Summary

Today's telecommunication market is populated by a great variety of available networking solutions. Given the exponential growth of network infrastructures and customers' needs, increasing system diversity is inevitable. A key emerging trend is the convergence of heterogeneous personal devices into spontaneous infrastructure-less networks. In such a diverse environment, flexible middleware technologies can serve as the "glue" between heterogeneous systems. Products from different vendors and network devices with different architectures could be brought together using a "common language" offered by middleware.

The objective of this thesis is to investigate middleware technologies as a vehicle of managing emerging, fixed and mobile networks. Four different middleware technologies are compared and evaluated in terms of performance and usability. The investigated technologies include the well-established CORBA platform, Mobile Agents and the XML-based Web Services/SOAP and XML-RPC technologies. XML-based technologies present a number of attractive features, such as the use of the HTTP transport protocol and easy integration with XML-structured data and Web browsers. On the other hand, Mobile Agents offer the inherent feature of software migration that can be used for network element programmability and capability enhancement, albeit at a high performance cost.

Two different middleware case studies are examined in order to assess their suitability for emerging network architectures. The first case study investigates an Agent-based middleware system for managing Quality of Service in IP Differentiated Services networks. The middleware platform is addressed to a fixed network infrastructure, demonstrating the integration of heterogeneous network elements. The second case study investigates a middleware-based programmable infrastructure that allows the nodes of a mobile ad hoc network to download and activate required protocol and service software dynamically. This enables the alignment of nodes capabilities and allows, for instance, full-scale quality of service-based communication among heterogeneous ad hoc network nodes.

Key words: Middleware, XML, Mobile Agents, Network Management, Quality of Service, Programmability, Ad Hoc networks
Dedicated to my parents Christos and Katerina
Acknowledgments

I would like to thank my supervisor Professor George Pavlou for his support during the last 4 years. Without his guidance and encouragement this thesis would not be what it is today.

I would also like to thank my colleagues, as well as friends, in the Networks Group of CCSR: Paris Flegkas, Christos Bohoris, Panos Trimintzios, Alvin Yew, Sunil Iyengar, Haitham Cruickshank, Ning Wang, Frederico Cavalcanti, Antonio Liotta, Zhili Sun and Ilias Andrikopoulos who were always there to help me whenever I needed to.

Special thanks to my colleagues in the PAN project: Siva Sivavakeesar and Apostolis Malatras for their valuable input and always friendly cooperation.

Also, special thanks to my partners in the MANTRIP project: Telma Mota, Angelos Michalas, Kun Yang, Marco Ventura, Walter Eaves and Alex Galls for a smooth and productive cooperation.

A very special thanks goes out to my good friends: George Lazaridis, Dimitris Mavrakis, Stelios Choulis, Elefni Karakosta, Michael Kalkavouras, Khaldoon Al-Naimi and Micro for making these 4 years look so bright and cheerful.
Contents

Summary ............................................................................................................................................. ii
Acknowledgments .......................................................................................................................... iv
Contents ........................................................................................................................................... v
List of Figures ...................................................................................................................................... viii
Abbreviations ..................................................................................................................................... ix
Thesis Related Publications ......................................................................................................... xii
Thesis Related Projects ................................................................................................................. xiii

PART I - THESIS BACKGROUND ............................................................................................. 1

1 Introduction ..................................................................................................................................... 1
  1.1 Research Motivation .............................................................................................................. 2
  1.2 Thesis Roadmap .................................................................................................................... 4

2 Background ..................................................................................................................................... 5
  2.1 Network Management .......................................................................................................... 5
    2.1.1 The Centralised Network Management Architecture ...................................................... 6
    2.1.2 Decentralised Network Management Architecture ............................................................ 7
      2.1.2.1 Mobile Agents for Network Management .................................................................... 7
      2.1.2.2 Management by delegation .......................................................................................... 7
      2.1.2.3 CORBA based management ....................................................................................... 8
    2.1.3 SNMP ............................................................................................................................ 8
    2.1.4 CMIP ........................................................................................................................... 9
    2.1.5 TMN ........................................................................................................................... 9
  2.2 Quality of Service in IP Networks ........................................................................................ 11
    2.2.1 Integrated Services - RSVP ............................................................................................ 11
    2.2.2 Differentiated Services .................................................................................................... 12
  2.3 Mobile Ad Hoc Networks ...................................................................................................... 14

PART II - THESIS CONTRIBUTIONS ......................................................................................... 17

3 Middleware Technologies .......................................................................................................... 17
## Contents

5.4 Middleware Communication and Components ....................................................... 55  
5.4.1 System Operation ............................................................................................. 57  
5.4.2 Loadable plugins ............................................................................................. 58  
5.4.3 Plugin Election and Distribution ...................................................................... 59  
5.5 Platform Evaluation ............................................................................................ 60  
5.5.1 TestBed Experimentation ................................................................................ 60  
5.5.2 Simulation ........................................................................................................ 63  
5.6 Conclusions ......................................................................................................... 65  

6 Summary and Conclusions ..................................................................................... 67  
6.1 Thesis Summary .................................................................................................. 57  
6.2 Thesis Contributions ......................................................................................... 67  
6.2.1 Comparison of Middleware Technologies ...................................................... 67  
6.2.2 Assessment of Mobile Agents within a QOS Configuration and Performance Management System ................................................................. 68  
6.2.3 Proposal of XML-Based Programmable Middleware for Ad Hoc networks ........ 68  
6.3 Conclusions ....................................................................................................... 69  
6.4 Future Directions .............................................................................................. 71  
6.4.1 Compression in XML-based technologies ...................................................... 71  
6.4.2 Exploitation of Context in Middleware Applications ...................................... 71  
6.4.3 Distributed Management Approach in Ad Hoc networks ................................ 72  

Bibliography .............................................................................................................. 73  

Appendix A – UML Diagrams .................................................................................... 80
List of Figures

Figure 2-1: TMN separation of functional concerns ................................................................. 11
Figure 2-2: Typical arrangement of a Packet Classifier and a Traffic Conditioner .................. 13
Figure 2-3: An example MANET topology .............................................................................. 15
Figure 2-4: A MANET with a single hop connection to the Internet ....................................... 16
Figure 2-5: MANET-enabling technologies summary ............................................................... 16
Figure 3-1: Mobile Agent migration example ............................................................................ 18
Figure 3-2: CORBA ORB Architecture .................................................................................... 21
Figure 3-3: SOAP Envelope .................................................................................................... 23
Figure 3-4: Middleware technologies features comparison ....................................................... 25
Figure 3-5: Response time for the four middleware technologies ........................................... 27
Figure 3-6: Overhead traffic produced by the four middleware technologies ......................... 28
Figure 3-7: Memory usage at server side .................................................................................. 29
Figure 3-8: Lines of written code ............................................................................................ 30
Figure 4-1: Architecture for Network Configuration and Monitoring ...................................... 36
Figure 4-2: Mobile Agent-based performance monitoring system .......................................... 38
Figure 4-3: Differentiated Service architecture ....................................................................... 39
Figure 4-4: Performance Monitoring System Functionality .................................................... 43
Figure 4-5: QoS Configuration functionality .......................................................................... 44
Figure 4-6: DiffServ TestBed .................................................................................................. 46
Figure 4-7: QoS Management Application – Configuration GUI ............................................ 48
Figure 4-8: QoS Management Application – Monitoring GUI ................................................ 49
Figure 4-9: QoS Management Application – Reconfiguration GUI ......................................... 50
Figure 5-1: Middleware platform architecture ....................................................................... 57
Figure 5-2: Plugin election cost function .................................................................................. 59
Figure 5-3: TestBed configuration – Test case 1 ...................................................................... 61
Figure 5-4: Average control cost incurred per node as a function of increasing node count ...... 63
Figure 5-5: Average latency as a function of increasing node count ....................................... 64
Figure 5-6: Average control cost incurred per node as a function of increasing node density ... 65
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AODV</td>
<td>Ad hoc On-Demand Distance Vector</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ATM</td>
<td>Asynchronous Transfer Mode</td>
</tr>
<tr>
<td>BML</td>
<td>Business Management Layer</td>
</tr>
<tr>
<td>CDC</td>
<td>Connected Device Configuration</td>
</tr>
<tr>
<td>CH</td>
<td>Cluster head</td>
</tr>
<tr>
<td>CMIP</td>
<td>Common Management Information Protocol</td>
</tr>
<tr>
<td>CMIS</td>
<td>Common Management Information Service</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker</td>
</tr>
<tr>
<td>DBM</td>
<td>Delegation Backplane Middleware</td>
</tr>
<tr>
<td>DiffServ</td>
<td>Differentiated Services</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name Server</td>
</tr>
<tr>
<td>DSCP</td>
<td>Differentiated Services Code Point</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition</td>
</tr>
<tr>
<td>EML</td>
<td>Element Management Layer</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunication Standard Institute</td>
</tr>
<tr>
<td>FIPA</td>
<td>Foundation of Intelligent Physical Agents</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocol</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Definition Language</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IIOP</td>
<td>Internet Inter-ORB Protocol</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IPPM</td>
<td>IP Performance Metrics</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organisation for Standardization</td>
</tr>
<tr>
<td>ITU-T</td>
<td>International Telecommunication Union (-Telecommunications sector)</td>
</tr>
<tr>
<td>J2ME</td>
<td>Java 2 Micro Edition</td>
</tr>
<tr>
<td>JAIN</td>
<td>Java API Initiative</td>
</tr>
<tr>
<td>Java-RMI</td>
<td>Java Remote Method Invocation</td>
</tr>
<tr>
<td>JRMP</td>
<td>Java Remote Method Protocol</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>MA</td>
<td>Mobile Agent</td>
</tr>
<tr>
<td>MANET</td>
<td>Mobile Ad Hoc Networks</td>
</tr>
<tr>
<td>MAT</td>
<td>Mobile Agent Technology</td>
</tr>
<tr>
<td>MbD</td>
<td>Management by Delegation</td>
</tr>
<tr>
<td>MIMB</td>
<td>Multipurpose Internet Mail Extensions</td>
</tr>
<tr>
<td>MN</td>
<td>Mobile Node</td>
</tr>
<tr>
<td>MO</td>
<td>Managed Object</td>
</tr>
<tr>
<td>NE</td>
<td>Network Element</td>
</tr>
<tr>
<td>NEF</td>
<td>Network Element Function</td>
</tr>
<tr>
<td>NEL</td>
<td>Network Element Layer</td>
</tr>
<tr>
<td>NH</td>
<td>network head</td>
</tr>
<tr>
<td>NML</td>
<td>Network Management Layer</td>
</tr>
<tr>
<td>NMS</td>
<td>Network Management Systems</td>
</tr>
<tr>
<td>ODP</td>
<td>Open Distributed Processing</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management group</td>
</tr>
<tr>
<td>OSA</td>
<td>Open service Architecture</td>
</tr>
<tr>
<td>OSI</td>
<td>Open System Interconnection</td>
</tr>
<tr>
<td>OSPF</td>
<td>Open Shortest Path First</td>
</tr>
<tr>
<td>PDA</td>
<td>Personal Digital Assistant</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
</tr>
<tr>
<td>PHB</td>
<td>Per-Hop Behaviour</td>
</tr>
<tr>
<td>POA</td>
<td>Portable Object Adapter</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RDS</td>
<td>Remote Delegation Service</td>
</tr>
<tr>
<td>RMON</td>
<td>Remote Monitoring</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>RSVP</td>
<td>Resource Reservation Protocol</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SLS</td>
<td>Service Level Specification</td>
</tr>
<tr>
<td>SML</td>
<td>Service Management Layer</td>
</tr>
<tr>
<td>SMTP</td>
<td>Simple Mail Transfer Protocol</td>
</tr>
<tr>
<td>SNMP</td>
<td>Simple Network Management Protocol</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TCS</td>
<td>Traffic Conditioning Specification</td>
</tr>
<tr>
<td>TDS</td>
<td>Tabular Data Stream</td>
</tr>
<tr>
<td>TFTP</td>
<td>Trivial File Transfer Protocol</td>
</tr>
<tr>
<td>TMN</td>
<td>Telecommunications Management Network</td>
</tr>
<tr>
<td>TOS</td>
<td>Type of Service</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery and Integration</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
<tr>
<td>VPrP</td>
<td>Virtual Provisioned Pipe</td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless Local Area Networks</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Definition Language</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Wed</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
Thesis Related Publications

Journal Papers


Conference Papers


Thesis Related Projects


PART I - THESIS BACKGROUND

Chapter 1

1 Introduction

Today's research world and telecommunication market is troubled about the convergence of heterogeneous infrastructures, network elements and applications from different vendors. The need for coexistence of diverse environments is evident every day, as personal devices tend to form infrastructure-less ad-hoc networks and network operators hope for better interoperability. A way of harmonising heterogeneous environments and diverse applications is by the use of Middleware. Middleware can be defined as a mediation layer between two otherwise separate applications or separate products that serve as the glue between the two systems. Middleware is sometimes called plumbing because it connects two sides of an application and passes data between them. Middleware can be used to provide high-level abstractions and services to applications, to ease application programming, application integration, and system management tasks. In this sense, middleware moves beyond transaction management to encompass database management systems, web servers, application servers, content management systems, and similar tools that support the application development and delivery process.

