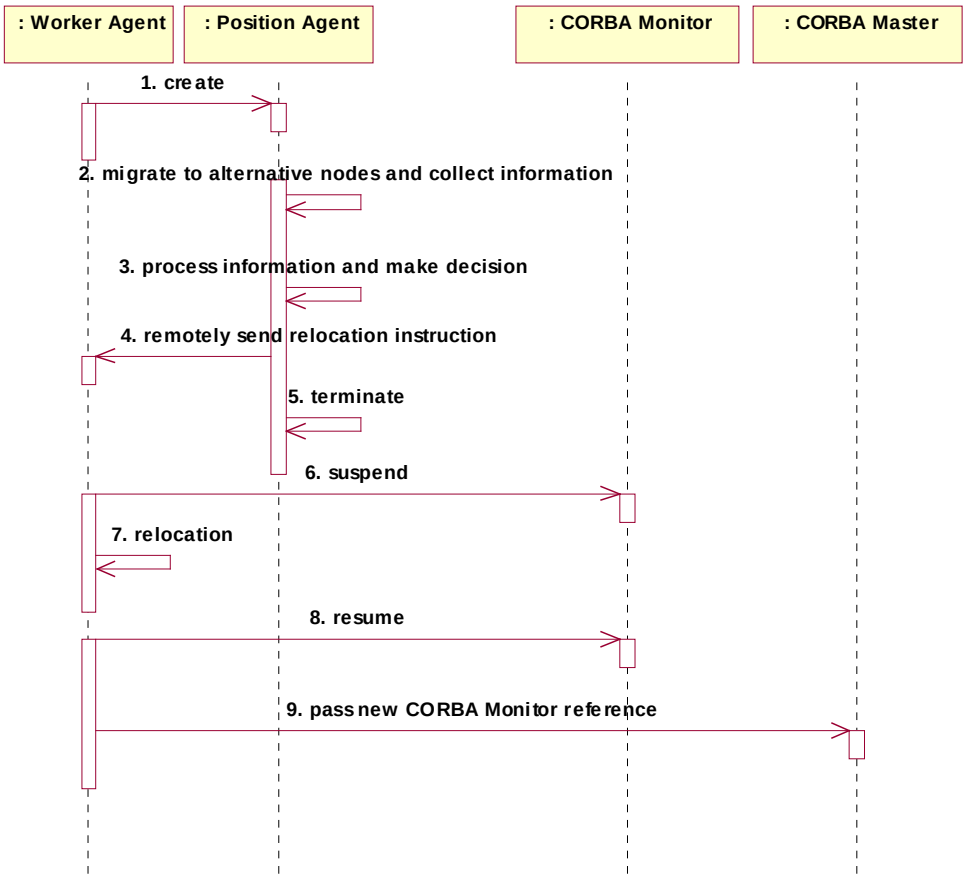


Figure 7-4: A typical relocation scenario in the Hybrid system

Further to this initial scenario, the Worker agent periodically creates a Position agent to examine the possibility of relocating in a more efficient node (see Figure 7-4). The created Position agent visits in a weak mobility manner a number of alternative nodes and collects information on their conditions at each hop. The information collected at each node is provided by an enhanced stationary Target agent that allows access to the local memory utilization, network delay to the managed node, and number of executing agent threads. When the Position agent finishes its task

at the last visited node it analyses the collected information and makes a decision on whether relocation is beneficial. The analysis places importance on network delay to the managed node in order to ensure efficient performance monitoring at the alternative location. If the requirement for low network delay between the alternative node of execution and the managed node is met by a number of nodes then the selection logic takes into account the nodes with low memory utilization and subsequently makes a decision based on the number of executing agent threads as a final factor. The Position agent's logic also considers the conditions in the current node of execution in relation to the best alternative candidate and will only instruct relocation if the improvement in the alternative node is substantial enough to be desirable. If a relocation decision is made, the Position agent remotely instructs the Worker on the suitable relocation node to migrate and finally terminates. The Worker subsequently suspends the currently running monitor and migrates to the designated node. Upon arrival it initiates the CORBA monitor that can resume its operation from the previously suspended session information. The Worker finally needs to notify the Master of the new Monitor reference in order to allow again the direct communication between the two CORBA objects.

The system's approach has allowed us to exploit a number of mobile agent capabilities allowing for dynamic and programmable performance monitoring. In the section that follows we will present experimental results showing the improvements in performance introduced by the collaboration of agents with static distributed objects.

7.4 Assessment

7.4.1 Methodology and Environment

The methodology and environment used for the experimental assessment follows the descriptions given in part 5.4.2. In these lines, traffic and response time measurements involve the remote transfer of reports of increased size and the migration of a Worker for performance monitoring. The memory usage at a managed node was measured as an indicator of computational resource requirements. Finally, source code sizes are compared as an indication of development overheads. The graphs and descriptions presented in the following sub-sections focus on the hybrid system. For comparison purposes data presented in Chapter 5 on the performance of Java-RMI, CORBA, Grasshopper and Jasmin Script-MIB are reproduced here.

7.4.2 Results

7.4.2.1 Remote Data Transfers

Regarding the reporting and notification operations, the hybrid system's approach of communicating CORBA servers means that the system enjoys the low remote communication overheads of CORBA. In Figure 6-2 we can see that in the same levels with CORBA, the hybrid approach can offer very low response times comparable to those incurred by the CodeShell based system and about 5 times better than the Grasshopper-only system. In addition, Figure 6-3 shows that the hybrid approach incurs around 30% more traffic overheads than the CodeShell system.

7.4.2.2 Software Migration

The hybrid system measurement results regarding the required time and traffic associated with the migration of a Worker entity can be seen in Figure 7-5 and Figure 7-6 respectively.

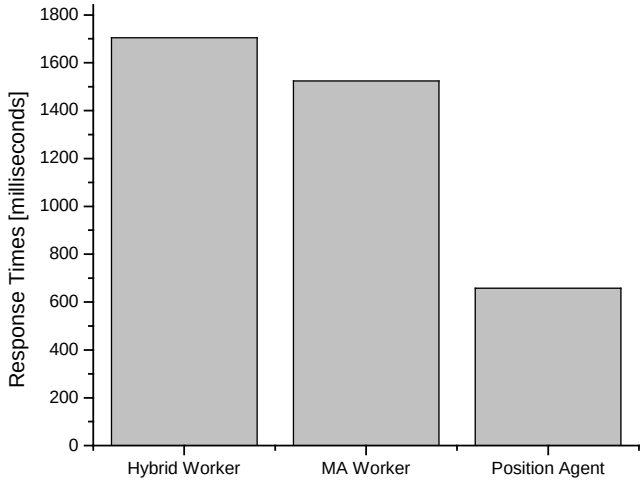


Figure 7-5: Comparison of Hybrid system mobile agents migration times

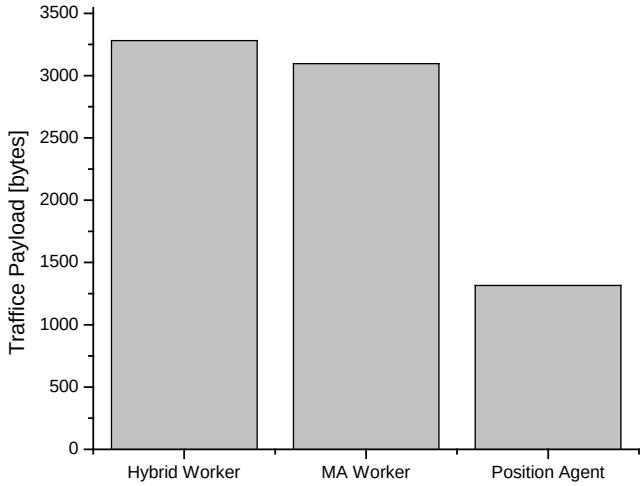


Figure 7-6: Comparison of Hybrid system mobile agents migration traffic

The Worker Agent of the hybrid system gave slightly higher overheads compared to the Worker Agent of the Mobile Agents only system. This is due to the additional CORBA specific code required in the entities of the hybrid system. The Position Agent is common for both systems and much smaller than the Worker Agent resulting in significantly smaller performance overheads that make it suitable for the Weak mobility model. For comparison with the performance overheads for software migration reported above, in a CORBA distributed objects system the creation of a distributed object through a factory requires typically less than 15ms to complete and incurs around 500 bytes of traffic [Pavl98].

7.4.2.3 Memory Usage at Managed Node

The results regarding the JVM memory usage at a managed node can be seen in Figure 7-7.

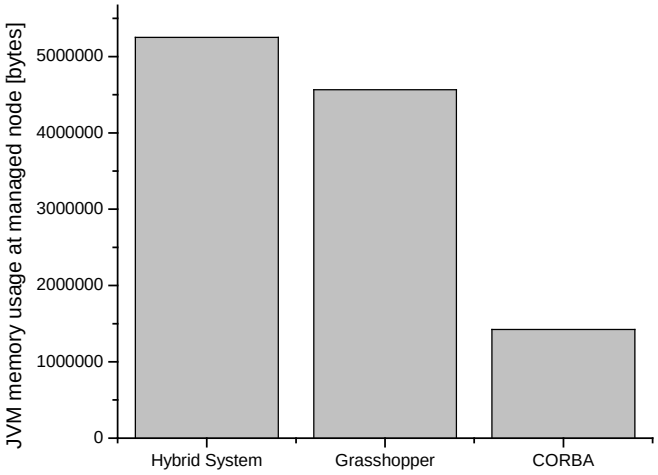


Figure 7-7: Comparison of Hybrid system JVM memory usage

The hybrid approach requires the most amount of memory as it combines both CORBA and mobile agent entities and supporting platform facilities. We have to note here that the reported values of total memory regarding the hybrid system involve the total of two JVMs used, one for the CORBA target object and one for the Grasshopper execution environment including the performance monitoring entities within. A DCN network consisting of workstations dedicated to the management of network nodes can be deployed in order to address the significant memory usage requirements of the Hybrid and Grasshopper systems.

7.4.2.4 Software Development Metrics

The results regarding the source code size required for the development of a hybrid performance monitoring system can be seen in Figure 7-8.

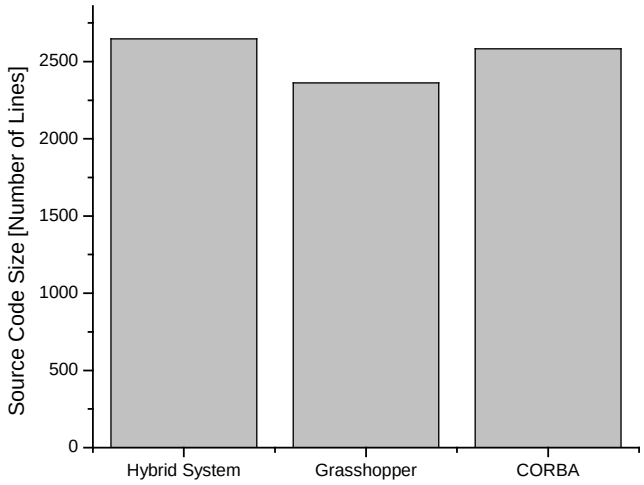


Figure 7-8: Comparison of Hybrid system source code size

The hybrid system includes code that is both CORBA and mobile agent platform specific. As such it contains more code than the Grasshopper and CORBA only systems. While the difference with the CORBA only system is small, the high level of abstraction of the Grasshopper APIs means that a significantly lower source code size is achieved. Compared to the 2648 lines required for the hybrid system, the Grasshopper only system required 2363 lines of code resulting in a significant difference of 285 lines.