Middleware provides the mechanism by which network applications communicate. This includes in the case of database networking for example the service of putting packages of query results data into network transport packets. Microsoft SQL Server, for example, uses Sybase's Tabular Data Stream (TDS) protocol to handle formatting of data for transport across the network. Client and server do not have to have intimate knowledge of each other in order for work to get done. Differences between platform specific encodings like big-endian and little-endian or EBCDIC and ASCII are typically hidden by middleware. Middleware often runs on a variety of platforms, letting the organisation utilize all its existing desktop and server hardware as applications require. Still, some middleware products find it hard to look beyond Windows clients and UNIX or Windows NT servers. Middleware often makes networking choices transparent to application programmers. Moreover, many solutions don't just offer a simple name service for their server or service names. Advanced middleware solutions offer centralised naming services with some level
Chapter 1. Introduction

of distribution. The issues are the same as those associated with the Domain Name System (DNS) on the Internet or NetWare Directory Services (NDS) on NetWare.

The marketplace for middleware technology is continuously changing. There is an increasing demand to apply middleware technologies to a wider variety of application domains including real-time systems, embedded systems, fault-tolerant systems, multimedia and mobility. In this context, providing Quality of Service (QoS) is one of the major challenges. Applications should be able to specify and modify QoS requirements. The middleware should be able to provide the requested QoS and adapt to any required changes in QoS (i.e. throughput, loss rate, etc). New media types and new applications might introduce new QoS characteristics. In order to support these new requirements, middleware QoS management must be extensible.

Middleware technologies can find applications in ubiquitous networks, such as ad hoc. The concept of mobile ad hoc networks (MANETs) has recently received significant attention due to the increasing popularity of tetherless computing and the rapid growth of wireless networking. In ad hoc networks, the mobile nodes (MNs) are free to move randomly and organise themselves arbitrarily; thus, the network’s wireless topology may change rapidly and unpredictably. Typically this kind of network operates in a standalone fashion or may be connected to an infrastructure-based network through a gateway, e.g. wireless LAN access point, base station, etc. Conventional wireless networks require as a prerequisite some form of fixed network infrastructure and centralised administration for their operation. In contrast, since MANETs are self-creating, self-organising and self-administrating, individual MNs of the network are responsible for dynamically discovering other nodes they can communicate with. This way of dynamically creating a network often requires the ability to rapidly create, deploy and manage services and protocols in response to user demands.

1.1 Research Motivation

Network management tasks are traditionally performed through appropriate management protocols, such as the IETF Simple Network Management Protocol (SNMP), and the ISO Common Management Information Protocol (CMIP). Both SNMP and CMIP are based on the manager-agent model. According to this management model, a central management node (Manager) interacts with management agents (Agents) that are embedded in the network devices. This interaction is based on a well-defined management protocol, which specifies a packet format and an information structure for performing a particular set of operations. Although a host of management systems follow this manager-agent model, several drawbacks are attributed to this centralised management paradigm. More specifically, it is believed that this client/server approach leads to poor performance since the exchange of management information can be susceptible to
congestion around the Manager node. The latter is inevitable whenever the manager attempts to carry out many simultaneous management operations to managed devices. In addition, the scalability of this approach is also questioned, especially when it comes to managing networks of considerable size and complexity. This is particularly pertinent to emerging IP networks, e.g. ad hoc, which are often decentralised global meshes of numerous network devices.

In view of this situation, existing management strategies are refined and new paradigms are proposed in order to boost decentralised approaches in network management. Related attempts are RMON (Remote Monitoring) and the proxy agent paradigm specified within SNMPv2 [rfc1908]. Effort has been also allocated to techniques that make use of mobile code, such as MBD (Management by Delegation) [Yemi91] [Gold96], which move the management function closer to the network devices. Recently, the Mobile Agent technology has emerged as a promising solution towards implementing strategies that distribute and automate management tasks. There are still though several open research issues as mobile agent technology needs to mature in order to support proper decentralisation of network management. Recently, a first wave of research activities has produced considerable results related to this new technology. Furthermore, standardisation efforts have taken place and solutions to thorny issues such as security, portability, resource management, and control of mobile agents have been devised.

The convergence between Telecommunications and Computing, as exemplified by palmtop computers integrated with cellular phones or by palmtop/laptop computers with wireless network cards, has the capability to bring together mobile users and computing services and applications through ubiquitous intelligent communications. Key contributors are developments in mobile/wireless communications, such as cellular systems, Wireless Local Area Networks (WLANs), Bluetooth, the proliferation of various types of mobile devices, e.g. latest generation mobile phones, palmtop/laptop computers, etc., and, most important, the use of the Internet Protocol (IP) as the unifying basis of packet-based communications. Paired with context awareness, thin clients, intelligent adaptation, cyber foraging and anticipation of user intent among other, ubiquitous communication systems may evolve into pervasive computing systems that will eventually recede into the background of daily life.

In this context, there is the opportunity to reconcile the perspectives of the telecommunications and computing communities through dynamically programmable network architectures that can support cooperation, adaptability and alignment with respect to the required basic capabilities and additional services of the devices that form an ad hoc network. For example, while routing and quality of service conform to standardised frameworks and protocols in fixed IP networks, there are many potential solutions for ad hoc networks that depend also on the characteristics of the particular ad hoc network, e.g. topology volatility, characteristics of radio links, capabilities of terminodes (ad hoc nodes), etc. Given the multitude of potential solutions that may be
environment-dependent, programmability is of paramount importance to allow terminodes to be enhanced on the fly with the required communication capabilities in the ad hoc environment. In addition, services/applications may migrate to more powerful devices that have the required capabilities while less powerful devices may outsource computing tasks, so called cyber foraging. Programmability is possible through recent advances in distributed systems technologies and transportable “execute-anywhere” software.

1.2 Thesis Roadmap

The remaining of this thesis is organised as follows.

Chapter 2 describes the background topics associated with this thesis. These include theoretical aspects of network management, quality of service in IP network and mobile ad hoc networks. Middleware technologies are not covered in this chapter as they are thoroughly discussed in Chapter 3.

Chapter 3 presents a review of four well known middleware technologies, namely; Mobile Agents, CORBA, SOAP and XML-RPC. For each technology, one commercial platform is selected, where a prototype application is build on. In this way, we assess the performance and usability of each one and draw some conclusions on their suitability for certain applications.

Chapter 4 presents an Agent-based middleware system for managing the Quality of service in IP DiffServ capable Networks. The system is responsible for configuring and monitoring a desired level of QoS, which can be offered to a network customer through a Service Level Agreement (SLA). Mobile Agent Technology (MAT) is used to implement the QoS management system and IP Differentiated Services (DiffServ) technology is used as a test case.

Chapter 5 presents a XML-RPC based programmable infrastructure that allows the nodes of a mobile ad hoc network to download and activate required protocol and service software dynamically. This will enable the alignment of the nodes’ capabilities and allow, for instance, full-scale quality of service-based communication among heterogeneous ad hoc network nodes.

Finally, Chapter 6 concludes the thesis, summarising the main results and contributions. It discusses the work from a global viewpoint, indicates the extent to 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 thesis. These include theoretical aspects of network management, quality of service in IP networks and mobile ad hoc networks mobile agents. Middleware technologies are not covered in this chapter as they are thoroughly discussed in the next chapter 3.

2.1 Network Management

Network Management Systems (NMS) are strongly rooted in the manager/agent model of distributed systems. In this model a network manager residing on a central station contains most of the management logic and processes the data collected from physically distributed agents.

The two main communication protocols used are Simple Network Management Protocol (SNMP) and Common Management Information Protocol (CMIP) but they don’t provide the required scalability that is needed in today’s predominantly complex networks due to the large number of network components, vast topologies, and unpredictable network dynamics.

The basic idea to address these problems is to bring management intelligence as close as possible to the managed resources.

One of the prominent techniques providing a solution is Management by Delegation. Instead of traditional methods of exchanging client/server messages, the manager station specifies a task to be carried out by locating an agent on involved devices, where the actual execution of the task takes place. However, in the concept of Management by Delegation, as described in subsection 2.1.2.2, the static nature of management agents still leaves considerable control responsibility in the domain of the manager.

On the other hand the use of mobile agents affords new opportunities for the distribution of processing and control in network management. Rather than transporting the data to a central location, mobile agents operate in the same network locale as the data and return only relevant or compiled data, thereby reducing the management traffic load on the network.
2.1.1 The Centralised Network Management Architecture

Several network management standards exist nowadays. The most prominent ones are the ISO/OSI CMIP and IETF's SNMP. Both OSI based management systems and Internet based management systems follow a centralised client-server model. In this client-server model, the client is the managing system and the server is the managed system. In each network element there is a software application called an agent, which is responsible for performing operations on the managed objects, which represent logical or physical resources on the device. The client is a NMS, which interrogates, usually periodically, all the network elements and based on the information gathered builds a picture about the network status and performance. If problems occur, the server will issue appropriate operations to the network elements to overcome the problems. Furthermore, as the number of network elements increases, the network management traffic and the processing power required by the network management system increase too and impose a significant overhead in the overall cost of the management, making it less competitive. Nevertheless, ISO/OSI systems perform better because CMIP supports **scoping** and **filtering**. Scoping refers to the limitation of an operation to a set of objects. Filtering refers to the limitation of an operation to a set of objects determined by a result of a Boolean expression.

The client-server approach has a number of drawbacks:

- **Inclusion of new services in the server is not an easy task:** the network management station can only invoke a fixed set of predefined services that the server supports. These services can only be enhanced if a new version of the server is compiled and installed. For the realisation of more sophisticated services the interface between the client and the server has to change.

- **Uses resources inefficiently:** most of the processing is performed on the network management station. Although the ISO/OSI approach seems to be better by having scoping and filtering operations, the network manager system is responsible for the proper operation of the whole network. It has to correlate information that receives from many network devices and respond by issuing appropriate SNMP or CMIP commands. As the processing power available in network devices increases it is more efficient to move the processing logic to the data rather than the data to the logic.

- **Depends on network availability:** Both CMIP and SNMP depend on the availability of the network in order to perform their operations. SNMP uses UDP, which is a connectionless unreliable transport protocol while CMIP relies on a connection-oriented reliable transport protocol, which has first to establish a connection before invoking any operations. However, both protocols rely on the underlying networks to a great extent. This makes management very difficult in cases of intermittent network availability.
- Increases network traffic. The polling approach used in these systems (more so in SNMP rather than CMIP-based systems) has a negative impact on the bandwidth utilisation as the number of network elements grows.
- Non-scalable. It is apparent that these architectures are not suitable for large-scale networks, due to the inefficient use of processing and network resources.

2.1.2 Decentralised Network Management Architecture

Nowadays telecommunication networks are characterised by complexity and heterogeneity. These characteristics are likely to be more apparent in the future as new technologies emerge and have to be integrated with the legacy systems. It is also known that we are moving towards an open market of communication services where the vision is "information any time, at any place, in any form". In this context, the quality, customisation and instant provision of services will be the main points of competition between service providers. Service providers need more flexible and scalable approaches to network management than the existing ones.

2.1.2.1 Mobile Agents for Network Management

The Mobile Agent Technology promises to deliver more distributed, scalable and customisable solutions to today's telecommunication management needs. An agent is an autonomous software program that performs a task on behalf of a user or another process. The agent's intelligence may vary from simple predefined rules to self-learning Artificial Intelligence machines. Agents may co-operate with each other, execute asynchronous or synchronous operations and migrate to remote nodes in order to accomplish a task. In this case, they are referred to as Mobile Agents (MA). The MAs can utilise the services offered on the destination system. That makes this approach more general in terms of requirements and capabilities that most current distributed systems provide.

2.1.2.2 Management by delegation

Management by Delegation (MbD) is a novel network management paradigm introduced by Goldszmidt and Yemini [Golds93]. This approach although similar to the Mobile-Agent approach differs in what is called "Elastic Processing". Elastic Processing is a distributed computing paradigm that supports dynamic extensibility of remote software processes, i.e., it is both spatial and temporal distribution. Elastic processes are executing programs that can dynamically integrate new functionality sent to them from external processes as delegated agents. Elastic processing is language independent and supports explicit remote control of agent's execution. The technologies that support remote elasticity consist of a "Remote Delegation Service" (RDS) and a multithreaded "Delegation Backplane Middleware" (DBM) architecture for elastic processes.
The DBM runtime environment implements a software "backplane" where delegated programs are loaded and can be executed as threads in a shared address space. DBM supports translation and dynamic linking of delegated code, a multithreaded execution environment, dynamic resource allocation, and inter-process communications. Mobile Agents are written in a specific, usually interpreted language, like Java. RDS provides the ability to remotely configure an elastic process, control the execution of delegated agents, and convey information to and from these agents. It supports a generic neutral mechanism for dynamically extending processes under remote control.

Delegated agents have been written in several languages. An elastic process can be dynamically extended with a new interpreter for a scripting programming language. The process will then be able to accept delegated agents written in that language. Delegated agents can be compiled or interpreted, while remote scripting agents are always interpreted. Another difference is that elastic processes permit explicit remote control of the execution of delegated agents to authorised parties. In addition, elastic processes can be configured and customised to support arbitrary security and safety policies while scripting technologies typically enforce a pre-defined "one-size-fits-all" security policy. Finally, RDS can execute over both reliable (TCP) and unreliable (UDP) transport protocols.

2.1.2.3 CORBA based management

OMG's Common Object Request Broker Architecture (CORBA) [OMG92] is being increasingly advocated for telecommunications management [Rut96]. As it happens with the manager-agent protocols, CORBA provides client-server interoperability between remote systems through the use of standardised protocols and multiple object access available with CMIP and SNMP. In addition, CORBA supports standardised APIs for performing remote procedure calls from clients to server objects. This is done through standardising the mapping between various programming languages and the Interface Definition Language (IDL) used to describe CORBA server interfaces. Compilers can generate client stub and server skeleton code in a number of languages, e.g. C, C++, Java, and Ada.

2.1.3 SNMP

Many different technologies are currently being put forward as candidates for communications management. The first management specific technologies were two rival manager-agent protocols, which were initially designed for managing network elements. Manager-agent protocols enact management on communications resources through their representation on an agent as a collection of Managed Objects (MOs).
One of the manager-agent protocols, SNMP [rfc1157], emerged from the Internet community. This allows access to notional managed objects in an agent using a best-effort transport protocol (UDP). SNMP is the dominant technology used in the management of Internet network elements.

2.1.4 CMIP

The other manager-agent protocol is the Common Management Information Protocol (CMIP) [X711] underlying OSI System Management framework X700 series. This has been widely accepted by the telecommunications community, and was adopted for implementing the physical architecture in TMN. When applied to inter-domain management the need was identified for CMIP to be integrated with X.500 directory in order to allow transparent navigation between MOs on separate agents [Stath95][Bjer94]. The TMF (Tele-Management Forum) has developed some service management related interface agreements using CMIP, and several research projects have attempted to implement service management OSFs using CMIP platforms [Hall96][Griff96]. Research experiences in developing CMIP OSs reveal that the difficulties experienced were closely related to the CMIP Application Programming Interface (API) made available to the developer in the platform used. These APIs ranged from low level ones such as XMP/XOM used in Hewlett-Packards OpenView system, to the high level RMIB C++ [Pavlou94] and Tcl/Tk APIs used in the OSIMIS platform [Pavlou95]. The differing nature of these APIs also precluded the reuse of code across platforms, though recently the TMF has produced an open C++ CMIP API [Chatt97].

Interoperation between managers and agents implemented using the different protocols is possible using gateways for converting between SNMP and CMIP [McCa97][Dassow97]. However, when accessing a CMIP agent from an SNMP manager via such a gateway, some of the protocol features of the more functionally rich CMIP, e.g. scoping and filtering, are lost.

2.1.5 TMN

The Telecommunications Management Network (TMN) attempts to achieve integrated network management in the telecommunications industry by calling for all management systems to include an interoperable interface which permits each system to be integrated into a larger management hierarchy. By means of such interfaces, hierarchical integration of management systems can be achieved within the administration of a single service provider. Interoperable interfaces also make possible the interconnection of multiple service providers and the connection of customer to service provider, resulting in the possibility of industry-wide integration of management infrastructures at all levels.
TMN is defined in a series of recommendations from the ITU-T [M3000]. The TMN architecture is specified in recommendation M.3010 [M3010] and defines a Logical Layer Architecture in which conceptual layers address different concerns within a provider’s management network. The following layers are identified (listed from the bottom up):

- A Network Element Layer (NEL) containing the network resources to be managed.
- An Element Management Layer (EML) that is concerned with the management of individual network elements.
- A Network Management Layer (NML) that is concerned with managing the whole network.
- A Service Management Layer (SML) that is concerned with the management of customer services.
- A Business Management Layer (BML) that is concerned with the management of the entire enterprise

Each layer is intended to provide the layer above it with the functions required to perform its functions.

M.3010 specifies a functional architecture that identifies types of functional blocks and the types of reference points that exist between them. The taxonomy of functional blocks makes the distinction between: a Network Element Function (NEF) managing individual network elements; a Mediation Function (MF) that mediates between different TMN interfaces; a Q-Adapter Function to non-TMN compliant network element managers; a Work-Station Function (WSF) presenting information to human operators; and a general Operations System Function (OSF) that monitors, co-ordinates and controls telecommunications functions. The functional architecture identifies reference points that define the functions that may be exchanged between functional blocks. Reference points therefore provide the basis for defining interfaces between physical implementations of the functional blocks. The functional architecture also distinguishes between the types of reference points connecting functional blocks within a single TMN (q reference points) and those connecting OSFs in different TMNs (x reference points). TMN management functions are categorised, for the purpose of standardisation, into the areas of Fault management, Configuration management, Accounting management, Performance management and Security management. This categorisation is commonly referred to by the acronym FCAPS. A functional decomposition of the TMN problem area can therefore be represented as a five by five grid consisting of FCAPS in one axis and the TMN layers in another [DES403]. This can also be further decomposed along a third axis according to whether inter or intra domain issues are being addressed, i.e. whether the resulting reference points are x or q type (Figure 2-1).
TMN interfaces are defined according to the methodology described in recommendation M.3020 [M3020], which results in the definition of management services, management functions and management information models. The TMN management services defined today in [M3200] are mostly network-related, with the exceptions of Customer Administration and Tariff, Charging and Accounting Administration. The more detailed TMN management functions defined in [M3400] also mostly address the network and network element management layers.

Some of management functions have been refined into information models, though mostly for function in the EML and NML, e.g. the generic network management in [M3100]. Some generic systems management functions exist in the form of OSI-SM System Management Functions [X700].

2.2 Quality of Service in IP Networks

2.2.1 Integrated Services – RSVP

The Internet Engineering Task Force (IETF) has proposed many service models and mechanisms to meet the demand for QoS. Among them is the Integrated services (Int-Serv) / Resource Reservation Protocol (RSVP) model. The Integrated Services model is characterised by resource reservation. For real-time applications, before data are transmitted, the applications must first set up paths and reserve resources. RSVP is a signaling protocol for setting up paths and reserving resources.

The Integrated Services model for IP QoS architecture defines three classes of service [NORT98]:

- Guaranteed with bandwidth, bounded delay, and no-loss guarantees.
- Controlled load approximately best-effort service in a lightly loaded network.
• Best-effort similar to what the Internet currently provides under a variety of load conditions, from light to heavy.

RSVP-Resource Reservation Protocol

The RSVP [White99] provides the signalling to enable network resource reservation. Although typically used on a per-flow basis, RSVP is also used to reserve resources for aggregates RSVP is the most complex of all the QoS technologies, for applications (hosts) and for network elements (routers and switches). As a result it also represents the maximum differentiation from standard “best-effort” IP service and provides the highest levels of QoS in terms of service guarantees, granularity of resource allocation and detail of feedback to QoS-enabled applications and users.

2.2.2 Differentiated Services

DIFFerentiated SERVices, as proposed by the IETF Differentiated Services Working Group, allows IP traffic to be classified into a finite number of service classes that receive different forwarding treatment. For example, traffic belonging to a higher priority and/or delay service class receives some form of preferential treatment over traffic classified into a lower service class. Differentiated services do not attempt to give explicit end-to-end guarantees. Instead, in congested network elements, traffic with a higher class of priority has a higher probability of getting through, or in case of delay priority, is scheduled for transmission before traffic that is less delay-sensitive.

The information required to perform actual differentiation in the network elements is carried in the Type of Service (TOS) field of the IPv4 packet headers or the Traffic Class field of the IPv6 packet headers, referred to as the DS Field or Codepoint (DSCP) [Nicho99]. Thus, since the information required by the buffer management and scheduling mechanisms is carried within the packet, differentiated services do not require signalling protocols to control the mechanisms that are used to select different treatment for the individual packets. Ideally, the amount of state information, which is required to be maintained per node, is proportional to the number of service classes and not proportional to the number of application flows.

At each differentiated services user/provider boundary, the service provided is defined by means of a Service Level Agreement (SLA). The SLA is a contract, established either statically or dynamically, that specifies the overall performance and features, which can be expected by a customer. Because differentiated services are for unidirectional traffic only, each direction must be considered separately. The subset of the SLA, which provides the technical specification of the service, is referred to as the Service Level Specification (SLS).
A profound subset of the SLS is the Traffic Conditioning Specification (TCS), which specifies detailed service parameters for each service level. These service parameters include service performance parameters (e.g. throughput, latency, drop probability) and traffic profiles corresponding to the requested service. Furthermore, the TCS may define the marking and shaping functions to be provided.

The Differentiated Services architecture is composed of a number of functional elements, namely packet classifiers, traffic conditioners and Per-Hop forwarding Behaviours (PHBs). According to the basic differentiated services architecture, all these elements are normally placed in ingress (i.e. edge) nodes of a differentiated services domain while interior and egress nodes exhibit only PHB functionality. In the following paragraphs a short description for each of the elements is given and the various components that comprise them are briefly presented [Nicho99]. The Differentiated Service ingress node functions (apart from PHB forwarding) are presented in Figure 2-2.

![Diagram](image)

Figure 2-2: Typical arrangement of a Packet Classifier and a Traffic Conditioner.

**Packet Classifiers**

Packet classification is a significant function, which is normally required at the edge of the differentiated services network. Its goal is to provide identification of the packets belonging to a traffic stream that may receive differentiated services. Classification is done with packet classifiers, which select packets based on the content of packet headers according to well-defined rules determined by the Traffic Conditioning Agreement or SLS. Two types of classifiers are currently defined: the Behaviour Aggregate (BA) classifier, which selects packets based on the DS Codepoint only, and the Multi-Field (MF) classifier, which performs the selection based on the combination of one or more header fields.

**Traffic Conditioners**

Traffic conditioners form the most vital part of a differentiated services network. Their goal is to apply conditioning functions on the previously classified packets according to a predefined TCS. A traffic conditioner consists of one or more of the following components:

- **Meter**
A device which measures the temporal properties of a traffic stream selected by a classifier.

**Marker**

A device that sets the DS Codepoint in a packet based on well-defined rules.

**Shaper**

A device that delays packets within a traffic stream to cause the stream to conform to some defined traffic profile.

**Dropper/Policer**

A device that discards packets based on specified rules (e.g. when the traffic stream does not conform to its TCS).

**Per-Hop Forwarding Behaviours (PHB)**

A PHB is a description of the externally observable forwarding behaviour of a differentiated services node, applied to a collection of packets with the same DS Codepoint that are crossing a link in a particular direction (called differentiated services behaviour aggregate). Each service class is associated with a PHB. PHBs are defined in terms of behaviour characteristics relevant to service provisioning policies, and not in terms of particular implementations. PHBs may also be specified in terms of their resource priority relative to other PHBs, or in terms of their relative observable traffic characteristics. These PHBs are normally specified as group PHBs and are implemented by means of buffer management and packet scheduling mechanisms.

### 2.3 Mobile Ad Hoc Networks

A Mobile Ad Hoc Network (MANET) is a distributed, wireless, mobile, multihop network architecture that relies on no pre-existing network infrastructure for its deployment. The only requirement for a MANET being deployed is the existence of at least two mobile nodes (MN) in communication range with each other that will form the MANET. The nodes comprising the MANET are characterised by their dynamic nature, meaning that they can move in and out of the network at any time and with no pre- or post-condition being met [Perk01] [Mack98]. The network topology itself is not static, but it can change dynamically with time, being dependent on the high degree of mobility the mobile nodes exhibit [Chak01] [Hass02] [Mack98b] (Figure 2-3). In [Chlam03] an extensive up-to-date literature review on mobile ad hoc networking is presented.
The basic characteristics of MANETs [rfc2501] are dynamic topologies, the fact that they are bandwidth-constrained, variable capacity links and energy-constrained operation. The research interest towards MANETs is partly evident by the fact that IETF has devoted a Charter Group [CGroup] to deal with issues regarding MANETs. MANETs can be viewed as an extension to the current Internet since they can be connected to it through one wireless hop (Figure 2-4). Due to its inherent nature a MANET is considered as an unreliable and unstable environment as far as communications are involved. It comes as no surprise that a plethora of researchers are dealing with the issue of routing in such networks.

The dominant technologies that enable the deployment of MANETs as they have emerged in recent years are Wireless Local Area Networks (IEEE 802.11 WLAN) and Bluetooth. Table 1 summarises these technologies. It should be stated though that the family of IEEE 802.11 standards has been dominating the market, with other rivals assuming a small share although there is the option of coexistence amongst them. The main reason for this is the fact that IEEE 802.11 operates in the unlicensed ISM 2.4GHz band (802.11a operates in the 5GHz band) enabling thus easy and cheap deployment configurations.
In the research area of mobile ad hoc networks, fundamental is the work undertaken by Hubaux and his fellow researchers as part of the Terminodes project [Blaz01]. The notion of combining terminal and node capabilities in every mobile node is the driving force of this project. While self-organisation stands as a key issue in this research, as well as routing, security and incentives for cooperation, there are aspects such as context-awareness that are overlooked. An important act of this research is the holistic approach it adopts as far as MANETs are concerned in contrast to the majority of other efforts that only deal with the application-specific issues and disregard the underlying network issues.

<table>
<thead>
<tr>
<th>Enabling Technology</th>
<th>Managing Body</th>
<th>Communication Range</th>
<th>Link Capacity</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLAN</td>
<td>IEEE 802.11 Working Group</td>
<td>100-500 m</td>
<td>From 2Mbps to 54Mbps</td>
<td>[Anast03], [80211a], [80211b], [MattGast]</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>Bluetooth Special Interest Group</td>
<td>Approximately 10 m</td>
<td>Up to 1Mbps</td>
<td>[Bisd01], [BT04], [Conti03]</td>
</tr>
</tbody>
</table>
PART II – THESIS CONTRIBUTIONS

Chapter 3

3 Middleware Technologies

Middleware can be defined as a software that connects two otherwise separate applications or separate products and serve as the glue between the two systems. It is, therefore, distinct from import and export features that may be built into one of the applications. Middleware is sometimes called plumbing because it connects two sides of an application and passes data between them. This chapter presents a review of four well known middleware technologies, namely: Mobile Agents, CORBA, SOAP and XML-RPC. For each technology, one commercial platform was selected, where a prototype application was built on. In this way, we could assess the performance and usability of each one and draw conclusions on their suitability for certain applications.