7.5 Conclusions

In this chapter we presented the approach, design and evaluation of a hybrid network management solution based on a combination of mobile agents and CORBA. A demonstrating system for network performance monitoring illustrates the approach used to exploit mobility and autonomy

of agents as well as the good performance levels offered by static CORBA objects. An important result arising from this work is that we can easily enhance an existing CORBA system with mobile agent capabilities, through a small alteration of the static CORBA objects at the network management server and no changes necessary to the CORBA servers code executing in the managed nodes. In our case study for performance monitoring small alterations were made to the Master of a CORBA only system in order to create and communicate with the Worker Agent while the code of the CORBA Monitor required no modifications. Constrained mobility of agents provides an easy and effective approach to programmability of management logic at network nodes. Through our practical experience we have identified as a rule of thumb that agents exhibiting constrained mobility can be used to replace CORBA factories located at a managed node. Programmability ensures a flexible system that evolves quickly along with changing network management requirements, an element that was missing from the TMN systems of the mid 90's and hindered their success. As shown by our experimental results the hybrid approach resulted in a significant reduction of performance overheads in the network for typical management operations compared to a mobile agents only system and in similar levels with CORBA. Naturally, agent migration is an important additional overhead compared to CORBA systems but the constrained mobility strategy of long-term execution in a network node helps to ease the migration overheads concerns. In addition in our proposed approach weak mobility of agents was exploited efficiently by ensuring that the size of the Position Agent was small. Our hybrid system approach also exploits the autonomous nature of mobile agents allowing us to fine-tune the system's operation by efficiently positioning our agents and balancing the usage of resources in the managed network (i.e. achieving intelligent adaptation to the environment as described in Chapter 4).

Chapter 8

8 Summary and Conclusions

This final chapter provides the epilogue of this thesis. The summary provided aims to present the overall picture of the thesis approach and issues examined. A discussion of thesis contributions highlights the approaches followed, the work performed and the extend reached towards the thesis objectives appearing in Chapter 1. This is followed by a section on the thesis conclusions presenting the main findings and results as well as their impact to network management. Finally, we discuss a number of future research directions identified through this thesis work.

8.1 Thesis Summary

The thesis focuses on the investigation of the role of mobile agents in the area of network performance management. This is reflected in the research hypothesis of this thesis:

"Software mobile agents can provide an easy means for programmability of managed nodes mainly by following a 'Constrained' migration strategy to a single destination node. This can influence significantly the capabilities of a network performance management system operating within the context of currently envisioned network applications. Programmability can be effectively retained in agent systems that are based on execution environments specially formulated for improved network performance."

A theoretical study on agent mobility importantly unveiled a 'Constrained' strategy for mobile agent based network management that combines a single hop migration intention with autonomous behaviour. Through this strategy, a number of mobile agent benefits important to performance management were identified, relating primarily to the easy support for programmability of network elements and the autonomous, self-configurable and fault tolerant agent operation.

The benefits of constrained mobility for mobile agent-based network management were manifested in three different scenarios of network performance management. The constrained mobility strategy is well suited for long-term management tasks (a particularly common case) and in fact it was the strategy of choice in all three case studies, when a combination of mobility and autonomy was appropriate. The mobile agent-based performance management capabilities of the case studies were thoroughly assessed through a number of experiments and measurements of

system performance highlighting the performance issues of mobile agents as compared to distributed objects and mobile code approaches. An important drawback identified is that the advanced capabilities of modern mobile agent frameworks typically incur significant performance overheads. In the direction of addressing this drawback, the thesis proposes two network management solutions based on specially formulated execution environments that retain important mobile agent benefits while reducing network performance overheads. In the first approach, the capabilities of a distributed object framework are extended using a minimalist approach providing only those facilities required for the support of the constrained agent mobility strategy. In the second approach, distributed objects and mobile agents collaborate to form a hybrid performance monitoring system that combines the good network performance of distributed objects with programmable and autonomous capabilities of mobile agents. The results on the performance overheads of the two approaches were favourable and this reinforces their position as solutions that assist the network management evolutionary path from static distributed objects towards fully featured mobile agent frameworks.

8.2 Discussion of Thesis Contributions

8.2.1 Identification of agent migration strategies for network management

The methodology used for this task was based on a theoretical analysis of network management requirements and mobile agent characteristics in order to derive suitable migration strategies exploiting agent autonomy and mobility. For each strategy identified, the differences from previous approaches based on mobile code were identified and guidelines were given on the practical suitability and expected benefits. A constrained mobility strategy for mobile agent-based network management is the most promising approach identified and motivated more in depth work in this direction.

8.2.2 Assessment of mobile agents within case studies of performance management

With strategies for mobile agent-based network management clearly defined the work focused on three case studies involving performance management that would allow the practical assessment of the role and suitability of mobile agents. The requirements of all three case studies led to the selection and application of a constrained mobility strategy as the most suitable for providing long term network performance monitoring capabilities in an autonomous and programmable manner. The case studies also allowed for the practical identification of system benefits that arise from the use of mobile agents as well as the disadvantages from their use. Both advantages and

disadvantages from the use of mobile agents are clearly contrasted with alternative approaches based on mobile code and distributed objects.

8.2.3 Evaluation of mobile agents as compared to alternative modern approaches

The work produced a thorough evaluation of performance issues of mobile agents based on a common denominator of performance monitoring capabilities taken from the performance monitoring systems of the three case studies. Our evaluation methodology involved four performance monitoring systems based on mobile agents (Grasshopper), mobile code (Jasmin Script-MIB) and distributed objects (CORBA and Java-RMI) all with similar functionality. A number of measurements were taken over a dedicated network testbed highlighting the relative performance issues of each system. In addition, software development parameters were considered as an indication of the complexity of working with each approach.

8.2.4 Proposal of efficient mobile agent-based solutions for network management

In view of the comparative measurements of system performance showing a significant performance overhead for the mobile agent-based system, the work was directed to the investigation of mobile agent solutions that address this drawback.

A first proposal enhances the capabilities of a distributed object platform in order to support constrained mobile agents. This was realized through the development of a prototype platform for constrained mobility (named the CodeShell). The performance monitoring capabilities available were ported to operate over the CodeShell platform and allowed for the successful demonstration of the mobile agents benefits identified previously based on the constrained mobility strategy. Further experimental measurements were taken in an effort to thoroughly assess the performance gains of the CodeShell-based performance monitoring solution.

The second proposal is based on the idea of combining the capabilities of a typical, modern mobile agent and distributed object frameworks through a hybrid performance monitoring system of collaborating mobile agents and distributed objects. The work attempts to exploit the best characteristics of these frameworks in terms of autonomous/self-configurable/programmable operation of mobile agents and low performance overheads of distributed objects. Such a hybrid performance monitoring system was successfully developed and demonstrated based on collaborating Grasshopper mobile agents and CORBA static distributed objects. As before, a number of measurements over a real network testbed environment were taken in order to assess the performance and effectiveness of this system.

8.3 Conclusions

8.3.1 Mobile Agent Strategies

The study of various mobile agent strategies for network management was key to the identification of effective models for the exploitation of mobility and autonomy in agent systems. The proposed strategies involve agents equipped with a 'weighted' level of computational intelligence in order to minimize performance overheads during operations considered as potential bottlenecks.

Constrained mobility is the most important strategy for mobile agent-based network management identified by this thesis. The strategy involves a minimum of three agents, two stationary and one mobile that can together provide an effective network management system combining autonomous operation and a single-hop migration to the managed node. The single-hop migration of Constrained mobility makes it suitable for the majority of network management scenarios, typically involving long term tasks where programmability and autonomous operation is desirable. Programmability can be easily achieved, by creating an updated version of a mobile agent that migrates and replaces the agent already executing at a managed node. In comparison to the strategies involving multi-hop mobility, the single-hop migration and long term execution of a task associated with Constrained mobility means that any agent migration overheads have small impact on the managed network over time. From another viewpoint, we have seen that while in the strategies involving multi-hop mobility it is crucial that an agent carries small amounts of computational intelligence in order to keep migration overheads at modest levels, Constrained mobility allows flexibility for incorporating more functionality within a single-hop agent. In overall the author's findings support the first sentence of the thesis statement: *"Software mobile agents can provide an easy means for programmability of managed nodes mainly by following a 'Constrained' migration strategy to a single destination node."*

The Weak and Strong mobility strategies, involving an autonomous agent system that relies on a mobile agent migrating and executing in a number of nodes, were found to be best suited for short-term management tasks for which there is no existing management support within the targeted network elements. We have seen that regarding Weak mobility there is some limited application in current and envisioned network management systems (e.g. [Liot01b], the Location Agent capabilities described in Chapter 7, etc). On the other hand, although theoretically Strong mobility makes sense, in practice and through the author's work on case studies of network management, no hard requirements were encountered, necessitating its use.

8.3.2 Case Studies of Mobile Agent-based Performance Management

The case study scenarios showed that there are a number of recent requirements for programmable and autonomous network management capabilities. In the case study of "ATM-based Active VPN management", the enterprise that uses the VPN could customize management capabilities of agents to match their specific service needs and expects the agent system to re-configure its management tasks according to factors such as changing network conditions or service requirements. In the case study of "QoS Configuration and SLA Auditing of IP-DiffServ Networks", mobile agents would self-configure with the appropriate network management functionality modules according to the specific network technology identified. Autonomous self-configuration was also particularly important in the highly dynamic VHE environment, where network management functionality needs to be deployed by the VHE system itself in a tailor-made fashion according to the network technology and service requirements, when and where needed. The Constrained mobility strategy was selected and used in all three case studies, as it was the most suitable for the network management requirements encountered, involving long-term tasks for which programmability and autonomy were desirable. Unfortunately, the advanced capabilities of modern mobile agent frameworks are typically associated with additional performance overheads that may not always be acceptable. The relevant experimental results showed that the mobile agent-based approach performed poorly compared to the alternative systems in comparison. Modern mobile agent platforms aim to be general purpose, all round solutions by aggregating agent-related functionality, most of which, was never required within the network management context of our case studies. All that was identified as necessary for the creation of our network management systems was the required facilities for constrained mobility. The assessment of mobile agent technology through the case studies helped the author confirm the validity of the second sentence in the thesis statement: *"This* (Constrained Mobility) *can influence significantly the capabilities of a network performance management system operating within the context of currently envisioned network applications."*

8.3.3 Proposals for Efficient Constrained Mobile Agents

The thesis importantly presents a lightweight approach to the facilities required for the realization of constrained mobile agents. The proposed prototype platform suitably exploits and enhances the efficient remote communication and naming facilities already available in a distributed object framework. This is in contrast with typical mobile agent platforms that provide their own communication and naming facilities, usually large and complex functional blocks burdened with the task of location-transparent communication between multi-hop agents. The extension of a distributed object framework to support single-hop, constrained mobile agents proved to be a much simpler task, suggesting that constrained mobility can be easily integrated in this manner in

network management systems. A thorough performance evaluation of a performance monitoring system based on the platform, established that it is possible to realise management systems following constrained mobility that at the same time achieve performance levels comparable to distributed object systems. The CodeShell platform proved to perform many magnitudes better than Grasshopper, while still offering all the advantages from constrained mobile agents, important for network management. Furthermore, compared to the CodeShell, the Script-MIB mobile code system provides less (i.e. limited capabilities for pushing and initialising a server object), while also incurring greater performance overheads.