3.1 Mobile Agents

A Mobile Agent (MA) is a software entity that has the ability to migrate from one node to another in a network of heterogeneous computer systems in order to carry out specific network operations. The agent’s intelligence may vary from simple predefined rules to self-learning Artificial Intelligence machines. Agents may co-operate with each other, execute asynchronous or synchronous operations and migrate to remote nodes in order to accomplish a task. An agent can suspend its execution at an arbitrary point, transport itself to another machine and resume execution on the new machine. [Boho00c]. Agents can also be static if they are not required to traverse the network but need to serve a particular function in a network node. An agent migration is illustrated in Figure 3-1, where a static agent, responsible for the management system, “spawns” a mobile agent in order to perform a management task in a remote host.
Chapter 3. Middleware Technologies

Figure 3-1: Mobile Agent migration example

The first Mobile Agent system was developed by General Magic [White94] in 1996, introducing
the Telescript Agents, a simple but yet a complete agent system. Some of the recent mobile agent
systems developed are the Grasshopper platform [Breu98] by IKV, the Voyager platform
[VOYAG] by ObjectSpace and the Aglets [AGLETS] by IBM. Research in the area of mobile
agents looked at languages that are suitable for mobile agent programming, and languages for
agent communication. A significant amount of effort was put into security, control and design
issues. Each of the above mentioned systems focus on different aspects of mobile agents, e.g.
Aglets on security, Grasshopper on the implementation of the FIPA [FIPA01] [FIPA02] and
MASIF [MASIF] standards.

3.1.1 Potential Benefits

Mobile Agents Technologies (MAT) may help overcome the restrictions of the client-server
paradigm and the overheads of a centralised approach. More specifically the MA technology is
considered to have the following advantages:

- **Extensibility of functionality**: by delegating a new task to an agent and sending it to the
  remote node, the functionality of the server can be enhanced without upgrading the
  software. For example a network manager may send an agent that performs filtering to an
  SNMP capable device to increase its functionality. The agent will locally interrogate the
  server and then return the requested value to the manager. To add another feature to the
  server all we have to do is send another agent, or remove the existing one and send a more
  intelligent one.

- **Reduction of load in the NMS**: by moving the intelligence near the data, the processing
  load on the network management station is minimised. The agent will do all the
  processing and return the result to the network manager. Some processing load is still
  required on behalf of the network manager. This makes the architecture more scalable.
• **Reduction in network traffic**: since most of the interactions are local, the amount of network traffic due to management operations is reduced. On the other hand, the traffic increases due to the migration of the agent from node to node, but in most cases the data that needs to be processed is less than the amount of the agent’s code.

• **Robustness and fault tolerance**: Agents can interact asynchronously without loss of accuracy, so even if the network is not operational, the agent will continue to function and will report to the manager when the network is available. In addition, the agent will continue to perform his task even if the delegating entity is no longer active. This increases the robustness and fault tolerance of the system.

• **Increased Responsiveness**: The agents reside near the network elements so they can respond to network events, avoiding delays caused by network congestion. In response to an event of a failure, agents can interact with each other in order to reconfigure the network. This makes network management much more distributed and fault tolerant than the traditional approaches.

### 3.1.2 Problematic 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:

• **Standardisation and Interoperability**: Inconsistency has greatly hindered the adoption of mobile agent technology with limited standardisation so far. Even the few standardisation 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) 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.

• **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 utilisation, but this often comes at the expense of higher utilisation 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 centralised 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 centralised protocols, to distributed object frameworks, followed by mobile code solutions and later by mobile agents.

### 3.2 CORBA

The OMG's Common Object Request Broker Architecture (CORBA) [OMG95], is an open standard for distributed middleware that allows objects to interoperate across networks regardless of the language in which they were written or the platform on which they are deployed. CORBA provides client-server interoperability between remote systems through the use of standardised protocols and multiple object access available in CMIP and SNMP. In addition, CORBA supports the standardisation of APIs for performing remote procedure calls from clients to server objects. This is done through standardising the mapping between various programming languages and the Interface Definition Language (IDL) used to describe CORBA server interfaces. Compilers can generate client stub and server skeleton code in a number of languages, e.g. C, C++, Java, and Ada.

CORBA allows applications to communicate with one another without being aware of the hardware or software systems or the location of the application, using the standard IIOP protocol. A client can transparently invoke a method on a server object. The two objects can reside at the same node or at different nodes across the network, both using the services of the Object Request Broker (ORB). The ORB intercepts the call, locates the (remote) server object that can serve the request, invokes its method passing the required parameters and returns the results to the client.
Thus, the ORB provides interoperability between applications on different machines in heterogeneous distributed environments and brings together multiple object systems. The basis for interoperability comes from Interface Definition Language (IDL). The CORBA ORB architecture can be seen on Figure 3-2.

CORBA 1.0 was introduced in 1991, defining the Interface Definition Language (IDL) and the Application Programming Interfaces (APIs), which enabled the client/server object interaction within a specific implementation of an Object Request Broker (ORB). Version 1.0 included a single language mapping for the C language. CORBA version 2.0 was released in 1996, introducing the Internet Inter-ORB Protocol (IIOP) and new IDL language mappings for C++ and Smalltalk, while the v2.3 extension, in 1999, added support for Java. CORBA version 3.0 was finally released in 1999, introducing the Portable Object Adapter (POA), the CORBA Messaging specifications and “Object by Value” Support. The Portable Object Adapter is used mainly to provide portability for CORBA server applications, while the Messaging Specifications adds asynchronous messaging, time-independent invocation and facilities for specifying messaging Quality of Service (QoS) to CORBA.

CORBA finds its position in many situations due to its flexibility of integrating machines from different vendors, operating systems and sizes ranging from mainframes to desktops, hand-holds and embedded systems. One of its most important, as well most frequent, uses is in servers that must handle large number of clients, at high hit rates, with high reliability. Nevertheless, it is not just used for large applications, as specialised versions of CORBA can be suitable for real-time systems, and embedded systems.
3.3 XML

3.3.1 SOAP

The Simple Object Access Protocol (SOAP) [SOAP] for Web Services is an emerging distribution middleware technology based on a lightweight and simple XML-based protocol, which allows applications to exchange structured and typed information using the World Wide Web (WWW). SOAP was designed to enable automated Web services based on a shared and open Web infrastructure. SOAP applications can be written in a wide range of programming languages, such as C++ and Java and be used in combination with a variety of Internet protocols and formats, such as HTTP, SMTP, and MIME. SOAP supports different styles of information exchange, including:

- **Remote Procedure Call (RPC)**, which allows for request-response processing, where an endpoint receives a procedure-oriented message and replies with a correlated response message.

- **Message-oriented information exchange**, which supports organisations and applications that need to exchange business or other types of documents where a message is sent but the sender may not expect or wait for an immediate response.

A SOAP message consists of a SOAP envelope that encloses two data structures, the SOAP header and the SOAP body. The SOAP header, which is optional, contains information about the request defined in the SOAP body, such as contextual, transactional, security or user profile information. The SOAP body contains the actual Web Service request or the reply to a request in XML format. The high-level structure of a SOAP message is shown in Figure 3-3. The SOAP messages, carrying Web Service requests and responses, can conform to the Web Service Definition Language (WSDL) [WSDL]. The latter provides the definition of the available Web Services and can be compared to the CORBA IDL. Definition in WSDL include the SOAP message used to access the Web Services, the protocols over which such SOAP messages can be exchanged, and the Internet locations where these Web Services can be accessed. The WSDL descriptors can also reside in a Universal Description, Discovery and Integration of Web Services (UDDI) [UDDI] or other directory services, and they can also be provided via configuration or other means such as in the body of SOAP request replies.
SOAP makes use of the XML language for machine-to-machine communications, as the means for describing data. XML, which is definition-driven language through the use of DTDs and schemas, allows information to be manipulated programmatically. Because XML is human-readable and text-based, it is ideal as a transport framework for loosely coupled Web services.

The SOAP specification provides a standard way to encode requests and responses. It describes the structure and data types of message payloads using the XML Schema. The way that SOAP is used for the message and response of a Web Service is the following:

- The SOAP client builds or picks an XML document that conforms to the SOAP specification and which contains a request for the service.
- The SOAP client sends the XML document to a SOAP server, and the SOAP servlet running on the server side handles the document, using for example HTTP or HTTPS.
- The Web service receives the SOAP message, and dispatches the message as a service invocation to the application providing the requested service.
- A response from the service is returned to the SOAP server, again using the SOAP protocol, and this message is returned to the originating SOAP client.

The initial version of SOAP was introduced in 1998, when there was no schema language or type system for XML, which was actually just becoming an official recommendation at that time. A significant amount of effort was put on defining a type system, which had a handful of primitive types, composites that are accessed by name (structs) and composites accessed by position.
(arrays). In May 2000, a number of companies and organisations, such as HP, IBM, IONA and Microsoft proposed the SOAP version 1.1 to W3C and the formation of a working group in the area of XML-based protocols. Moreover, the first public Working Draft of SOAP version 1.2 was published from W3C in December 2001 and finally the official v1.2 recommendations in June 2003.

3.3.2 XML-RPC

XML-RPC [XMLRPC] is distributed middleware technology that allows software running on disparate operating systems, running in different environments to realise remote procedure calls (RPCs) over the Internet. RPCs are achieved using the HTTP protocol for the transport and XML for the encoding part. XML-RPC can be considered as a subset of the SOAP protocol, unburdened from unnecessary complexity. XML-RPC was originally a research project named SOAP, when in early 1998 due to political reasons it was divided to what it is known today as XML-RPC and SOAP. The first kept its simplicity by all means as a protocol, while the latter developed to a fully featured middleware protocol. XML-RPC is now a final specification, which is less verbose and easier to implement than SOAP. Both SOAP and XML-RPC work by turning a set of parameters (scalars, strings, dates, arrays, records, and binary data) into XML for transmission. XML-RPC is defined as operating over an HTTP connection, while SOAP describes the envelope format for an RPC request, which may be sent over HTTP, SMTP or some other protocol. An important distinction between the two is that SOAP passes parameters by name, while XML-RPC by position. This is crucial, as a routine that depends on the order of parameters in XML-RPC must be called carefully to ensure correct results. SOAP allows for user record types by extending the XML document using XML Schemas. XML-RPC only allows for the base types defined in the specification, which is usually adequate in most of the cases. XML-RPC’s greatest feature is its simplicity. It is extremely easy to understand, implement, and debug. The syntax is so uncomplicated that it is very easy to find, and avoid, mistakes.

3.4 Comparative Analysis

The four middleware technologies, presented in the previous sub-sections, appear to have major differences in their approach to management. This section presents a performance and feature comparison, based on prototype implementations in all four technologies, in order to investigate how they perform against each other and identify their suitability for specific applications.
3.4.1 Features Comparison

As mentioned above, the four middleware technologies presented here are characterised by their diverse features. This includes, for example, different approaches to message encoding and transfer, location transparency, security and parameter passing. Figure 3-4 presents the features supported by each middleware technology.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Mobile Agents</th>
<th>CORBA</th>
<th>SOAP</th>
<th>XML-RPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>System description</td>
<td>Java interface</td>
<td>IDL</td>
<td>WSDL/XML schemas</td>
<td>N/A</td>
</tr>
<tr>
<td>Error handling</td>
<td>Java exception</td>
<td>IDL exception</td>
<td>SOAP fault messages</td>
<td>XML-RPC fault messages</td>
</tr>
<tr>
<td>Parameter Passing</td>
<td>by reference or value</td>
<td>by reference or value</td>
<td>by value</td>
<td>by value</td>
</tr>
<tr>
<td>Message encoding</td>
<td>binary format</td>
<td>CDR/binary format</td>
<td>XML/Unicode</td>
<td>XML/Unicode</td>
</tr>
<tr>
<td>Security</td>
<td>X.509 certificates, SSL</td>
<td>CORBA security service</td>
<td>HTTP/SSL, XML signature</td>
<td>HTTP/SSL, XML signature</td>
</tr>
<tr>
<td>Location transparency</td>
<td>Agent identifier</td>
<td>Object references</td>
<td>URL</td>
<td>URL</td>
</tr>
<tr>
<td>Events</td>
<td>FIPA ACL</td>
<td>CORBA event service</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Language support</td>
<td>Mainly Java</td>
<td>any language with an IDL binding</td>
<td>any language</td>
<td>any language</td>
</tr>
<tr>
<td>Service discovery</td>
<td>Agent registry</td>
<td>CORBA naming/trading service</td>
<td>UDDI</td>
<td>N/A</td>
</tr>
<tr>
<td>Registry</td>
<td>Agent registry</td>
<td>Interface Repository</td>
<td>UDDI/WSDL</td>
<td>N/A</td>
</tr>
<tr>
<td>Transfer protocol</td>
<td>JRMP</td>
<td>GIOP/IOP</td>
<td>HTTP</td>
<td>HTTP</td>
</tr>
</tbody>
</table>

Figure 3-4: Middleware technologies features comparison

In SOAP, service interfaces are specified in the Web Services Description Language (WSDL), which constitutes a general XML-based framework for the description of services as
communication endpoints capable of exchanging messages. It describes the service location through a Uniform Resource Identifier (URI), supported operations, and messages to be exchanged. WSDL can be considered as broadly equivalent to IDL for CORBA and Java Interface for Mobile Agents. In this context, URIs are broadly equivalent to CORBA IORs and Mobile Agent identifiers. SOAP and XML-RPC are stateless protocols with XML-based encoding, in contrast to CORBA and Mobile Agent that use binary encoding. They can support request/response/error remote call interactions and are broadly equivalent to CORBA GIOP when used that way. The default SOAP mapping on HTTP/TCP/IP can be considered equivalent to CORBA IIOP and JRMP.