As mentioned above, the lightweight CodeShell platform was created as an enhancement to the capabilities of a distributed object framework to support constrained mobility but in some cases the possibility of integrating the capabilities of an existing distributed object framework and a mobile agent platform may be desirable. A 'hybrid' system for performance monitoring consisting of collaborating mobile agents and CORBA servers illustrated the potential of the approach by exploiting mobility and autonomy of agents as well as the good performance levels offered by static CORBA objects. An important result arising from this work is that we can easily enhance an existing CORBA system with mobile agent capabilities, through a small alteration of the static CORBA objects at the network management server and no changes necessary to the CORBA servers code executing in the managed nodes. In our case study for performance monitoring, small alterations were made to the Master of a CORBA only system in order to create and communicate with the Worker agent, while the code of the CORBA Monitor required no modifications. Through our practical experience we have identified as a rule of thumb that agents exhibiting constrained mobility can be used to replace CORBA factories located at a managed node. Programmability ensures a flexible system that evolves quickly along with changing network management requirements, an element that was missing from network management systems based on early standards and hindered their success. As shown by our experimental results the hybrid approach resulted in a significant reduction of performance overheads in the network for typical management operations compared to a mobile agents only system and in similar levels with CORBA.

The two proposals presented allow the exploitation of benefits from constrained mobility in an efficient manner and provide solutions that smooth the evolutionary transition of network management systems from static distributed objects towards, eventually fully featured mobile agent frameworks. This work allowed us to support the last sentence of the thesis statement: "*Programmability can be effectively retained in agent systems that are based on execution environments specially formulated for improved network performance.*"

8.4 Future Directions

The thesis work on mobile agent-based performance management opens several new venues of research. We focus on additional work that would enhance and complement the thesis. The future research directions presented below are not necessarily presented in order of importance.

8.4.1 Quality Metrics for the Assessment of Autonomy

This thesis applied a number of "quantitative" network performance metrics in order to assess the efficiency of constrained mobile agents. Further to this, there is a need for the specification of suitable metrics that would indicate "qualitatively" the impact of autonomy compared to a non-autonomous system (e.g. for the evaluation of the autonomous capabilities presented in Section 4.3.2 regarding intelligent adaptation to the environment). Such metrics should be used in order to accurately assess the impact of autonomy for varying levels of computational intelligence.

8.4.2 Investigation of the limits of "Weighted" Intelligence

The thesis produced work that contained basic autonomous behaviour that is important to network management tasks (i.e. self-configuration capabilities, selection of migration node, autonomous adaptation to encountered errors in performance management operation as described in Section 4.3). Also assisted by the use of "quality" metrics for autonomous behaviour as mentioned above it would be interesting to investigate behaviours that offer an increment in intelligence complexity. In this direction it would be beneficial to know the effective limits of "weighted" computational intelligence. As complexity increases, new considerations arise and need to be investigated such as the element of trust between the user and the mobile agent system.

8.4.3 Standard Protocol of Collaboration of Network Management Agents

This thesis presented a "lightweight" approach to constrained mobility that efficiently allows agents to communicate either via pre-defined interfaces or generic messages. The specification of a standard protocol for the collaboration between network management agents in the Worker, Master and Target roles would allow the creation of systems comprised of agents developed by different vendors.

8.4.4 Mobile Agents for the Management of Mobile Ad-hoc networks

While the thesis studied the role of mobile agents for the management of fixed network infrastructures, mobile ad-hoc networks have no central management station and impose different requirements for network management capabilities and operation. An ad-hoc network is a highly

dynamic network environment that could benefit from programmable and autonomous operation of management entities in the form of mobile agents. In this environment, network management tasks should consider an autonomously selected group of network nodes, with any node able to act in the role of a managed node and a manager, as required. For instance, when a node in the manager role is about to disconnect, any required functionality and state can migrate to an autonomously selected successor node. With the managed nodes of the group notified of this change the management operation can subsequently continue. Whether Constrained mobility is still the most suitable and efficient strategy for the required management tasks within this context or Weak/Strong mobility have a new opportunity is an issue to be investigated.

8.4.5 Exploitation of ACL and XML for Flexible External Negotiation

The Constrained mobility strategy can be effectively delivered without the use of ACL-based messages through carefully designed interfaces. Despite that, as the functionality exposed by a Master agent to the user becomes more extended and complex the use of ACL and XML based messages may become beneficial. The approach would allow for greater flexibility in collaboration between the Master agent and a client of the system by separating the syntax and data of collaboration outside the code within XML documents.

Bibliography

[3GPP00] 3rd Generation Partnership Project (3GPP), Technical Specification 22.121 v4.0.0, *"The Virtual Home Environment (Release 4)"*, October 2000.

[Abdu99] H. Abdu, H. Lutfiyya and M. Bauer, *"A model for adaptive monitoring configurations"*, IM 1999 - IFIP/IEEE International Symposium on Integrated Network Management, no. 1, pp. 371-384, May 1999.

[Acha97] A. Acharya and M. Ranganathan, J. Saltz, *"Sumatra: A Language for Resource-Aware Mobile Programs"*, Mobile Object Systems, J. Vitek and C. Tschudin, eds., Springer Verlag, pp. 111-130, April 1997.

[AGLETS] Aglets Mobile Agent Platform, http://www.trl.ibm.com/aglets/

[Alme99a] G. Almes, S. Kalidindi, M. Zekauskas, *"RFC2679 – A One-way Delay Metric for IPPM"*, The Internet Society, September 1999.

[Alme99b] G. Almes, S. Kalidindi, M. Zekauskas, *"RFC2681 – A Round-trip Delay Metric for IPPM"*, The Internet Society, September 1999.

[Altm01] J. Altmann, F. Gruber, L. Klug, W. Stockner, E. Weippl, *"Using Mobile Agents in Real World: A Survey and Evaluation of Agent Platforms"*, Proceedings of Autonomous Agents 2001, 2nd Workshop on Infrastructure for Agents, MAS, and Scalable MAS, 2001.

[APRIL] Agent Process Interaction Language (APRIL), Imperial College London,

http://www-lp.doc.ic.ac.uk/UserPages/staff/klc/klc.html

[Bald97] M. Baldi, S. Gai, G. P. Picco, *"Exploiting Code Mobility in Decentralized and Flexible Network Management"*, Proceedings of the First International Workshop on Mobile Agents, Berlin, Germany, April 1997.

[Bies98a] A. Bieszczad, B. Pagurek, T. White, *"Mobile Agents for Network Management"*, IEEE Communication Surveys, vol. 1, no. 1, pp. 2-9, September 1998.

[Bies98b] A. Bieszczad, B. Pagurek, T. White, *"Mobile Agents for Network Management"*, IEEE Communications Surveys, Fourth Quarter 1998, vol. 1, no. 1, pp. 2-9, 1998.

[Boho00a] **C. Bohoris**, G. Pavlou, H. Cruickshank, *"Using Mobile Agents for Network Performance Management"*, Proceedings of the IFIP/IEEE Network Operations and Management Symposium (NOMS '00), Hawaii, USA, J. Hong, R. Weihmayer, eds., pp. 637-652, IEEE, April 2000.

[Boho00b] **C. Bohoris**, A. Liotta, G. Pavlou, *"Software Agent Constrained Mobility for Network Performance Monitoring"*, Proceedings of the 6th IFIP Conference on Intelligence in Networks (SmartNet 2000), Vienna, Austria, ed. H.R. van As, pp. 367-387, Kluwer, September 2000.

[Boho00c] **C. Bohoris**, A. Liotta, G. Pavlou, *"Evaluation of Constrained Mobility for Programmability in Network Management"*, In Services Management in Intelligent Networks, Proceedings of the 11th IEEE/IFIP International Workshop on Distributed Systems: Operations and Management (DSOM '00), Austin, Texas, USA, A. Ambler, S. Calo, G. Kar, eds., pp. 243-257, Springer, December 2000.

[Boho02] **C. Bohoris**, A. Liotta, G. Pavlou, *"A Hybrid approach to Network Performance Monitoring based on Mobile Agents and CORBA"*, Proceedings of the 4th International Workshop on Mobile Agents for Telecommunication Applications (MATA '02), Barcelona, Spain, A. Karmouch, T. Magedanz, J. Delgado, eds., pp. 151-162, Springer, October 2002.

[Boho03] **C. Bohoris**, G. Pavlou, A. Liotta, *"Mobile Agent-based Performance Management for the Virtual Home Environment"*, Journal of Network and System Management (JNSM), vol. 11, no. 2, pp. 133-149, Plenum Publishing, June 2003.

[Breu98] M. Breugst, I. Busse, S. Covaci, T. Magedanz, *"Grasshopper - A Mobile Agent Platform for IN Based Service Environments"*, IEEE Intelligent Networks Workshop, Bordeaux, France, May 1998.

[Cari98] W. Caripe, G. Cybenko, K. Moizumi, R. Gray, *"Network Awareness and Mobile Agent Systems"*, IEEE Communications Magazine, no. 7, pp. 44-49, July 1998.

[Carz97] A. Carzaniga, G. P. Picco, G. Vigna, *"Designing Distributed Applications with Mobile Code Paradigms"*, Proceedings of the 19th International Conference on Software Engineering (ICSE'97). p. 22-32, May 1997.

[Case90] J. Case, M. Fedor, M. Schoffstall, J. Davin, *"A Simple Network Management Protocol (SNMP)"*, IETF RFC 1157, 1990.

[CEMF] Cisco Systems, Element Management Framework (EMF) CORBA Gateway, http://www.cisco.com/warp/public/cc/pd/nemnsw/emf/prodlit/crba_ds.htm

[Chai92] B. Chaib-draa, B. Moulin, R. Mandiau, P. Millot, *"Trends in Distributed Artificial Intelligence"*, Artificial Intelligence Review, vol. 6, pp. 35-66, 1995.

[Chat00] F. G. Chatzipapadopoulos, M. K. Perdikeas, I. S. Venieris, *"Mobile Agent and CORBA Technologies in the Broadband Intelligent Network"*, IEEE Communications Magazine, Vol. 38, No. 6, pp. 116-124, June 2000.

[Chei98] M. Cheikhrouhou, P. Conti, J. Labetoulle, *"Intelligent Agents in Network Management, a state of the art"*, Networking and Information Systems, vol. 1, no. 1, pp. 9-38, 1998.

[Chei00a] M. Cheikhrouhou and J. Labetoulle, *"When management agents become autonomous, how to ensure their reliability?"*, NOMS 2000 - IEEE/IFIP Network Operations and Management Symposium, no. 1, pp. 949-950, September 2000.

[Chei00b] M. Cheikhrouhou, P. Conti, J. Labetoulle, *"Automatic configuration of PVCs in ATM networks with software agents"*, IEEE/IFIP Network Operations and Management Symposium (NOMS'00), no. 1, pp. 535-548, September 2000.

[Clim97] Climate Cluster, http://www.fokus.gmd.de/research/cc/ecco/climate/

[CORB95] Object Management Group, *"The Common Object Request Broker: Architecture and Specification (CORBA)"*, Version 2.0, 1995.

[DAGE] D' Agents Mobile Agent Platform, http://www.cs.dartmouth.edu/~agent/

[Dori96] M. Dorigo, V. Maniezzo, A. Colorni, *"The Ant System: Optimization by a Colony of Cooperating Agents"*, IEEE Transactions on Systems, Man and Cybernetics – Part B, vol. 26, no. 1, pp. 1-13, 1996.

[Elda98] M. El-Darieby, *"Intelligent Mobile Agents for Network Fault Management"*, Technical Report SCE-98-13, System and Computer Engineering, Carleton University, 1998.

[Elda99] M. El-Darieby, A. Bieszczad, *"Intelligent mobile agents: Towards network fault management automation"*, IFIP/IEEE International Symposium on Integrated Network Management (IM'99), no. 1, pp. 611-622, May 1999.

[Feng01] N. Feng, A. Gang, T. White, B. Pagurek, *"Dynamic Evolution of Network Management Software by Software Hot-Swapping"*, In Proc. of the Seventh IFIP/IEEE International Symposium on Integrated Network Management (IM 2001), Seattle, pp. 63-76, May, 2001.

[FIPA01] FIPA, *"FIPA ACL Message Structure Specification"*, version E, October 2001.

[FIPA02] FIPA, *"FIPA Abstract Architecture Specification"*, version J, February 2002.

[Fran96] S. Franklin, A. Graesser, and *"Is it an Agent or just a Program? A Taxonomy for Autonomous Agents"*, Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL '96), Budapest, Hungary, J. P. Muller, M. J. Wooldridge, N. R. Jennings, eds., pp. 21-35, Springer-Verlag, 1996.

[Fugg97] A. Fuggetta, G. P. Picco, G. Vigna, *"Understanding Code Mobility"*, IEEE Transactions on Software Engineering, vol. 24, no. 5, pp. 342-361, 1998.

[Gali02] A. Galis, S. Covaci (eds.), *"Mobile Intelligent Agents Applied to Communication Management Systems"*, CRC Press, ISBN: 0849392012, January 2002.

[Gava99a] D. Gavalas, D. Greenwood, M. Ghanbari, M. O'Mahony, *"Complimentary Polling Modes for Network Performance Management Employing Mobile Agents"*, Proceedings of the IEEE Global Communications Conference (Globecom'99), pp. 401-405, December 1999.

[Gava99b] D. Gavalas, D. Greenwood, M. Ghanbari, M. O'Mahony, *"A Hybrid Centralised - Distributed Network Management Architecture"*, Proceedings of the 4th IEEE Symposium on Computers and Communications (ISCC'99), pp. 434-441, July 1999.

[Gava99c] D. Gavalas, D. Greenwood, M. Ghanbari, M. O'Mahony, *"An Infrastructure for Distributed and Dynamic Network Management based on Mobile Agent Technology"*, Proceedings of the IEEE International Conference on Communications (ICC'99), pp. 1362-1366, June 1999.

[Gava00] D. Gavalas, D. Greenwood, M. Ghanbari, M. O'Mahony, *"Advanced Network Monitoring Applications Based on Mobile/Intelligent Agent Technology"*, Computer Communications Journal, Special issue on Mobile Agents for Telecommunication Applications, vol. 23, no.8, pp. 720-730, April 2000.

[Gava01] D. Gavalas, *"Mobile Software Agents for Network Monitoring and Performance Management"*, PhD Thesis, University of Essex, July 2001.

[Glit02] R. H. Glitho, E. Olougouna, S. Pierre, *"Mobile agents and their use for information retrieval: A brief overview and an elaborate case study"*, IEEE Network, vol. 16, no. 1, pp. 34-41, January 2002.

[Gold96] G. Goldszmidt, *"Distributed Management By Delegation"*, PhD Thesis, Columbia University, 1996.

[GRAS] Grasshopper Mobile Agent Platform, IKV++ Technologies,

http://www.grasshopper.de/

[Gray95] R. S. Gray, *"Agent Tcl: A transportable agent system"*, Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95), Baltimore, Maryland, December, 1995.

[Grif00] D. Griffin, G. Pavlou, P. Georgatsos, *"Providing Customisable Network Management Services Through Mobile Agents"*, Proceedings of the 7th International Conference on Intelligence in Services and Networks (IS&N '00), G. Stamoulis, A. Mullery, D. Prevedourou, K. Start, eds., pp. 209-226, Springer, February 2000.

[Guia01] M. H. Guiagoussou, R. Boutaba, M. Kadoch, *"A Java API for Advanced Faults Management"*, IEEE/IFIP International Symposium on Integrated Network Management (IM'01), pp. 483-498, 2001.

[Guth98] A. Guther, M. Zell, (eds.), *"Platform Enhancement Requirements – Public Deliverable 1"*, ACTS MIAMI project (AC338), 1998.

[Hall97] D. A. Halls, *"Applying Mobile Code to Distributed Systems"*, PhD Thesis, University of Cambridge, 1997.

[Hari95] C. G. Harrison, D. M. Chess, A. Kershenbaum, *"Mobile Agents: Are they a good idea?"* Technical Report, IBM Research Division, Watson Research Center, March 1995.

[Haye95] B. Hayes-Roth, *"An Architecture for Adaptive Intelligent Systems"*, Artificial Intelligence: Special Issue on Agents and Interactivity, 72, pp. 329-365, 1995.

[I610] ITU-T, *"Recommendation I.610, B-ISDN operation and maintenance principles and functions, Revision 2"*, November 1995.

[Isma00] L. Ismail, D. Hagimont, J. Mossiere, *"Evaluation of the Mobile Agent Technology: Comparison with the Client/Server Paradigm"*, Proceedings of the International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'99), Denver, Colorado, U.S.A., November 1999.

[James] James Mobile Agent Platform, http://maria.dei.uc.pt/james2/overview.html

[Java95] Sun Microsystems, *"The Java Language Specification"*, October 1995, http://hava.sun.com/docs/books/jls/index.html

[JAVAB] Sun Microsystems, JavaBeans component framework,

http://java.sun.com/products/javabeans/

[Jasmin] Jasmin Script-MIB Platform, http://www.ibr.cs.tu-bs.de/projects/jasmin/

[Joha95] D. Johansen, R. van Renesse, F. B. Schneider, *"An Introduction to the TACOMA Distributed System – Version 1.0"*, Technical Report 95-23, University of Tromso and Cornell University, June 1995.

[JMX] Sun Microsystems, *"Java Management Extensions (JMX)"*, http://java.sun.com/products/JavaManagement/

[JRMI] Sun Microsystems, *"Java Remote Method Invocation (Java-RMI)"*, http://java.sun.com/products/jdk/rmi/

[Kaha97] M. Kahani, H. Beadle, *"Decentralised Approaches for Network Management"*, Computer Communications Review, vol. 27, no. 3, pp. 36-47, July 1997.

[Kawa00] R. Kawamura and R. Stadler, *"Active distributed management for IP networks"*, IEEE Communications Magazine, vol. 38, no. 4, pp. 114-120, April 2000.

[Kotz99] D. Kotz, R. Gray, *"Mobile Agents and the Future of the Internet"*, in ACM Operating Systems Review, 33(3), pp. 7-13, August 1999.

[Kotz02] D. Kotz, R. Gray, D. Rus, *"Future Directions for Mobile Agent Research"*, Technical Article, IEEE Distributed Systems, vol. 3, no 8, August 2002.

[KQML] Knowledge Query and Manipulation Language (KQML), http://www.cs.umbc.edu/kqml/

[Lang98] D. B. Lange, *"Mobile Objects and Mobile Agents: The Future of Distributed Computing?"*, Proceedings of the European Conference on Object-Oriented Programming (ECOOP'98), 1998.

[Lash94] Y. Lashkari, M. Metral, P. Maes, *"Collaborative Interface Agents"*, Proceedings of the 12th National Conference on Artificial Intelligence, 1994.

[Levi99] D. Levi, J. Schoenwaelder, "*Definitions of Managed Objects for the Delegation of Management Scripts"*, IETF RFC2592, 1999.

[Liot99] A. Liotta, G. Knight, G. Pavlou, *"On the Performance and Scalability of Decentralised Monitoring Using Mobile Agents"*, Proceedings of the 10th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM'99), pp. 3-18, October 1999.

[Liot01a] A. Liotta, G. Pavlou, G. Knight, *"Active Distributed Monitoring for Dynamic Large-scale Networks"*, Proceedings of the IEEE International Conference on Communications (ICC'01), Helsinki, Finland, Vol. 5, pp. 1544-1550, IEEE, June 2001.

[Liot01b] A. Liotta, *"Towards Flexible and Scalable Distributed Monitoring with Mobile Agents"*, PhD Thesis, University College London, July 2001.

[Liot02a] A. Liotta, G. Pavlou, G. Knight, *"Exploiting Agent Mobility for Large Scale Network Monitoring, IEEE Network*", Special issue on Applicability of Mobile Agents to Telecommunications, vol. 16, no. 3, IEEE, May/June 2002.

[Liot02b] A. Liotta, A. Yew, **C. Bohoris**, G. Pavlou, *"Delivering Service Adaptation with 3G Technology"*, Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 2002), Montreal, Canada, Lecture Notes in Computer Science, vol. 2506, pp. 108-120, Springer, October 2002.

[Liot02c] A. Liotta, A. Yew, **C. Bohoris**, G. Pavlou, *"Supporting Adaptation-aware Services through the Virtual Home Environment"*, Proceedings of the Hewlett-Packard Openview University Association Plenary Workshop 2002 (HP-OVUA), June 11-13, 2002.

[M3010] ITU-T, *"Recommendation M.3010, Principles for a Telecommunications Management Network (TMN)"*, Study Group IV, 1996.

[Maes91] P. Maes, *"The agent network architecture (ANA)"*, SIGART Bulletin, 2(4):115-120, 1991.

[Maes95] P. Maes, *"Artificial Life Meets Entertainment: Life like Autonomous Agents"*, Communications of the ACM, 38, 11, pp. 108-114, 1995.

[Mage96] T. Magendaz, K. Rothermel, S. Krause, *"Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?"*, Proceedings of the IEEE Infocom '96, March 1996.

[MANT] Management Testing & Reconfiguration of IP based networks using mobile software agents (MANTRIP) project, IST -1999-10921,

http://www.solinet-research.com/mantrip/

[Marc99] K. Marcus, *"Improving reliability of intelligent agents for network management"*, Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM'99), no. 1, pp. 941-942, May 1999.

[MASI98] Object Management Group, *"Mobile Agent System Interoperability Facility"*, 1998.

[McCl91] K. McCloghrie, M. Rose, *"RFC1213 – Management Information Base for Network Management of TCP/IP-based internets: MIB-II"*, The Internet Society, March 1991.

[McCl94] K. McCloghrie, F. Kastenholz, *"RFC1573 – Evolution of the Interfaces Group of MIB-II"*, The Internet Society, January 1994.

[Mich00] A. Michalas, (ed.), *"System Requirements and High Level Architecture – Public Deliverable 2.1"*, IST MANTRIP project (IST -1999-10921), 2000.

[MIAMI] Mobile Intelligent Agents for Managing the Information Infrastructure (MIAMI) project, ACTS 338,

http://www.cordis.lu/infowin/acts/rus/projects/ac338.htm

[OMG95] Object Management Group, *"The Common Object Request Broker: Architecture and Specification (CORBA)"*, Version 2.0, 1995.

[Oren94] E. Oren, D. Weld, *"A Softbot-Based Interface to the Internet"*, Communications of the ACM, 37, 7, pp. 72-79, 1994.

[Papa00a] T. Papaioannou, *"Mobile Information Agents in Cyberspace: State of the Art and Vision"*, Proceedings of the 4th International Workshop on Cooperating Information Agents (CIA '00), 2000.