SOAP service specification and interface discovery are supported through Universal Description, Discovery, and Integration (UDDI). When used for service specification discovery, it is broadly equivalent to the CORBA Interface Repository; when used for interface location discovery, it is broadly equivalent to the CORBA Naming and Trading services. The equivalent is the registry for Mobile Agents, while this feature has been removed from XML-RPC. It would be interesting to note that XML-based technologies and CORBA can support any programming language, contrary to Mobile Agents that are mainly limited to Java. Last but not least, Mobile Agents and specifically the Grasshopper platform use X.509 certificates and SSL for security, while CORBA uses its own security mechanisms. On the other hand, security for the XML-based technologies is still immature. Current solutions include HTTP over SSL and XML digital signature. In the next subsection, we present a number of performance measurements that were conducted in order to evaluate the middleware platforms.

3.4.2 Performance Evaluation

For the performance evaluation of the middleware technologies, different prototypes were implemented that perform equivalent operations for all technologies. The measurements taken include the response time required for invoking a simple operation on a remote machine, the traffic produced by the latter and the memory usage requirements of each middleware platform. Other measurements include the effort required by the developer to program in each middleware platform, quantified by the number of lines of each application. This includes only the lines written by the developer and not the lines automatically generated by the middleware platform.

For the experimental environment, the software platforms that were used include the Grasshopper v2.2 Mobile Agents platform by IKV++ [GRASS], the Java IDL platform for CORBA in Sun Microsystems's JDK v1.3.1 [Java], the WASP Web services platform by Systinet [SYST] and the XML-RPC platform provided by Apache [APACHE]. All the RPCs were realised between two different Pentium III, Celeron 1 GHz, Linux PCs with 256 Mbytes of RAM, connected in “back-
to-back" mode through a dedicated 100 Mb/s Ethernet link. The Java 2 Standard Edition JDK 1.3.1 version was used on the Linux PCs, running the Red Hat 7.3 Linux operating system and compiled with the stable kernel version 2.4.17. All the measurement results presented here are an average of 100 recorded samples.

### 3.4.2.1 Response Time

In order to measure the time required to carry out a simple operation, by each middleware platform, four Java programs were developed. The task was to obtain a byte array, containing 10 longs, from a remote system. This was realised by a simple remote procedure call (RPC) in a client-server fashion. In order to keep the comparison fair, we decided not to migrate a mobile agent at the remote host, in order to fetch the byte array, but to use a remote procedure call as well. The overhead of migrating is discussed later on, as it would not be comparable with a simple RPC. The response time measured for each middleware platform that was required for the selected operation can be seen in Figure 3-5.

![Figure 3-5: Response time for the four middleware technologies](image)

CORBA required 4 milliseconds to complete the task, which is almost 4 times faster than Mobile Agents. XML-RPC response time is similar to CORBA with 5 milliseconds, while SOAP is nearly double with 9 milliseconds, but still half than Mobile Agents. CORBA manages to outperform clearly all its rivals, but with the XML-RPC fluctuating at similar levels. On the other hand, Mobile Agents and Grasshopper specifically, exhibits the worst response time performance, being 4 times slower than CORBA and 2 times slower than SOAP.
3.4.2.2 Traffic Requirements

For the next measurement, the traffic generated for the selected operation, by each middleware platform, can be seen in Figure 3-6.

![Figure 3-6: Overhead traffic produced by the four middleware technologies](image)

Regarding the overhead traffic, Mobile Agents produce 352 bytes, which is similar to the 316 bytes of CORBA. SOAP produces 2052 bytes, while XML-RPC is nearly half of SOAP at 953 bytes. CORBA is again the winner but with Mobile Agents being only 10% more bandwidth demanding. On the other hand, SOAP, and WASP specifically, exhibits the worst traffic performance by producing 6 times more overhead traffic. XML-RPC requires half the bandwidth but is still 3 times more expensive than CORBA and Mobile Agents. The reason why XML-based technologies require much more bandwidth than CORBA or Mobile Agents is because the latter use binary format for the transfer syntax, while the former use Unicode/plain format. This usually corresponds to a plain-text, very verbose, XML document, which has to be transferred between network nodes. A way of tackling the problem of excess traffic in XML-based technologies is the use of compression on the exchanged XML messages. This could seriously reduce the overhead traffic but on the other hand, it would require more processing power at the network nodes in order to (de)compress the XML messages.

3.4.2.3 Memory Usage

Regarding the resource requirements at a server side, we have measured the memory usage required for the execution of the middleware applications on the supporting platform. The
memory usage results can be seen in Figure 3-7. It should be noted here that the memory usage of each middleware application was calculated as the difference 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. This was considered necessary as the Java Virtual Machine occupies a large amount of memory upon initialisation, which is similar to all Java programs and hence the four examined middleware platforms. Only by measuring the occupied memory in the Java Virtual Machine we could conduct a fair comparison.

![Figure 3-7: Memory usage at server side](image)

The Grasshopper Mobile Agent platform requires 4567 kbytes, which is about 3 times more than the 1456 kbytes of CORBA. WASP/SOAP requires a substantial 3020 kbytes while Apache/XML-RPC requires as little as 192 kbytes. It is clear that Grasshopper is the most heavy-weight platform but one could argue that it provides the richest set of capabilities, such as software migration. CORBA appears to be quite inexpensive in terms of memory usage, as it requires 3 times less than Grasshopper and 2 times less than WASP. The big surprise in these measurements was the memory usage exhibited by XML-RPC, which is 23 times less than Grasshopper, 15 times less than WASP and even 7 times less than CORBA. This really validates the statement that XML-RPC is designed to be light-weight. Nevertheless, it provides much more limited functionality than the other technologies, but in specific applications it could be considered as adequate.
The memory requirements of the middleware platforms are crucial if we consider applicability in emerging networks, such as ad hoc, where devices typically offer a modest amount of computational resources. In the case where the network nodes are equipped with plenty of memory and processing power, then it might be legitimate to consider the use of Mobile Agents. It should be stated here that the evaluation carried out for the memory usage was heavily dependent on the middleware platforms used. The obtained results could be different if other platforms, with different capabilities, were selected. We tried to use a set of common middleware platforms and avoid any optimised ones, such as CORBA for embedded systems.

3.4.2.4 Platform Usability

For the last measurement, we assess each platform in terms of usability. A chosen metric is the lines of code that a developer has to write in order to implement the functionality of our test cases. This includes only the lines written by the developer and not the lines automatically generated by the middleware platform. The measured lines of source can be seen in Figure 3-8.

![Figure 3-8: Lines of written code](image)

Mobile Agents, SOAP and XML-RPC appear to have about the same requirements in terms of coding quantity, with 78, 81 and 80 lines respectively. On the other hand, CORBA requires 96 lines of code for realising the desired functionality, which is 19% more than Grasshopper. It should be noted here that the written source code follows the Java programming Style recommendations [Verm00] and is not optimised for space saving. The difference in lines of written code also reflects the easiness of coding in each of the middleware platforms. Based on
the author's experiences, the simplest platform to program on is probably the Apache/XML-RPC followed by the Grasshopper/MA. The latter though, would require more effort when designing a system with several Mobile Agents travelling around the network. By comparing the WASP/SOAP to the JAVA/CORBA platform, it can be said that the former appears more flexible and user friendly than CORBA. It is also evident from the amount of lines of code that CORBA requires more effort in programming.

3.4.2.5 Software Migration

In this subsection, we present some basic performance measurements that relate to the migration of Mobile Agents between two network nodes. In the previous subsection, we presented a test case where the client-server communication was realised through a remote procedure call. In this case and through the use of Mobile Agents, a piece of software from the client migrates to the server side, performs the desired operations and returns back with the result. This is depicted in Figure 3-1. For this scenario, the measurements taken include the time required for a complete operation, as well as the traffic produced. These were measured to be 1,418 milliseconds and 2,932 bytes respectively. 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 [Pavlou98]. In this respect a mobile agent takes for its setup about 100 times longer and incurs about 25 times more traffic compared to a distributed 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. 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.

3.5 Conclusions

This chapter presented a review of four promising middleware technologies. A selected platform of each technology was evaluated in terms of performance and usability. It was shown that CORBA exhibits better response time and traffic performance than Mobile Agents and SOAP, while XML-RPC was fluctuating at similar levels. It was also found to be less expensive in terms of memory usage compared to Mobile Agents and SOAP. The initial performance evaluation of SOAP is encouraging but also highlights some expected problems. Information retrieval times are approximately twice those of CORBA, but the key problem is the amount of management traffic incurred due to the XML-based encodings, which can be up to six times that of CORBA. This can be reduced through compression at the expense of slower retrieval times. XML-RPC, which can be regarded as a subset of SOAP, appears to be twice more efficient than the latter and in some cases very close to CORBA. The major advantage of the Apache/XML-RPC platform that was observed during the measurements is its very low memory usage compared to the other platforms.
This indeed makes the platform exceptionally light-weight and suitable for resource constrained devices that can be found in a typical ad hoc environment. It can also be said that programming wise, XML-based approaches appear more flexible and user friendly than CORBA, although SOAP usability is similar to CORBA due to the stub-based APIs. It is also evident from the amount of lines of code that CORBA requires more effort in programming.

Grasshopper, which is a popular and all-round Mobile Agent platform [Guth98] [Mich00] [Silv00], exhibits significantly higher performance costs than the other middleware approaches of our comparison. The use of Mobile Agents in a resource constrained environment should be treated with care, as the cost of migrating, presented in 3.4.2.5, is bandwidth and time consuming. Of course the latter technology offers the capability of software migration across a network that can possibly lead to the reduction of remote interactions and hence traffic overhead. It might also be possible for an Intelligent Agent to migrate to a powerful node and execute a complicated task in less time. This of course would require a more complex logic and possible utilisation of context information.

In summary, SOAP and XML-RPC are promising technologies but, being XML-based, have more overheads than Mobile Agents and CORBA. On the other hand, being XML-based is also their biggest attraction, due to potential easy integration with other applications. Mobile Agents is probably the richest middleware technology in term of capabilities, but is let down by the significant platform resource requirements. In Chapter 4 next, we present a Mobile Agent based approach for monitoring and configuration of next generation fixed IP networks.
Chapter 4

4 Middleware for QoS Monitoring and Configuration of NG Fixed IP Networks

Telecommunication network infrastructures become more and more distributed and heterogeneous, with various network technologies integrated (e.g. ATM, IP, ADSL), and several types of equipment coexisting, typically from different vendors. At the same time, services are becoming more and more demanding in terms of quality of service. In this context, efficient network management is of paramount importance for network operators. The telecommunication business does not any more depend on the type and cost of the services but mostly on their quality and how quickly they can be introduced. In this context, management platforms should be flexible and intelligent enough to dynamically configure, monitor and eventually reconfigure the network. On one hand, these platforms should be able to inform the clients about the quality of the requested services and on the other hand, network administrators must be able to manage their network resources in order to efficiently provide Quality of Service. Service Level Agreements (SLAs) between network operators / service providers and clients should be honoured in a manner absolutely transparent to the users.

This chapter presents an Agent-based middleware system for managing the Quality of Service in IP DiffServ capable Networks. The system is responsible for configuring and monitoring a desired level of QoS, which can be offered to a network customer through a Service Level Agreement (SLA). The Mobile Agent Technology (MAT) is used to realise the QoS management system while IP Differentiated Services (DiffServ) technology is used as a test case.

4.1 Related Work

The European Commission (EC) has acknowledged agent technology as being extremely important towards tackling a host of telecommunication issues related to service engineering and network management. This is manifested by the fact that numerous agent projects were funded in the scope of the 3rd call for proposals of the ACTS programme. These projects aimed at exploiting the potential benefits of software agents in a wide range of application fields. A cluster of agent-related European projects was established (i.e. the CLIMATE cluster see:
This cluster of research activities includes both intelligent agent based projects, as well as projects which adopt mobile agent based implementations. As far as the latter are concerned, they have targeted different application fields, thus making apparent that the benefits of mobile agents could be applicable to a broad spectrum of applications. Projects that constitute representative examples of EC funded mobile agent projects are:

The MARINE project aimed at distributing intelligence in the network by applying distributed object technologies and mobile agents to the network environment. Specifically, the project considered mobile agent based deployment of IN service logic offering services to both fixed and mobile users. Existing IN devices were enhanced in order to become adapted to the distributed environment.

The MONTAGE project validated the benefits of mobile agent technology in a number of different telecommunication problems. It aimed at creating agent-based Accounting, Charging and Personal Mobility support services and validating them through trials involving real users and service providers. Specifically, the project focused on: (a) Exploiting agent technology to support efficient service provision to fixed and mobile users in competitive telecommunications environments. (b) Demonstrating the usefulness and applicability of agent technology in handling complexities, related to service provision, accounting and charging in the context of personal mobility, and (c) Assessing the trade-offs (regarding system complexity, performance, ease of use, etc.) for all involved stakeholders by building and using agents and agent-based services.

The MARINER project addressed the objectives and the requirements for load control strategies in the context of a Distributed-IN environment. The focus is on the specification of a multi-agent architecture incorporating a number of different agent types, each responsible for a well-defined subset of the tasks necessary for the realisation of the overall load control strategy. The multi-agent architecture, implementing an appropriately selected load control strategy, was realised in terms of a prototype multi-agent system implementation built upon a suitably real-time enhanced agent system development and execution platform.