[Papa00b] T. Papaioannou, *"On the Structuring of Distributed Systems: The argument for Mobility"*, PhD Thesis, Loughborough University, February 2000.

[PARLAY] The Parlay Group, http://www.parlay.org

[Pavl96] G. Pavlou, G. Mykoniatis, J. Sanchez, *"Distributed Intelligent Monitoring and Reporting Facilities"*, IEE Distributed Systems Engineering Journal (DSEJ), Special Issue on Management, vol. 3, no. 2, pp. 124-135, IOP Publishing, 1996.

[Pavl98] G. Pavlou, *"Telecommunications Management Network: a Novel Approach Towards its Architecture and Realisation Through Object-Oriented Software Platforms"*, PhD Thesis, University College London, March 1998.

[Pavl00] G. Pavlou, A. Liotta, **C. Bohoris**, D. Griffin, P. Georgatsos, *"Providing Customisable Remote Management Sevices Using Mobile Agents"*, Proceedings of the Hewlett-Packard Openview University Association Plenary Workshop 2000 (HP-OVUA), Santorini, Greece, June 12-14, 2000.

[Paxs98] V. Paxson, G. Almes, J. Mahdavi, M. Mathis, *"RFC2330 – Framework for IP Performance Metrics"*, The Internet Society, May 1998.

[Picc98] G. P. Picco, *"Understanding, Evaluating, Formalizing and Exploiting Code Mobility"*, PhD Thesis, Politecnico di Torino, February 1998.

[Puli00] A. Puliafito and O. Tomarchio, *"Using Mobile Agents to implement flexible Network Management strategies"*, Computer Communication Journal, 23(8):708-719, April 2000.

[Raz00] D. Raz and Y. Shavitt, *"Active networks for efficient distributed network management"*, IEEE Communications Magazine, vol. 38, no. 3, pp. 138-143, March 2000.

[Raza98] K. S. Raza, *"Implementation of Plug-and-Play Printer with Mobile Agents"*, Technical Report, SCE-98-04, System and Computer Engineering, Carleton University, 1998.

[Rubi99] M. Rubinstein, O. C. Duarte, *"Evaluating the Performance of Mobile Agents in Network Management"*, Proceedings of the IEEE Global Communications Conference (Globecom'99), December 1999.

[Saha99] A. Sahai, C. Morin, *"Mobile Agents for Managing Networks: MAGENTA perspective"*, Chapter 15 in the book "Software Agents for Future Communications Systems", A. Hayzelden, J. Bigham eds., Springer Verlag, April 1999.

[Scho97] J. Schonwalder, *"Network Management by Delegation – From Research Prototypes Towards Standards"*, 8th Joint European Networking Conference (JENC8), Edinburgh, May 1997.

[Scho99] J. Schonwalder, J. Quittek, *"RFC2593 – Script MIB Extensibility Protocol Version 1.0"*, The Internet Society, May 1999.

[Scho00] J. Schonwalder, J. Quittek, C. Kappler, *"Building Distributed Management Applications with the IETF Script-MIB"*, IEEE Journal on Selected Areas in Communications, vol. 18, no. 5, pp. 702-714, May 2000.

[Schw00] I. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, C. Partridge, *"Smart Packets: Applying Active Networks to Network Management"*, ACM Transactions on Computer Systems, vol. 18, no. 1, pp. 67-88, 2000.

[Silv00] L. Silva, G. Soares, P. Martins, V. Batista, L. Santos, *"Comparing the Performance of Mobile Agent Systems: A Study of Benchmarking"*, Journal of Computer Communications, Special Issue on Mobile Agents for Telecommunication Applications, January 2000.

[Simo02] P. Simoes, J. Rodrigues, L. Silva and F. Boavida, *"Distributed retrieval of management information: Is it about mobility, locality or distribution?"*, Proceedings of the IEEE/IFIP Network Operations and Management Symposium 2002, vol. 8, no. 1, pp. 79-96, April 2002.

[Smit94] D. C. Smith, A. Cypher, J. Spohrer, *"KidSim: Programming Agents Without a Programming Language"*, Communications of the ACM, 37, 7, pp. 55-67, 1994.

[SOAP] Simple Object Access Protocol 1.1, May 2000, http://www.w3.org/TR/SOAP/

[Stam90] J. W. Stamos, D. K. Gifford, *"Remote Evaluation"*, ACM Transactions on Programming Languages and Systems. 12(4), pp. 537-565, October 1990.

[Stra96] M. Straber, J. Baumann, B. Hohl, *"Mole – A Java Based Mobile Agent System"*, Special Issues in Object-Oriented Programming: Workshop Reader of the 10th European Conference on Object-Oriented Programming ECOOP'96, M. Muhlauser, Ed., pp. 327-334, July 1996.

[Tenn96] D. Tennenhouse, D. Wetherall, *"Towards an Active Network Architecture"*, ACM Computer Communication Review, 26(2), pp. 5-18, April 1996.

[Tenn97] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, G. Minden, *"A Survey of Active Network Research"*, IEEE Communications Magazine, vol. 35, no. 1, pp. 80-86, January 1997.

[Tesi99] K. Tesink, *"RFC2515 – Definitions of Managed Objects for ATM Management"*, The Internet Society, February 1999.

[Thot99] M. Thottan, C. Ji, *"Fault prediction at the network layer using intelligent agents"*, IFIP/IEEE International Symposium on Integrated Network Management (IM'99), no. 1, pp. 745-759, May 1999.

[TINA95] The TINA Consortium, *"Overall Concepts and Principles of TINA"*, February 1995.

[TINA97] The TINA Consortium, *"Business Model and Reference Points Version: 4.0"*, May 1997.

[Vayi00] E. Vayias, J. Soldatos, J. Bigham, L. Cuthbert, Z. Luo, *"Intelligent Agents for ATM Network Control and Resource Management"*, Journal of Network and Systems Management, vol. 8, no. 3, pp. 373-395, Plenum Publishers, September 2000.

[VESPER] Virtual Home Environment for Service Personalization and Roaming Users (VESPER) project (2000-2002), IST-1999-10825,

http://www.ee.surrey.ac.uk/CCSR/IST/Vesper/

[Vign98a] G. Vigna, ed., *"Mobile Agents and Security"*, volume 1419 of Lecture Notes in Computer Science, Springer-Verlag, 1998.

[Vign98b] G. Vigna, *"Mobile Code Technologies, Paradigms, and Applications"*, PhD Thesis, Politecnico di Milano, Italy, February 1998.

[VOYA] Voyager Platform, Recursion Software,

http://www.recursionsw.com/products/voyager/voyager.asp

[Wald95] S. Waldbusser, *"RFC1757 – Remote Network Monitoring Management Information Base"*, The Internet Society, February 1995.

[Whit94] J. E. White, *"Telescript Technology: The Foundation for the Electronic Marketplace"*, Technical Report, General Magic, Inc., 1994.

[Whit98a] T. White, B. Pagurek, F. Oppacher, *"Connection Management using Adaptive Agents"*, Proceedings of the 1998 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98), Las Vegas, July 13-16, 1998.

[Whit98b] T. White, B. Pagurek, *"Towards Multi-Swarm Problem Solving in Networks"*, Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS'98), July 1998.

[Whit98c] T. White, A. Bieszczad, B. Pagurek, *"Distributed Fault Location in Networks using Mobile Agents"*, Proceedings of the International Workshop on Agents in Telecommunications Applications (IATA'98), Paris, France, July 1998.

[Wija00] Y. I. Wijata, D. Niehaus, V. S. Frost, *"A scalable agent-based network measurement infrastructure"*, IEEE Communications Magazine, no. 9, pp. 174-183, September 2000.

[Wool95a] M. Wooldridge, N. Jennings, *"Intelligent Agents: Theory and Practice"*, The Knowledge Engineering Review, vol. 10(2), pp. 115-152, 1995.

[Wool95b] M. Wooldridge, N. R. Jennings, *"Agent Theories, Architectures, and Languages: a Survey"*, in Intelligent Agents, Berlin: Springer-Verlag, pp. 1-22, 1995.

[Wren99] M. Wren, J. Gutierrez, *"Agent and Web Technologies in Network Management"*, Proceedings of the IEEE Global Communications Conference (Globecom'99), December 1999.

[X738] ITU-T, *"Recommendation, X.738, Information Technology - Open Systems Interconnection, Systems Management Functions - Summarization Function"*, 1993.

[X739] ITU-T, *"Recommendation X.739, Information Technology - Open Systems Interconnection, Systems Management Functions - Metric Objects and Attributes"*, 1992.

[Yemi91] Y. Yemini, G. Goldszmidt, S. Yemini, *"Network Management by Delegation"*, in Integrated Network Management II, Krishnan, Zimmer, eds., pp. 95-107, Elsevier, 1991.

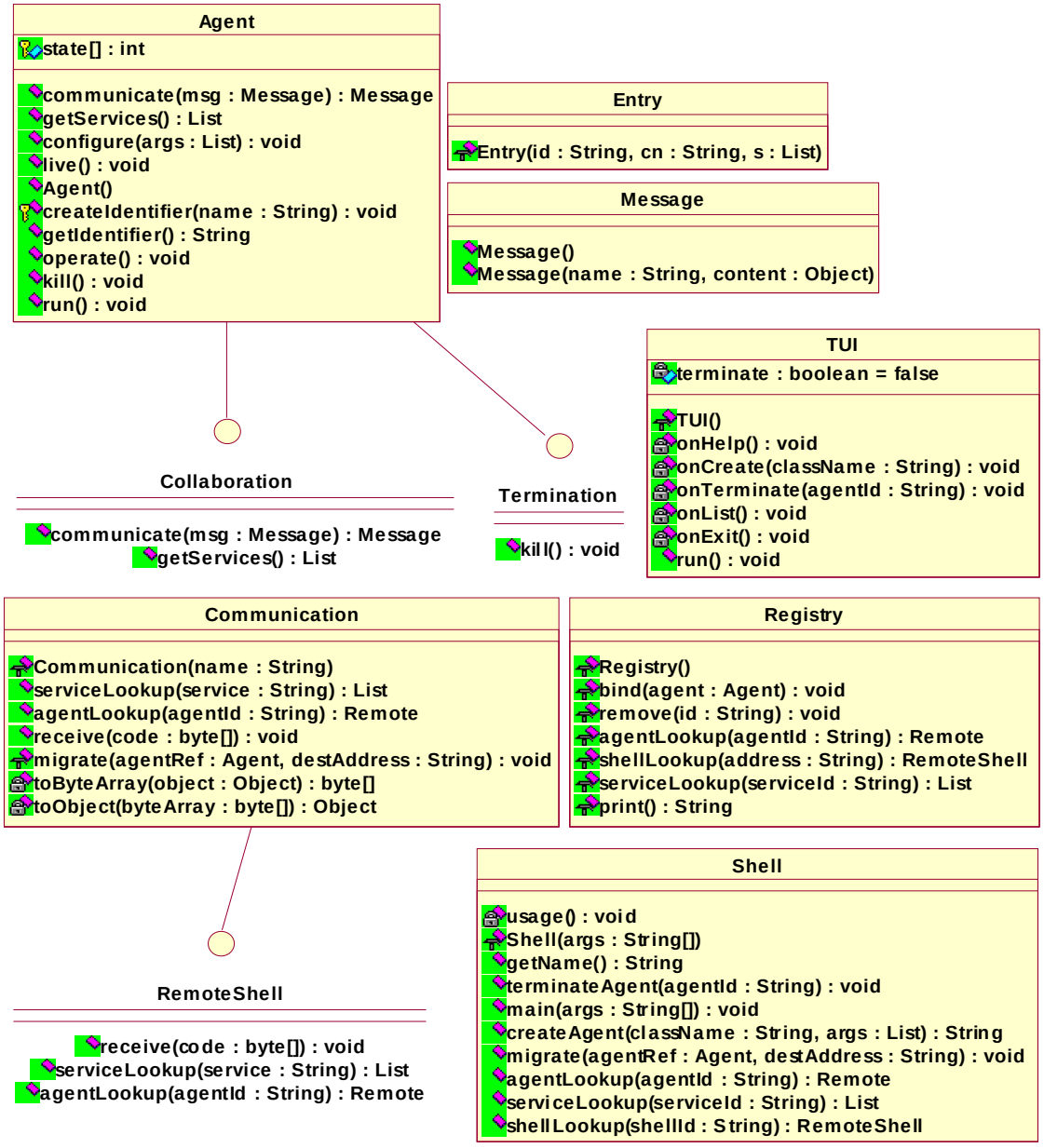
[Yew01] A. Yew, **C. Bohoris**, A. Liotta, G. Pavlou, *"Quality of Service Management for the Virtual Home Environment"*, In the Proceedings of the 12th IEEE/IFIP International Workshop on Distributed Systems: Operations and Management (DSOM '01), Nancy, France, O. Festor, A. Pras, eds., pp. 179-190, INRIA Press, October 2001.

[Yew01b] A. Yew, A.Liotta, **C. Bohoris**, G. Pavlou, *"Towards Quality of Service Aware Services for the Virtual Home Environment"*, Proceedings of the Hewlett-Packard Openview University Association Plenary Workshop 2001 (HP-OVUA), Berlin, Germany, June 24-27, 2001.

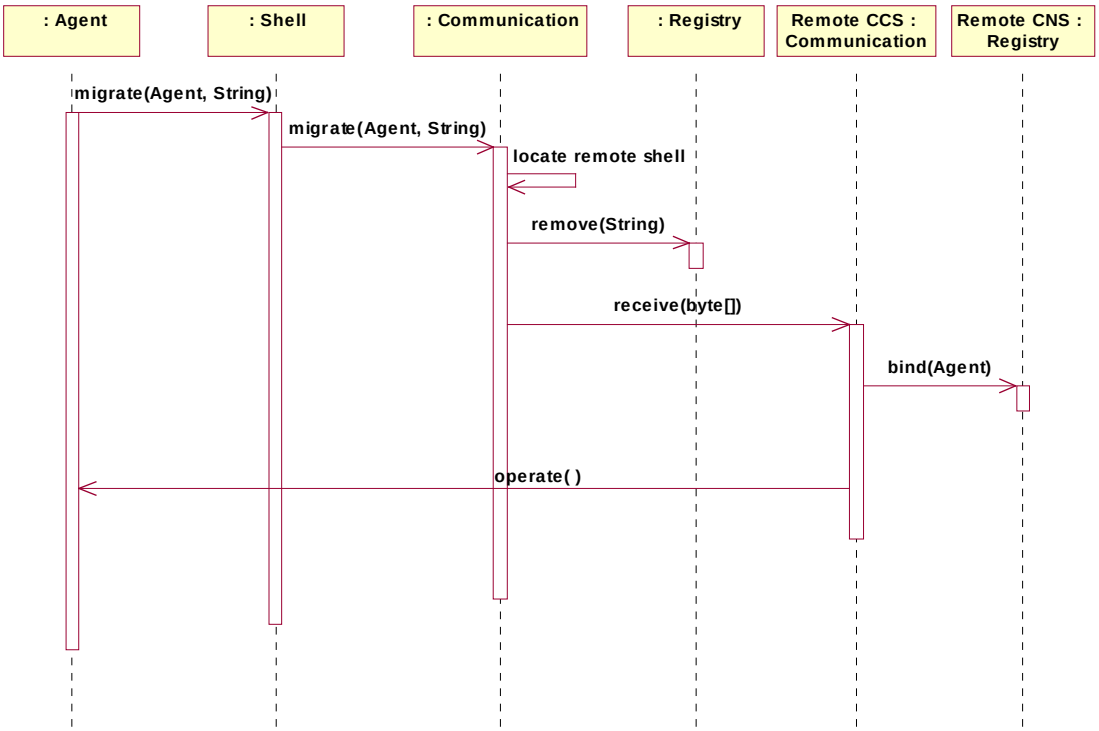
[Zapf99] M. Zapf, K. Herrmann, K. Geihs, *"Decentralized SNMP Management with Mobile Agents"*, Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management (IM'99), pp. 623-635, May 1999.