The MIAMI project's context was the emerging Open European Infrastructure (EII) and the Global Information Infrastructure (GII) which is characterised by its increasing distribution, its dynamic nature, and the complexity of its resources. To manage such an environment, increased intelligence in management solutions and the mobility of such solutions are becoming major requirements. The MIAMI project strived to create a unified mobile intelligent agent (MIA) framework. To achieve this, the OMG MASIF standards were validated, refined and enhanced according to the requirements for an Open EII. Intelligent mobile agent-based solutions for
management of the Open EII and for the provision of advanced communication and information services were developed. Thus, the project provided a reference implementation for the Unified MIA Framework (UMF) and for workflow management which eventually will benefit the entire Pan-European business environment. Research work in the context of the MIAMI ACTS project (Mobile Intelligent Agents in the Management of the Information infrastructure) focused on the design and implementation of mobile agents for an ATM performance management system within a single Connectivity Provider (CP) domain. The overall CP architecture followed TMN hierarchical layering principles and the underlying network technology was Asynchronous Transfer Mode (ATM). The work was conducted using commercial ATM switches and the mobile agents exhibited "weak mobility", weak in the sense that performance monitoring agents were sent to execute "within" network elements (i.e. ATM switches) and stayed there until their task was accomplished. The MIAMI performance management work was loosely based on the OSI Systems Management (OSI-SM) / TMN Metric Monitoring X.739 and Summarisation X.738 recommendations.

It is also noteworthy that within the ACTS 3rd Call projects considerable effort has been put into resolving issues related to the agent technology per se. Mobile Agent platforms and Agent Communication Languages hold a prominent position among these issues. It is no accident that several of these projects have conducted evaluation reviews of MA platforms, while other have declared themselves as FIPA case studies.

4.2 Functionality and Architecture

The QoS management system should be able to perform efficient Performance Management functions using Mobile Agents that allow service providers to negotiate, validate and test Service Level Specifications (SLS) agreed with the customers of the network. Performance Management is a vital need for a service provider, when the objective is to provide Quality of Service to the customer. The QoS management architecture makes use of middleware in order to achieve network technology independence, openness and flexibility but also protocol and vendor independence. The impact of introducing new IP-based technologies (i.e. DiffServ, MPLS) should be minimal, while the interfaces between the different layers should be open, flexible, and standard whenever possible in order to promote quick development of management applications. The platform should also be able to introduce new types of routers (i.e. Cisco, Linux, Redstone), eventually coming from different vendors by having no impact on the applications.
Figure 4-1: Architecture for Network Configuration and Monitoring

The Architecture for Network Configuration and Monitoring/Auditing is represented in Figure 4-1 and contains the following modules:

- **The QoS Configuration and Auditing Application(s):** This module contains a set of applications and user interfaces for requesting connections with a certain QoS level, based on a pre-agreed Service Level Agreement (SLA). It is responsible for monitoring the network resources in order to verify that the QoS parameters offered by a given SLA are being met.

- **The Parlay implementation layer:** This layer contains the Parlay API and a set of objects/agents that realise this API. The latter is thoroughly discussed in section 4.2.3.

- **The Adaptation layer:** This layer includes the Mobile Agents that migrate close to the routers for the purpose of configuration and monitoring. Several interfaces are offered to these agents through the LINUX-CISCO API, which allows for configuration of the routers independently from their type (i.e. vendor) and gathering of useful audit information from the resources (i.e. loss parameters). Due to the limited information that
is possible to gather from the routers, a set of Mobile Agents is used to simulate traffic and measure some additional parameters (e.g. delay and jitter).

### 4.2.1 Performance Monitoring Requirements

The performance monitoring application uses passive and active measurements in order to provide the performance information required by the Configuration and Auditing of QoS application. The system exploits mobility of agents through the use of a constrained mobility model [Boho00c], involving a suitable mobile agent that is sent to execute in a single network element. According to the constrained mobility model, a mobile agent is created and sent to a particular agent host, where its execution is confined only to this host. The mobile agent will perform its task locally at the network element, remotely sending performance reports in a scheduled manner, as well as on-the-fly notifications every time a performance threshold is triggered. The mobile agent carries the performance management logic for execution at a network element. This allows the functionality of the system to be easily customised or upgraded. Providing customisation of management logic imposes requirements for the following entities:

- **Static “master” agent**: is responsible of accepting requests and initiating the monitoring service, corresponding to the Network and Element Management Layer (NML and EML) functionality in terms of the TMN model.

- **Mobile “performance monitor” agents**: provides the functionality of a metric monitor and a summarisation object as specified in the X.739/X.738 standards [X739] [X738].

- **Static “target” agent**: provides “raw” performance information by wrapping the underlying technology (SNMP, CMIP, etc).

These software entities and their role can also be seen in Figure 4-2.
Figure 4-2: Mobile Agent-based performance monitoring system

The IP QoS performance parameters that are described within a service level agreement and required to be monitored are the following:

- **Availability**: The amount of time, usually in percentage, that the network is available.

- **Latency (Delay)**: IPPM defines one-way delay [rfc2679] as the time interval starting when the first bit of the packet is sent and ending when the last bit of the packet reaches the destination. Thus, the cut-through latency [rfc1242] can be computed by subtracting the transmission time of the packet, which is constant for a given packet size.

- **Round trip delay** [rfc2681] is defined as the time interval starting when the first bit of the packet is sent and ending when the last bit of the packet reaches the source again, after being received and retransmitted by the destination.

- **Instantaneous Packet Delay Variation (Jitter)**: IPPM defines ipdv [IPDV04] as the difference of the One-way-Delay of a packet and the One-way-Delay of the preceding packet in their stream.

- **Throughput**: is defined in [rfc1242] as “the maximum rate at which none of the offered frames are dropped by the device”.

- **Loss ratio**: is the percentage of sent packets discarded or not received at the destination.

In the following subsection we describe how the configuration of the QoS parameters is carried out through the mobile agents.
4.2.2 Configuration of QoS Parameters

The logical Differentiated Service Architecture is defined according to Figure 4-3. Each service existing in a DiffServ node is implemented by using several configurable elements. These elements will be accessed and configured by mobile agents (or configuration agents).

**Classifiers** are 1:N devices, which sort packets from the input stream into various output streams by matching component values of a packet’s classification key with filters. If the classification key is the DSCP value of the packets, the classifier is characterised as Behaviour Aggregate classifier (BA). If a combination of the packet header fields composes the classification field then the classifier is characterised as Multi-Field classifier (MF). A common type of MF classifier is a 5-tuple well-known classifier that classifies based on six fields from the IP header fields. A mobile agent may perform the following operation on a classifier of a DiffServ node:

- Configure filter parameters such as filter type (e.g. BA, Masked-DSCP, IPv4-6-tuple), classification key (DSCP value and/or other packet header values such as destination address, source address, IP protocol, source port and destination port in the case of the MF 6-tuple classifier) and output stream.
- Define a precedence of overlapping filters.

**Meters** are logically 1:N devices that measure traffic parameters (such as information rate and burst size) at which packets that make up a stream of traffic pass it. They compare these parameters to some set of thresholds and decide a level of conformance for the packets. We define three levels of conformance represented by colours, green represents conforming, yellow represents partially conforming and red represents non-conforming. Each conformance level is associated with a meter’s output. Different types of meters use different algorithms (usually a token bucket model) in order to decide the conforming level of a packet. In order to control the conformance level that the meter assigns to every packet, the mobile agents should define the following:

- Meter model and conformance algorithms used such as those specified in Single Rate Three Color marker (srtCM) [rfc2697], Two Rate Three Color Marker (trTCM) or others.
Chapter 4. Middleware for QoS Monitoring and Configuration of NG Fixed IP Networks

- Threshold values that the meter uses. As trTCM will probably be used for metering, those values could be: PIR (Peak Information Rate), CIR (Committed Information Rate).

- Define meter output

Markers are 1:1 elements that (re) colour the packet according to the results of the meter. The colour is coded in the DSCP field in a PHB specific manner. Mobile agents should be able to set or alter the value of the DSCP that is to be set by marker. According to trTCM marker, a packet is marked:

- red if it exceeds PIR
- yellow if it exceeds the CIR
- green depending on whether it exceeds or doesn’t exceed the CIR

Shaper/Dropper is an N:1 device which performs two separate tasks:

1. Delays packets in a traffic stream in order to bring the stream into compliance with a traffic profile, or because a source constrain (e.g. available bandwidth) prevents packet immediate forwarding.

2. Discard packets either because of buffering limitations or because a meter exceeds a configured profile.

Mobile agents should perform the following functions on a shaper/dropper component of a DiffServ node:

- Define and configure shaper’s parameters such as type, depth, delay and queue output.
- Define and Configure dropping algorithms (RED, RED-on-In-and-Out RIO, Drop-on-threshold) by taking into account the depth of shaper, thresholds and the PHB of the packet.
- Configure the dropping type (Tail, Head, Random dropping).
- Define shaper/dropper output.

Scheduler is an (N:1) element which gates the departure of each packet that arrives at one of its inputs, based on a scheduling algorithm namely service discipline such as first come, first served (FCFS), strictly priority, weighted fair bandwidth sharing (e.g. WFQ, WRR, etc). A set of parameters affects the scheduling of packets received at each input.

Mobile agents should be able to set and modify the parameters associated with the scheduling algorithm. These parameters are the following:

1. Static parameters such as relative priority associated with each of the scheduler’s inputs,
Chapter 4. Middleware for QoS Monitoring and Configuration of NG Fixed IP Networks

2. Absolute token bucket parameters for maximum or minimum rates associated with each of the scheduler’s inputs.

3. Parameters, such as packet length or DSCP, associated with the packet currently present at its input.

4. Absolute time and/or local state.

5. The scheduling algorithm to be selected. PHBs such as the class selectors EF, AF have descriptions or configuration parameters that strongly suggest the scheduling discipline to be enforced.

4.2.3 JAIN/Parlay API

A number of interfaces were considered for adoption and inclusion in the system, including the interfaces under development from: TSAS (OMG Telecommunications Domain Task Force Activities), OSA API (part of the 3GPP standardisation work), JAIN (Java API Initiative), Parlay API, Multi-Service Forum. Due to the functionality and status of the specification, the level of openness and the likely acceptance by the industry, the Parlay interfaces were adopted as the basis of inclusion in this system. A number of extensions, adaptations and clarification were made.

The Parlay Group [PARLAY] is an open multi-vendor consortium formed to develop open technology-independent Application Programming Interfaces (APIs) enabling third parties to develop applications and technology solutions to operate across multiple networking platform environments. Most of the Parlay specifications have been submitted and accepted by standardisation organisations such as 3GPP and ETSI. Key operators and vendors such as BT, Telecom Italia, France Telecom, IBM, SUN, Nokia, Ericsson, Lucent, Alcatel, Siemens, Nortel and Cisco are members of the Parlay consortium.

We adopted the Parlay Connectivity Manager API [PAR21] in order to configure QoS parameters in IP DiffServ Networks. This is an open API that may be used by any application developer (users, service providers, network operators, brokers, etc.). Besides following the Parlay specification we also decided to follow the approach suggested by the JAIN initiative [JAIN]. This group is supported by Sun Microsystems and has a similar objective to Parlay but more focused in scope, that is to define a set of open APIs in Java that abstract from specific network protocols and ease the fast development of next generation applications over a Java platform.

4.3 Middleware Components

In this section we describe the middleware components that are involved in the QoS Management system. The system is divided into a number of subsystems, which are mainly composed of
mobile and static agents. The Configuration and Reconfiguration Management Subsystem is responsible for (re-)configuring certain QoS parameters in the network elements. It caters for the initial configuration of the routers (i.e. definition of DiffServ parameters) as well as for the possible reconfiguration, in the case of SLA violation. The Performance Monitor Management Subsystem is responsible of monitoring the QoS parameters of the configured connections. It makes use of Active and Passive monitoring techniques for obtaining delay/jitter and bandwidth utilisation/loss statistics respectively. “Wrappers” are used, which are static agents that attach to network elements (i.e. Linux or Cisco routers), in order to provide a mediation layer between visiting mobile agents and the actual resources of network elements.

4.3.1 Performance Monitoring

The performance monitoring system constitutes an important part of the QoS configuration and auditing application. Performance monitoring activity can be initiated at any time required to ensure that the QoS offered by the network is in line with the properties of the SLA. The system is able to initiate its monitoring functionality by a request from the Resource Manager according to parameters present in the SLA. Responsible for the system initialisation and coordination is the MasterMonitorAgent, which can create on-the-fly the necessary monitoring mobile agents. The latter exploit mobility through the use of a constrained mobility model and follow the concept of passive and active monitoring in order to carry out their monitoring tasks. Passive measurements include the monitoring of throughput and loss ratio, which active measurements include the monitoring of availability, latency and jitter.

For the passive monitoring, the PassiveMonitorAgent migrates to an ingress access point where it requests raw performance information from the Monitoring Linux agent wrapper. Similarly for the active monitoring, the ActiveMonitorAgent migrates to an ingress access point where it exchanges an echo byte stream with the Monitoring Linux Agent, located at an egress access point. After a predefined period of time (report period), the monitoring agents compile the measurements captured up that point into monitoring reports, which are sent back to the MasterMonitorAgent where it makes them available to Resource Manager. The report and granularity periods are variable and could be modified on-the-fly. Figure 4-4 presents an overview of the system functionality. A UML sequence diagram of the performance monitoring operation can also be found in Appendix A.
The Performance Monitoring System depends mainly on mobile and static agents for performing the necessary monitoring tasks. More specifically the interfaces and objects involved in the monitoring system are the following:

**IMonitorManager**: this interface provides the means of connecting the Performance Monitoring system with the Resource Manager. It exposes the necessary methods for initiating and terminating the monitoring system according to a connection’s VPrP.