Appendix A – CodeShell UML Diagrams

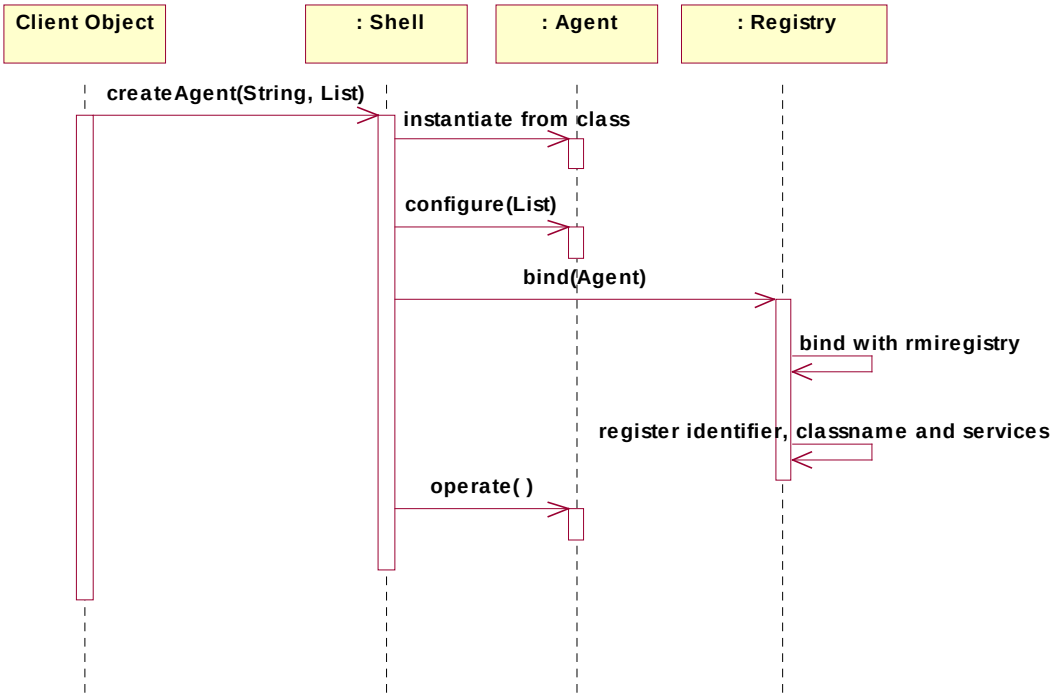
CodeShell Class Diagram



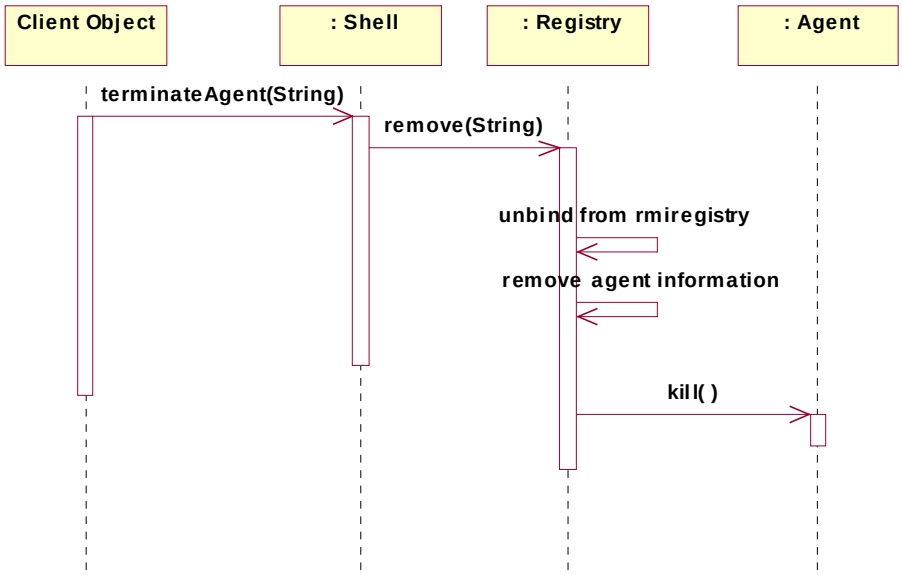
“Agent Migration” Sequence Diagram



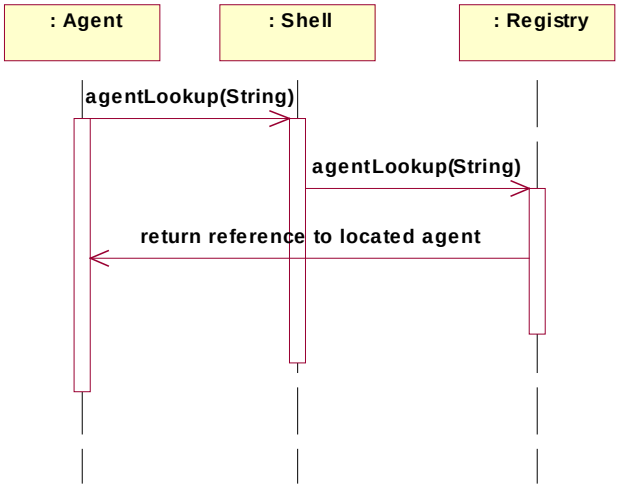
“Agent Creation” Sequence Diagram



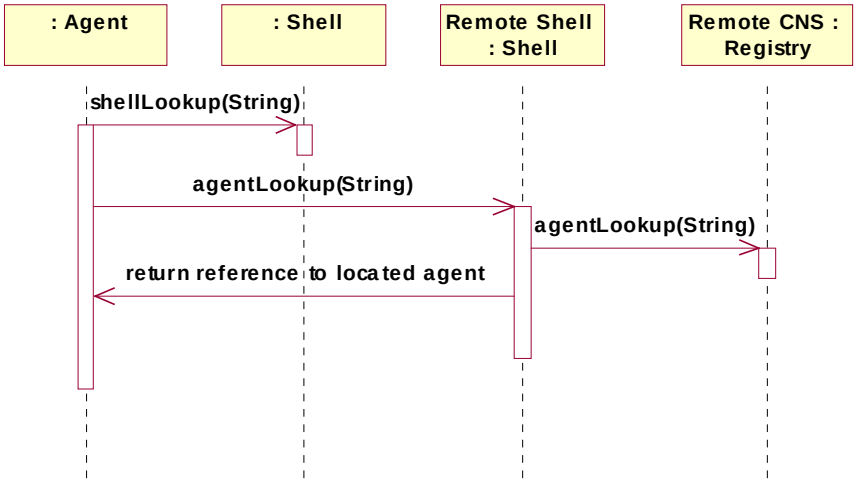
"Agent Termination" Sequence Diagram



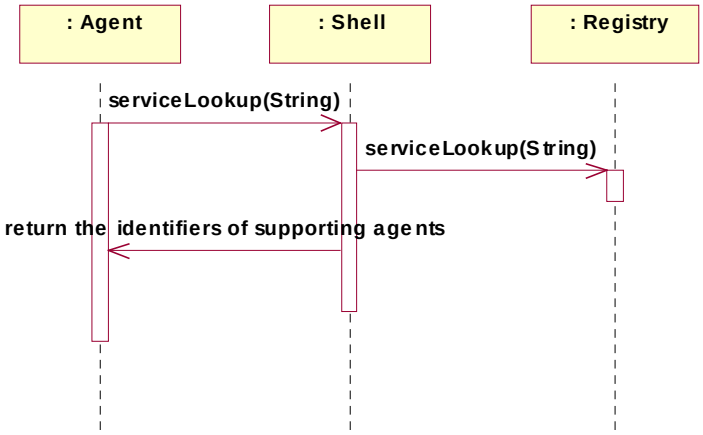
"Agent Lookup" Sequence Diagram



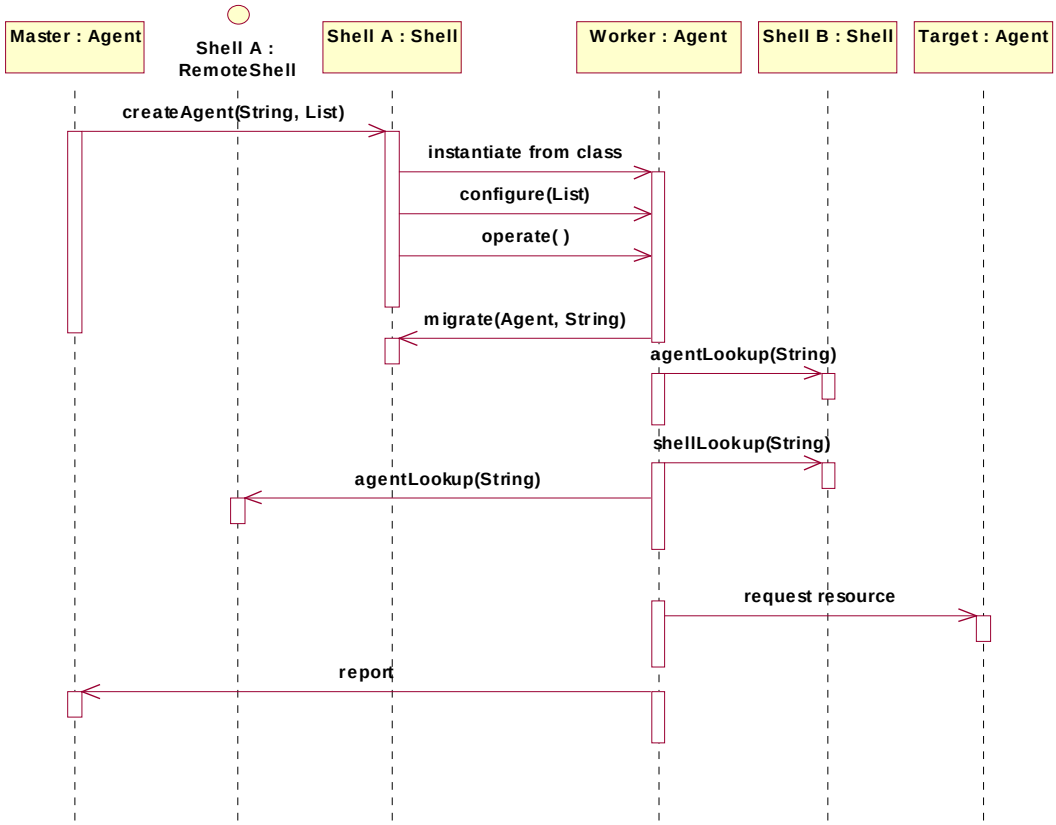
“Remote Agent Lookup” Sequence Diagram



“Service Lookup” Sequence Diagram



"Constrained Mobility Strategy" Sequence Diagram



Appendix B – CodeShell Source Code

Agent.java

```
/* Copyright (C) 2000-2002 Christos Bohoris, University of Surrey */
package uk.ac.surrey.codeshell;

import java.io.*;
import java.util.*;
import java.rmi.*;
import java.rmi.server.*;
import java.text.*;

/**
 * A base agent that must be inherited by all agents executing
 * within a Shell.
 *
 * @author Christos Bohoris
 */
public abstract class Agent extends UnicastRemoteObject implements Serializable, Runnable,
Collaboration, Termination {

    /* A unique identifier. */
    private String identifier;

    /* The agent state. */
    protected int[] state;

    /**
     * Pass a message to an agent.
     *
     * @param msg a message
     * @return a reply message
     */
    public abstract Message communicate(Message msg)
    throws RemoteException;

    /**
     * Get the services offered by an agent.
     *
     * @return typically an ArrayList of String elements reflecting on
     * service names
     */
    public abstract List getServices() throws RemoteException;

    /**
     * Configure a newly created agent. Invoked by the platform using
     * the arguments provided in Shell.createAgent(...)
     *
     * @param a list of initialization arguments
     */
    public abstract void configure(List args);

    /**
     * Specify agent behaviour.
     */
    public abstract void live();

    /**
     * Create a new Agent.
     */
    public Agent() throws RemoteException {
        identifier = null;
    }

    /**
     * Create a unique agent identifier.
     * Agents should invoke this within Agent.configure(...).
     * If a null parameter is passed then a unique name is
     * created automatically.
```

```
     *
     * @param name the agent's identifying name
     */
    protected void createIdentifier(String name) {
        if (identifier == null) {
            if (name == null) {
                String output;
                SimpleDateFormat formatter;

                formatter = new
                SimpleDateFormat("dd.MM.yyyy.HH.mm.ss.SS",
                Locale.getDefault());
                output = formatter.format(new Date());

                identifier = "agent_" + output;
            } else {
                identifier = name;
            }

        }
    }

    /**
     * Get the agent identifier.
     *
     * @return the unique agent identifier
     */
    public String getIdentifier() {
        return identifier;
    }

    /**
     * Initiate the agent's operation.
     */
    public final void operate() {
        Thread thread = new Thread(this, identifier);
        thread.start();
    }

    /**
     * Terminate an agent instance. Shell.terminateAgent(...) should
     * be invoked by agents
     * for appropriate removal.
     */
    public void kill() throws RemoteException {
        try {
            super.finalize();
        } catch (Throwable e) {
            System.out.println("Agent termination failed.\n" +
            e);
        }
    }

    /**
     * Executes Agent.live() within the dedicated agent thread.
     */
    public void run() {
        live();
    }

}
```

Collaboration.java

```
/* Copyright (C) 2000-2002 Christos Bohoris, University of Surrey */
package uk.ac.surrey.codeshell;

import java.rmi.*;
import java.util.*;

/**
 * Allows collaboration between agents offering a number of services.
 *
 * @author Christos Bohoris
 */
public interface Collaboration extends Remote {
```

```
    /**
     * Pass a message to an agent.
     *
     * @param msg a message
     * @return a reply message
     */
    public Message communicate(Message msg) throws RemoteException;

    /**
     * Get the services offered by an agent.
     *
     * @return typically an ArrayList of String elements reflecting on
     * service names
     */
    public List getServices() throws RemoteException;

}
```

Communication.java

```
/* Copyright (C) 2000-2002 Christos Bohoris, University of Surrey */
package uk.ac.surrey.codeshell;

import java.io.*;
import java.util.*;
import java.net.*;
import java.rmi.*;
import java.rmi.server.*;

/**
 * The Codeshell remote communication service.
 *
 * @author Christos Bohoris
 */
class Communication extends UnicastRemoteObject
implements RemoteShell {

    /*
     * Construct a Communication object.
     *
     * @param the shell name
     */
    Communication(String name) throws RemoteException {
        /* If binding fails because there is no running registry
           then create registry and re-try. */
        try {
            String host = InetAddress.getLocalHost().getHostAddress();
            Naming.rebind("//" + host + "/" + name, this);
        } catch (Exception e) {
            try {
                java.rmi.registry.LocateRegistry.createRegistry(1099);
                String host =
                InetAddress.getLocalHost().getHostAddress();
                Naming.rebind("//" + host + "/" + name, this);
            } catch (Exception fail) {
                System.out.println(
                "Shell initialization failed due to registry error.\n"
                + fail);
                System.exit(1);
            }
        }
    }

    /**
     * Check whether the specified service is offered by any of the
     * agents in the Shell.
     *
     * @param service the service name
     * @return typically an ArrayList of String elements reflecting on
     * identifiers of agents that offer the service
     */
    public List serviceLookup(String service) throws RemoteException {
        return Registry.serviceLookup(service);
    }
```

```
/**
 * Locate an agent.
 *
 * @param agentId the agent identifier
 * @return A reference to an agent remote communication interface
 */
public Remote agentLookup(String agentId) throws RemoteException {
    return Registry.agentLookup(agentId);
}

/**
 * Receive and resume a migrating agent.
 *
 * @param code the serialized agent in bytecode form
 */
public void receive(byte[] code) throws RemoteException {
    Agent agentRef = null;
    try {
         agentRef = (Agent) toObject(code);
         Registry.bind(agentRef);
    } catch (Exception e) {
        System.out.println(e);
    }
    agentRef.operate();

}

/**
 * Offers agent migration.
 *
 * @param agentRef the agent to migrate
 * @param destAddress the address of the destination shell
 */
static void migrate(Agent agentRef, String destAddress) {
    try {
        RemoteShell remoteShell =
        (RemoteShell)Naming.lookup(destAddress);
        remoteShell.receive(toByteArray(agentRef));
        Registry.remove(agentRef.getIdentifier());
    } catch (Exception e) {
        System.out.println("Agent migration to " +
        destAddress + " failed.\n" + e);
    }
}

/**
 * Serializes an object into bytecode form.
 *
 * @param object the object to serialize
 * @return the bytecode corresponding to the object
 */
private static byte[] toByteArray(Object object) throws Exception {
    ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
    ObjectOutputStream objOut = new ObjectOutputStream(byteOut);

    objOut.writeObject(object);
    objOut.flush();
    byte[] tmpArray = byteOut.toByteArray();
    objOut.close();

    return tmpArray;
}

/**
 * De-serializes bytecode into an object.
 *
 * @param byteArray the bytecode corresponding to the object
 * @return the de-serialized object
 */
private static Object toObject(byte[] byteArray) throws Exception {
    ByteArrayInputStream byteIn =
    new ByteArrayInputStream(byteArray);
    ObjectInputStream objIn = new ObjectInputStream(byteIn);

    Object tmpObj = objIn.readObject();
    objIn.close();
```

```
            return tmpObj;
        }

}
```

Entry.java

```
/* Copyright (C) 2000-2002 Christos Bohoris, University of Surrey */
package uk.ac.surrey.codeshell;

import java.util.*;

/**
 * A registry entry.
 *
 * @author Christos Bohoris
 */
class Entry {

    /** The agent identifier */
    String identifier;

    /** The agent class name */
    String className;

    /** The services offered by the agent */
    List services;

    /**
     * Initialize an entry.
     *
     * @param id the agent identifier
     * @param cn the agent class name
     * @param s the services offered by the agent
     */
    Entry(String id, String cn, List s) {
        identifier = id;
        className = cn;
        services = new ArrayList();
        services = s;
    }

}
```

Message.java

```
/* Copyright (C) 2000-2002 Christos Bohoris, University of Surrey */
package uk.ac.surrey.codeshell;

import java.io.*;

/**
 * A message communicated between agents.
 *
 * @author Christos Bohoris
 */
public class Message implements Serializable {

    /** The message name. */
    public String name;

    /** The message content. */
    public Object content;

    /**
     * Initialize an empty message.
     */
    public Message() {
        name = null;
        content = null;
    }

    /**
     * Initialize a message.
     *
```

```
     * @param name the message name
     * @param content the message content
     */
    public Message(String name, Object content) {
        this.name = name;
        this.content = content;
    }

}
```

Registry.java

```
/* Copyright (C) Christos Bohoris, University of Surrey */
package uk.ac.surrey.codeshell;

import java.util.*;
import java.rmi.*;
import java.net.*;

/**
 * A registry responsible for naming and lookup services.
 *
 * @author Christos Bohoris
 */
class Registry {
    /** A list of registered agents */
    static List registered;

    /**
     * Initialize a Registry.
     */
    Registry() {
        registered = new ArrayList();
    }

    /**
     * Register agent.
     *
     * @param agent the agent reference
     */
    static void bind(Agent agent) {
        try {
            String host = InetAddress.getLocalHost().getHostAddress();
            String identifier = "//" + host + "/" +
            agent.getIdentifier();
            Naming.rebind(identifier, agent);
            registered.add(new Entry(agent.getIdentifier(),
            agent.getClass().toString(), agent.getServices()));
        } catch (Exception e) {
            System.out.println("Agent binding failed.\n" + e);
        }
    }

    /**
     * Remove agent.
     *
     * @param id the agent identifier
     */
    static void remove(String id) {
        Termination c = null;
        try {
            String host = InetAddress.getLocalHost().getHostAddress();
            String identifier = "//" + host + "/" + id;
            c = (Termination)Naming.lookup(identifier);
            c.kill();
            Naming.unbind(id);
        } catch (Exception e) {
            System.out.println("Agent unbinding failed.\n" + e);
        }
        Entry item = null;
        for (int i = 0; i < registered.size(); i++) {
            item = (Entry)registered.get(i);
            if (item.identifier.equals(id)) {
                registered.remove(i);
                break;
            }
```

```
    }
}

/**
 * Locate an agent.
 *
 * @param agentId the agent identifier
 * @return A reference to an agent remote communication interface
 */
static Remote agentLookup(String agentId) {
    Remote c = null;
    try {
        String host = InetAddress.getLocalHost().getHostAddress();
        String identifier = "//" + host + "/" + agentId;
        c = (Remote)Naming.lookup(identifier);
    } catch (Exception e) {
        System.out.println(e);
    }

    return c;
}

/**
 * Locate a Shell.
 *
 * @param address the shell address
 * @return a reference to the shell
 */
static RemoteShell shellLookup(String address) {
    RemoteShell rs = null;
    try {
        rs = (RemoteShell)Naming.lookup(address);
    } catch (Exception e) {
        System.out.println("A Shell with address " + address +
        " could not be found.");
    }

    return rs;
}

/**
 * Check whether the specified service is offered by any of the
 * agents in the Shell.
 *
 * @param service the service name
 * @return typically an ArrayList of String elements reflecting on
 * identities of agents that offer the service
 */
static List serviceLookup(String serviceId) {
    List results = new ArrayList();
    List s = new ArrayList();
    for (int i = 0; i < registered.size(); i++) {
        s = ((Entry)registered.get(i)).services;
        for (int j = 0; j < s.size(); j++) {
            if (((String)s.get(i)).compareTo(serviceId) == 0) {
                results.add(((Entry)registered.get(i)).identifier);
                break;
            }
        }
    }
    return results;
}

/**
 * Provide information on the registered agents.
 *
 * @return the registered agent information
 */
static String print() {
    String r = "";
    Entry e = null;
    try {
        for (int i = 0; i < registered.size(); i++) {
            e = (Entry)registered.get(i);
            r = r + "{" + e.identifier + ", " +
            e.className + ": " + e.services.toString() + "}\n";
        }
```

```
        } catch (Exception ex) {
            System.out.println("List error.\n" + ex);
        }
        return r;
    }

}
```

RemoteShell.java

```
/* Copyright (C) Christos Bohoris, University of Surrey */
package uk.ac.surrey.codeshell;

import java.rmi.*;
import java.util.*;

/**
 * Allows remote access to the CodeShell communication service.
 *
 * @author Christos Bohoris
 */
public interface RemoteShell extends Remote {

    /**
     * Receive and resume a migrating agent.
     *
     * @param code the serialized agent in bytecode form
     */
    public void receive(byte[] code) throws RemoteException;

    /**
     * Check whether the specified service is offered by any of the agents in the Shell.
     *
     * @param service the service name
     * @return typically an ArrayList of String elements reflecting on identifiers of
     * agents that offer the service
     */
    public List serviceLookup(String service) throws RemoteException;

    /**
     * Locate an agent.
     *
     * @param agentId the agent identifier
     * @return A reference to an agent remote communication interface
     */
    public Remote agentLookup(String agentId) throws RemoteException;

}
```

Shell.java

```
/* Copyright (C) 2000-2002 Christos Bohoris, University of Surrey */
package uk.ac.surrey.codeshell;

import java.net.*;
import java.lang.reflect.*;
import java.rmi.*;
import java.util.*;

/**
 * A Shell is an execution environment for mobile agents following a
 * constrained mobility strategy.
 *
 * @author Christos Bohoris
 */
public class Shell {

    /* A unique, identying name. */
    static String name = null;

    /**
     * Display usage information.
     */
    private void usage() {
        System.out.println(
```

```
        "USAGE: java -jar codeshell.jar {name} {help}");
    }

    /**
     * Construct a Shell object.
     *
     * @param args command line arguments
     */
    Shell(String[] args) {
        new Registry();
        name = "Shell";
        String host = "localhost";
        try {
            host = InetAddress.getLocalHost().getHostAddress();
        } catch (Exception e) {
            System.out.println("Failed to get local host address.\n" +
            e);
        }
        if (args.length == 1) {
            if (args[0].equals("help")) {
                usage();
                System.exit(0);
            }
            name = args[0];
        } else {
          name = "Shell_" + host;
        }
        try {
            new Communication(name);
        } catch (Exception e) {
            System.out.println(
            "CodeShell communication service failure.\n" + e);
        }
        System.out.println("Shell: //" + host + "/" + name);
        new TUI();

    }

    /**
     * Provide the shell name.
     *
     * @return the shell name
     */
    public static String getName() {
        return name;
    }

    /**
     * Allows the termination of an agent and its removal from the
     * registry.
     *
     * @param agentId the agent to terminate
     */
    public static void terminateAgent(String agentId) {
        Registry.remove(agentId);
    }

    /**
     * Run program.
     *
     * @param args command line arguments
     */
    public static void main(String[] args) {
        new Shell(args);
    }

    /**
     * Allows the creation of an agent.
     *
     * @param className the class name of the agent to create
     * @param args list of initialization arguments
     * @return the agent identifier
     */
    public static String createAgent(String className, List args) {
        Agent obj  = null;
        try {
            Class agentClass =  Class.forName(className);
```

```
            obj = (Agent) agentClass.newInstance();

            // Invoke initiate.
            Class[] initParamTypes = new Class[1];
            initParamTypes[0] = List.class;
            Method initMethod = agentClass.getMethod("configure",
            initParamTypes);
            Object[] initParams = new ArrayList[1];
            initParams[0] = new ArrayList();
            initParams[0] = args;
            Object o = initMethod.invoke(obj, initParams);

            // Invoke operate.
            Method operMethod = agentClass.getMethod("operate", null);
            o = operMethod.invoke(obj, null);

            Registry.bind(obj);
        } catch (Exception e) {
            System.out.println("Failed to create agent.\n" + e);
            return null;
        }

        return obj.getIdentifier();
    }

    /**
     * Allows agents to migrate.
     *
     * @param agentRef the agent to migrate
     * @param destAddress the address of the destination shell
     */
    public static void migrate(Agent agentRef, String destAddress) {
        Communication.migrate(agentRef, destAddress);
    }

    /**
     * Locate an agent.
     *
     * @param agentId the agent identifier
     * @return A reference to an agent remote communication interface
     */
    public static Remote agentLookup(String agentId) {
        return Registry.agentLookup(agentId);
    }

    /**
     * Check whether the specified service is offered by any of the
     * agents in the Shell.
     *
     * @param service the service name
     * @return typically an ArrayList of String elements reflecting on
     * identifiers of agents that offer the service
     */
    public static List serviceLookup(String serviceId) {
        return Registry.serviceLookup(serviceId);
    }

    /**
     * Locate a Shell.
     *
     * @param shellId the shell identifier
     * @return a reference to the shell
     */
    public static RemoteShell shellLookup(String shellId) {
        return Registry.shellLookup(shellId);
    }

}
```

TUI.java

```
/* Copyright (C) Christos Bohoris, University of Surrey */
package uk.ac.surrey.codeshell;

import java.io.*;
```

```
/**
 * A Shell TUI.
 *
 * @author Christos Bohoris
 */
class TUI implements Runnable {

    /** The TUI thread. */
    private Thread thread;
    /** Indicates that TUI should terminate. */
    private boolean terminate = false;

    /**
     * Construct TUI. Invoked by the Shell.
     */
    TUI() {
        thread = new Thread(this, "tui");
        thread.start();
    }

    /**
     * The help action.
     */
    private void onHelp() {
        System.out.println("[c]reate agent {class name}");
        System.out.println("[t]erminate an agent {agent name}");
        System.out.println("[l]ist agents");
        System.out.println("[e]xit");
    }

    /**
     * The create action.
     *
     * @param the class name of the agent to create
     */
    private void onCreate(String className) {
        Shell.createAgent(className, null);
    }

          /**
           * The agent termination action.
           *
           * @param agentId the name of the agent to terminate
           */
          private void onTerminate(String agentId) {
                    Shell.terminateAgent(agentId);
          }

    /**
     * The list action.
     */
    private void onList() {
        System.out.println(Registry.print());
    }

    /**
     * The exit action.
     */
    private void onExit() {
        terminate = true;
        System.exit(0);
    }

    /**
     * Run on the TUI thread.
     */
    public void run() {
        String line = null;
        do {
            System.out.print("# ");
            BufferedReader in = new BufferedReader(new
            InputStreamReader(System.in));
            try {
                line = in.readLine();
            } catch (Exception e) {
                e.printStackTrace();
            }
```

```
            if (line.trim().startsWith("c") == true) {
                int i = line.trim().indexOf(" ");
                onCreate(line.trim().substring(i + 1));
            } else if (line.trim().startsWith("l") == true) {
                onList();
            } else if (line.trim().startsWith("t") == true) {
                int i = line.trim().indexOf(" ");
                onTerminate(line.trim().substring(i + 1));
            } else if (line.trim().startsWith("e") == true) {
                onExit();
            } else if (line.trim().startsWith("h") == true) {
                onHelp();
            } else {
                System.out.println("\t...");
            }
        } while (terminate == false);
    }

}
```

Termination.java

```
/* Copyright (C) 2000-2002 Christos Bohoris, University of Surrey */
package uk.ac.surrey.codeshell;

import java.rmi.*;

/**
 * Allows the remote termination of agent instances.
 *
 * @author Christos Bohoris
 */
public interface Termination extends Remote {

    /**
     * Terminate an agent instance.
     */
    public void kill() throws RemoteException;

}
```