**IGenericMonitorAgent**: an interface to the super class of the ActiveMonitorAgent and PassiveMonitorAgent. It is responsible for the communication between the latter two agents and the MasterMonitorAgent.

The implementation of these interfaces result in the following mobile agents:

**MasterMonitorAgent**: this agent is responsible for the coordination of the Performance Monitoring System in the context of creating/terminating a monitoring service for a specific VPrP. The agent will create the appropriate agents for the purpose of passive and active monitoring and consequently contact the Resource Manager with the monitoring reports.

**GenericMonitorAgent**: this agent provides most of the common functionality of the monitoring agents. It is designed to be monitoring application independent and to reduce the size of the monitoring agents.

**PassiveMonitoringAgent**: this agent migrates to the Adaptation Layer of a router in order to obtain passive monitoring information (i.e. throughput and loss ratio per class of service) by querying the Monitoring/Linux agent wrapper.
Active Monitor Agent: this agent migrates close to the Service Access Point under monitoring and exchange echo byte streams with a Monitoring Linux agent, located on the second Service Access Point in order to obtain active monitoring information (i.e. availability, latency and jitter).

4.3.2 QoS Configuration

The QoS Configuration and Reconfiguration Management Subsystem is responsible for (re-)configuring certain QoS parameters in the network elements (i.e. routers). On one hand, it caters for the initial configuration of the routers (i.e. define parameters for the DiffServ classes of service) and on the other hand, if a certain path flow is violating the thresholds defined by the QoS template, the administrator may trace the route of a certain VPrP (if static routing is being used), get the queue load of the routers involved and reconfigure them in order to improve the end-to-end QoS. Through this subsystem it is possible to configure and reconfigure the Random Early Detection (RED) parameters, the queue size per class of service and the maximum bandwidth allocated to each class. Furthermore, the reconfiguration module estimates new bandwidth values according to both the queue load per class, and the required delay spacing among classes defined by the user. This module implements the mobile agents that are sent as close as possible to the routers to perform (re-)configuration tasks. The QoS Configuration functionality can be seen in Figure 4-5. A UML sequence diagram of the QoS configuration operation can also be found in Appendix A.
The adaptation layer contains a wrapper and a scheduler. Both are static agents which reside in the routers (in the case of Linux routers) or as close as possible to routers (in the case of Cisco routers). The general idea of the wrapper is to provide a single interface, no matter what kind of router exists in the network. The behaviour of the wrapper will be different depending on the specific router but the interface will be the same. The advantage of this approach is to ensure vendor independence. An alternative would be to create a mobile agent specific for each type of router. This possibility was dropped in order to keep the agents as simple and small as possible. Therefore, the specific behaviour to handle each router is kept in the static agents instead of in the mobile agents. The wrappers contain the following interfaces:

**Edge_ConfiguringSLS**: This interface should be used to request *connectivity* with a certain QoS, i.e., to configure edge routers according to user requests (or SLA).

**Monitoring**: It should be used to collect the passive monitoring measurements, i.e., packet loss rate and throughput.

**Core_Configuring**: This interface is used by the Administrator in order to provide a basic configuration to all the routers, Core and Edge.

**Core_Reconfiguring**: This interface is used by the Administrator in order to modify the QoS configuration of core routers, in case of SLA violation observed by the monitoring system.

The scheduler is used to manage the time that *connectivity* should be active according to the user requests (or SLA). It contains the following interface:

**ConnectionManagerInterface**: Receives *connectivity* requests from the upper level and communicates with the wrapper in order to realise it.

The mobile agents that are involved in this subsystem are the following:

**QoSConfigureConnectionAgent**: this is a mobile agent that migrates to the Adaptation Layer of each edge router in order to talk with the scheduler static agent and configure the edge router according to user requests for connectivity with a certain QoS.

**QoSRemoveConnectionAgent**: this is a mobile agent that migrates to the Adaptation Layer of each edge router in order to talk with the scheduler static agent and remove the configured connectivity.

**ConfRoutAgent**: this agent migrates to the Adaptation Layer of each router in order to give it a basic configuration. It should be also used to change this configuration at run time (e.g., queues characteristics and scheduler behaviour).

**ReConfRoutAgent**: this agent is used to change the basic configuration at run time (e.g., queues characteristics and scheduler behaviour). It is created by the Reconfiguration manager and is sent
Chapter 4. Middleware for QoS Monitoring and Configuration of NG Fixed IP Networks

to Adaption Layer of a core router of the DiffServ network in order to interact with the Linux-Cisco API.

QoS Monitor Agent: this agent is responsible for the coordination between the Connectivity Management subsystem and the Performance Monitoring subsystem. It is responsible for informing the latter about the creation or deletion of VPrPs that require monitoring.

4.4 Platform Evaluation

In this section we present the evaluation of the QoS Configuration and Monitoring system that was carried out in a DiffServ capable TestBed. We present several scenarios of the system execution and also discuss the contribution of mobile agents in the context of this case study.

4.4.1 QoS Monitoring and Configuration

This subsection deals with the evaluation of the QoS Configuration and Monitoring system in a DiffServ capable TestBed.

4.4.1.1 Network Environment

The TestBed presented in Figure 4-6 was used to evaluate the QoS Management System. By using the QoS Application it was possible to configure both Linux and Cisco DiffServ routers according to the QoS parameters defined by the user, to Monitor these parameters and Reconfigure the routers if needed.

Figure 4-6: DiffServ TestBed
The TestBed consists of three routers: two Linux routers, one having the role of an edge router and the other of the core router and one Cisco router used as an edge router. Two Users, User A and User B are connected to each of the edge routers, User A to the Linux edge router and User B to the Cisco edge router. After configuring the routers, the two users were able to use a common Video on Demand application in order to test how different QoS configurations reflect the quality of the delivered video. A traffic generator was used to create traffic in order to load the network and test its behaviour under extreme load conditions. The administration of the system was located in a different machine, where the QoS Management system could be managed through a GUI front-end.

Note that the Linux wrapper and the scheduler can be located in the same machine as the Linux router while in the case of the Cisco router they have to be located in a different machine—a Linux machine (i.e. Cisco Adapter). For all the Linux machines we used the Mandrake 8.0 Linux operating system, the Grasshopper v2.1 Mobile Agents platform by IKV++, the Java JDK v1.3.1 by Sun Microsystems and the MGEN v3.2.1 tool for injecting traffic into the network. For the users’ machines, we used two PCs, running the Windows 2000 operating system, having as a test application the Windows Media player. Finally the Cisco router used for these experiments was a 7200, running the IOS V12.2-4.T3.

4.4.1.2 Use Case Scenarios

Three scenarios were carried out for the validation of the QoS Management system. This includes configuration, monitoring and reconfiguration operations.

Configuration Scenario

By using the graphical front end, we could reserve network connections by choosing values for the following set of parameters:

- Class of Service (Bronze, Gold, Silver, Best-Effort, or others)
- Service Access Points (SAPs) origin and destination
- Directionality
- Minimum guaranteed bandwidth
- Time scheduling
Figure 4-7: QoS Management Application – Configuration GUI

Taking the example presented in Figure 4-7, the user “jpm” is interested in using the network resources, every Tuesday between 10 pm and 12 pm, starting from 2002 December 12. Since the user would like for example to watch a movie, he is interested in a good quality of service in one direction but not so good in the opposite direction. Therefore he requests a bi-directional VPrP between SAP1 and SAP2 and chooses BE in the forward direction and Gold in the opposite direction. Furthermore he knows that the movie will not need much more than 2Mb/s. So, he will request to use this bandwidth that is a percentage of the total one available for the Gold class of service. This means that the traffic below 2Mb/s, i.e. the well-behaved traffic will be marked as Green and will have the highest priority (mapped into the EF DiffServ class of service). The traffic, which exceeds this value in 10% will be marked as Yellow (mapped into one of the AF DiffServ classes of service). If it exceeds more than 10% it will marked as Red (BE) and will have the lowest priority. When the day comes the routers will be configured according to the user requirements. During the activation time the user will be able to monitor QoS of the traffic being sent. He will also be able to see if the contract is being violated with respect to the QoS requirements.
Monitoring Scenario:

Through the monitoring interface, shown in Figure 4-8, the user may ask to monitor the VPrP and observe how the QoS parameter values vary. Performance monitoring tasks involve both active and passive measurements. Active measurements are taken by inserting (low-impact) test streams for measuring latency, jitter and availability, while passive measurements are taken by analysing the traffic passing from a network element for measuring throughput and packet loss. The system provides performance reports and notifications in a scheduled and 'on-the-fly' manner, respectively.

![Figure 4-8: QoS Management Application - Monitoring GUI](image)

Reconfiguration Scenario:

An administrator can log into the application and get all the router interfaces involved in a VPrP by using the "trace route" functionality. He/she can enter the reconfiguration screen, as depicted in Figure 4-9, select one or some of the traced IP addresses and get the RED parameters, the queue size and the actual queue load per class of service. The user can then reconfigure the router RED parameters and the queue size per class of service. Optionally, he can select the proposed bandwidth value per class of service given by the management system. After reconfiguring the administrator can go back to the monitoring GUI to check if the QoS parameters improved and, specifically, if the required delay spacing among the reconfigured classes is achieved.
4.4.2 Mobile Agents

The MA Technology has been broadly applied to all the management systems that support the QoS applications.

The main advantages that come from the use of MAT in the QoS applications have been proven by the scenarios described in the previous sections and are the following:

- MAT allows real separation between central and remote processing. Tasks can be remotely performed in an autonomous way. Due to the nature of agents, central units can be easily disconnected while agents can still perform remotely their tasks. Therefore, mobile agents can be used when a considerable amount of information needs to be exchanged repeatedly between central and remote units. In this case, it is possible to take advantage of MAT by deploying ‘intelligent’ agents that can embody partially the functionality of traditional management systems and move away from the centralised approach by performing autonomously their tasks remotely. In the case of the QoS applications, mobile agents migrate to Linux or Cisco routers (or as close as possible), in order to perform management tasks as configuration, monitoring and reconfiguration.
MAT can also be effective in upgrading software functionality remotely. In other words, it is possible to enhance the capabilities of the management platform by remotely updating code without interrupting the normal processing of the overall system. Due to the nature of the IP DiffServ network technology, it is not easy to map QoS service requirements into network QoS parameters. Several algorithms can be implemented in order to give feedback to the administrator on how to manage his IP network. It is possible that several adjustments need to be done until the network behaves properly, meaning accordingly to a user's QoS needs. In this case, mobile agents can efficiently perform the task of monitoring and reconfiguring the network and report on the status of the network.

MAT minimises processing overheads when properly used. It is possible to use the same agent in a number of different remote locations by selecting an optimised path. MAT gives the possibility of creating multiple instances that visit multiple locations or the same instance can visit sequentially various locations. Especially for large system management this can be a real advantage as management tasks are usually intensive. A large amount of information is normally produced, which overloads the network. This feature has not been proved by the QoS test-beds since we are using controlled test environments but the amount of information that needs to be transferred to a single router gives a good indication on how MAT can help decreasing the load created by the management traffic.

MAT also avoids promiscuity in terms of software engineering design and implementation. It compels software engineers to program in an autonomous manner. This can be achieved also with other software technologies but MAT promotes autonomous and technology independent software design and implementation, which increases the flexibility of network management systems.

MAT is not yet broadly deployed and it is maybe not mature enough. There are not well-known agent-oriented methodologies and in order to take real advantage of MAT, agents should be carefully designed. The size and the tasks performed by mobile agents must be properly balanced. Otherwise, flexibility can be lost and system performance can be seriously affected. Another disadvantage is that the more autonomous the distributed systems become the more difficult it will be to manage them.

Summarising, the main advantages of the agent technology in the context of the QoS Application are:

- Flexibility: Due to the loose coupling of agents they can be easily replaced without disrupting the normal system operation.
Chapter 4. Middleware for QoS Monitoring and Configuration of NG Fixed IP Networks

- Distribution: An agent-based system is inherently distributed; therefore, agents can either be statically located in the network elements (i.e. routers) or migrate there, (or as close as possible).
- Decentralisation: agents can act autonomously; therefore, management does not have to be performed totally in a centralised manner.
- Heterogeneity: Specific Agents (e.g. wrappers) can deal with specific types of network elements, providing vendor independency and legacy compatibility. Although this might not be an exclusive agent advantage it is important to mention that MAT can also cope with this very important feature.

4.5 Conclusions

In this chapter we described the architecture, design and API issues for a management system based on mobile agent technology that supports quality of service management in IP networks. Using mobile agents as the basic design and implementation technology for our system helped us to decentralise configuration and monitoring tasks. In addition, it promoted good software design and made relatively easy the implementation of such a complex system. On the other hand, mobile agent platforms seem somewhat immature and not yet streamlined for efficiency. Despite that, the savings for complex configuration and monitoring tasks are significant in comparison to client/server protocol-based or distributed object technologies. But most important, mobile agent technology supported the “programmability” of network nodes for configuration and monitoring purposes in a rapid manner, assuming only raw management capabilities available. We believe that mobile agent technology will play a significant role for decentralisation and programmability in the next generation of management systems supporting multi-service IP-based networks.

The work presented in this chapter was undertaken as a collaboration within the MANTRIP [MANT] EU project. The author was the major contributor to the Performance Monitoring system and agent wrappers, while he was a key contributor to the Configuration system, see also publications [Mota02] and [Yang02].
Chapter 5

5 Programmable Middleware for Configuration and Monitoring of Ad Hoc Networks

Mobile ad hoc networks are characterised by their heterogeneity and the diverse capabilities of the nodes forming the network. Almost any device equipped with a wireless network interface can join an ad hoc network. In such an environment it is difficult to deploy common services without a common understanding among the participating nodes and their capabilities, in terms of processing power, battery life, expected residence time and also in terms of already installed and operational software. This chapter presents a middleware-based programmable infrastructure that allows the nodes of a mobile ad hoc network to download and activate required protocol and service software dynamically. This enables the alignment of the nodes’ capabilities and allows, for instance, full-scale quality of service-based communication among heterogeneous ad hoc network nodes.

5.1 Related Work

Programmability in ad hoc networks is of paramount importance given the multitude of potential solutions for routing, QoS support and other application services. There are various different approaches to achieve programmability. Active control packets may carry code to be evaluated in routers [Tenne97]; this approach has also been used for active routing in ad hoc networks [Tschu00]. Mobile agents may be used in full mobility scenarios, carrying code and state to manipulate different network nodes, or in a constrained mobility mode [Boho00c] as a more flexible means for the management by delegation approach [Golds93]; in the latter, code is uploaded and executed in network nodes through “elastic management agents”, augmenting the node functionality. Programmability is also possible through the provision of suitable management interfaces that allow code to be uploaded to network nodes and activated in a controllable fashion. This approach has been first adopted in the Xbind framework, targeting the quick and flexible introduction of new telecommunication services in programmable network infrastructures [Lazar97]. It has given rise to the IEEE P1520 initiative for Programmable
Network Interfaces [Bisw98]. The Mobiware approach has relied on the Xbind approach, modifying and extending it for cellular networks [Angin98]. Other related approaches have been the Tempest framework, targeting the creation of virtual partitions in ATM networks [Merwe98] and the Phoenix framework, targeting new network services on reprogrammable router processors [Putz00]. It should be finally mentioned that while there exists research work on network programmability, there has been no attempt to apply it to ad hoc networks in the manner proposed here.

5.2 Functionality and Architecture

Our programmable platform follows a lightweight approach to achieve programmability through the use of loadable plugins. The latter are blocks of code that can be uploaded and executed on the ad hoc network nodes i.e. terminodes, in order to perform specific configuration and monitoring operations. This can be an installation of a new routing protocol, extensions to an existing one or any other function that the MANET could benefit from. In order to decide on a particular plugin to be used for a given scenario, a plugin election should take place utilising the current contextual information. This involves advertisements from every mobile node that can provide suitable plugins for a given election. A predefined election algorithm should be used and identify a plugin to install globally. The latter should then be distributed throughout the MANET in a peer-to-peer fashion. Following the installation of the elected plugin, the mobile nodes should activate, i.e. execute, the plugin. In the following sub-sections we describe the functionality and architecture of the middleware platform, highlighting important design decisions and presenting the relevant reasoning.

5.3 Centralised vs. Distributed Management

A key consideration in an ad hoc network is the management approach to be deployed. In a centralised approach, the whole MANET is grouped into clusters, each electing a local leader or cluster head (CH). CHs then cooperate and elect a global leader or network head (NH). Key management decisions, such as triggering and coordinating the plugin election process, are taken by the NH. This hierarchical approach is similar to that of routing protocols, e.g. OSPF, and scales well, limiting interactions within a cluster or among cluster heads. It also allows operation in a controlled distributed fashion, when decisions are taken not only by the NH but through cooperation and “voting” among the CHs. A diametrically different approach is a fully distributed one, in which all the terminodes have “equal rights” and determine collectively any management decisions to be taken. This approach requires more complex cooperation protocols and may not scale for large networks with 100’s of nodes.
For our intended use of aligning dynamically the capabilities of terminodes through programmability, we have opted for the centralised approach, employing CHs and a NH. This approach was chosen due to its inherent desirable features outlined below: i) the leader or NH can impose a uniform management approach over the ad hoc network formed, ii) it is easy for a leader, at any point in time, to make a decision as to the selection of the most optimal protocol to be deployed relatively quickly; this holds both at bootstrap time or when the already deployed protocol/service becomes inappropriate for the current context, and iii) this clustering approach is more scalable for a larger network. The decentralised approach has the key drawback that the voting mechanism can be time-consuming. This is especially lengthy when in our scenario the already deployed protocol becomes inappropriate for the current context. The difficulty arises because there is no central node (NH) or few nodes (CHs) responsible for triggering the redeployment process. More importantly, the decentralised approach would lead to a dangerous situation where any node may trigger this process unnecessarily.

The CHs/NH election process is based on contextual information and can take into account location, capabilities such as processing power, memory, battery life, etc., expected residence time and possibly owner’s privileges. For example, the CH/NH needs to be a relatively central node in the cluster or network respectively while there is no point in having a cluster head with high probability to move radically away from its current location soon. There is a lot of work in the literature on cluster formation and cluster head election. The CH election heuristic can be similar to the one that is proposed in [Siva04a] for a longer-term large-scale MANETs or similar to the one proposed in [Siva04b] for more spontaneous MANETs. It should also be noted that a deputy cluster head or network head is also elected, so that there is immediate “functionality fallback” should a CH/NH leave.

5.4 Middleware Communication and Components

We have chosen to use the lightweight XML-RPC protocol, presented in chapter 3, as the basis of communication between terminodes running the programmable platform. XML-RPC can be considered as a subset of the SOAP protocol, unburdened from unnecessary complexity. Like all remote procedure call approaches, it allows software running on different operating systems and hardware architectures to communicate through remote procedure calls (RPCs). We chose an XML-based approach because we also use XML to represent contextual data in terminodes and this achieves easy integration. We could have possibly chosen Web Services/SOAP, but this approach would have certainly been more heavyweight. In addition, SOAP, in the same fashion with distributed object technologies such as CORBA, necessitate object advertisement and discovery functionality, which is not required in our platform that relies on simple message
passing modelled through RPCs. Given our recent performance evaluation of XML and other management approaches [Pavlou04a] and the findings in chapter 3, we believe that XML-RPC provides a useful blend of functionality and performance.

In addition, in order to make the platform even less demanding on processing power and portable to mid-range and small devices, we used the Java 2 Micro Edition (J2ME) virtual machine. The latter requires a much smaller memory footprint than the standard or enterprise edition, but at the same time it is optimised for the processing power and I/O capabilities of specific categories of devices. We used the Connected Device Configuration (CDC) framework instead of the limited (CLDC) one, as the latter lacks the required support of advanced operations.

The programmable platform can be divided into two middleware modules according to relevant functionality, as shown in Figure 5-1: the Cluster (or Network) Head and the Terminode modules. Depending on a node's current status, middleware functionality switches between Cluster/Network Head and Terminode mode respectively. The Cluster Head components deal with the management of the programmable Terminodes regarding programmable plugin election and distribution, as well as (re)configuration. The key components of the Cluster Head module are:

- **Plugin Election and Distribution**: it is responsible for the initiation and coordination of the election process. It implements the plugin election algorithm and is responsible for the initiation, distribution and activation of the elected plugins across the ad hoc network.

- **Plugin Configuration**: it is responsible for the (re-)configuration of installed plugins across the ad hoc network. It is also able to modify on-the-fly key parameters of active plugins, if the latter support this functionality.

The Terminode components deal with the management of the programmable plugins regarding plugin advertisement, installation, storage, distribution and configuration. The key components of the Terminode module are:

- **Plugin Management**: is responsible for advertising available plugins to the Plugin Election component, listening for requests from the latter for distributing a specific plugin, communicating with its peer nodes for the purpose of exchanging network plugins, as well as for installing, executing, reconfiguring and terminating the operation of a particular plugin.

- **Plugin Repository (PR)**: is responsible for storing and exposing available plugins to the Plugin Management component when required. The latter is able to extract and advertise to the Plugin Election component the characteristics of the stored plugins for the purpose
of plugin election. The Plugin Management component offers all the necessary operations for storing and deleting plugins, as well as searching for a plugin by name or type.

![Middleware platform architecture](image)

Figure 5-1: Middleware platform architecture

The components of the programmable platform can also be seen in a UML class diagram in Appendix A.

5.4.1 System Operation

In this subsection we describe the complete system operation, from election triggering to plugin activation, considering an ad hoc network that consists of a single cluster. According to the current context or human user command, the CH triggers the election process by contacting Plugin Election and providing the type of the plugin currently required by the MANET. The Plugin Election object contacts all the terminodes of that cluster in order to request advertisements of candidate plugins. Each member node performs a lookup in its plugin repository for one or more suitable plugins that can satisfy the requirements of the election process and replies to the CH Plugin Election with the characteristics of the retrieved plugins. Each terminode knows how to locate the CH, as its network address was made available to terminodes after the CH election. The CH Plugin Election executes the election algorithm and decides on the most suitable plugin, based on the current context, e.g. by assigning different weights to criteria such as CPU time, memory required, etc. Following the actual plugin election, the CH contacts the terminodes that already possess the elected plugin and instructs them to distribute it across the cluster. Plugin distribution is carried out in a peer-to-peer fashion, with each “owning” node flooding the plugin
to its neighbours, until the point that the plugin is fully distributed. Prior to any plugin exchange, each node probes its peers, using an XML-RPC message, in order to find out if they already have acquired the plugin and avoid a possible plugin retransmission. The probed terminode looks up its plugin repository and replies with a true or false statement in the case it owns the plugins or not respectively. For the actual transmission, the terminode currently distributing the plugin contacts the neighbour MN and passes all the characteristics of the elected plugin using an XML-RPC message. This is followed by the actual transfer of the plugin's execution code using a TFTP file transfer. When a new plugin has been successfully installed, the node sends a notification to the Plugin Election of the CH. At this point, the plugin is installed and available to be activated. The CH Plugin Election object, after receiving installation notifications from all the member nodes or after a predefined timeout period, it floods an activation message across the cluster to instruct the member nodes to execute the elected plugin. Each member MN should then perform a lookup in its Repository for the plugin, in order to obtain its reference in the local file system and consequently execute it in user space. A UML sequence diagram of the programmable platform operation, describing the plugin election and distribution process can be found in Appendix A.

5.4.2 Loadable plugins

A programmable plugin is a dynamically loadable object or module that can be installed, removed, activated and configured on-the-fly in order to extend the functionality of a node. A loadable plugin is a Java object that implements a generic interface with methods supporting common necessary functionality. Such programmable plugins can be loaded and activated dynamically into the operating system’s user space at run-time. They can be instantiated as often as required, while it is possible to have several instances of the same plugin with different configurations. Plugins execute only in user space, so a new or extended ad hoc routing protocol should operate in user space. This implies a performance limitation for plugins that implement network device functionality but it is too difficult and dangerous to achieve programmability at kernel level. The key reason behind the selection of Java for implementing the programmable plugins is platform independence. This is required in a diverse ad hoc environment, as is also the case in our ad hoc network TestBed, which consists of laptops and a PDA with different computing architectures. It would have been possible to cater for plugins in compiled languages, e.g. C/C++, but this would complicate the system and would require many versions available for each plugin, one for each node operating system / hardware architecture combination present.

Plugins expose two programmable interfaces for the purpose of configuration and monitoring. This first interface is used to configure and alter various aspects of the plugin functionality at runtime. On the other hand, through the monitoring interface plugins can provide some information regarding their status and possibly various useful statistics collected from their operation. For
Chapter 5. Programmable Middleware for Configuration and Monitoring of Ad Hoc Networks

Each programmable plugin there are several characteristics to be considered, such as the CPU cycles required for its operation, the run-time memory size required, and the plugin physical size which governs the distribution overhead. Each characteristic is assigned a unique and ordered identifier in order to be comparable with other candidate plugins during the election process. Due to the dynamic nature of plugins' installation and reconfiguration, a security mechanism for authorisation and certification would be required in order to ensure stability in an untrusted environment – this aspect is however outside the scope of our current work.

5.4.3 Plugin Election and Distribution

As already explained, an election procedure takes place among the member nodes. The coordinator of the election phase is the cluster head, whose functionality switches to Cluster Head mode. At this point, each terminode, whose functionality switches to Terminode, collects the characteristics of the plugins that fit the current requirement as advertised by the CH and sends them to the Cluster Head in a unicast fashion. The CH decides on the most suitable plugin for the current context by executing the election algorithm. The plugin election is based on a deterministic selection process, where each characteristic of a plugin has a comparable identifier. The election mechanism uses a simple cost algorithm based on weights assigned to selection criteria. Each criterion is assigned a mathematical weight based on its importance to the election process, while each plugin characteristic has a unique identifier, ordered from 0 to 10. The election algorithm used is given by equation shown in Figure 5-2. It is possible to adjust the criteria weights, based on the current context and on constraints the nodes might impose. The plugins characteristics that are considered during the election process are the CPU utilisation, memory size required and the plugin size.

\[
f(x) = \sum_{i=1}^{n} w_i \times A_i(x) \quad x \text{ is the specific plugin}
\]

where \( w_i \in [0,1] \), \( \sum_{i=1}^{n} w_i = 1 \) and \( A_i \in [0,10] \), \( \forall i \in [1,n] \)

**Figure 5-2: Plugin election cost function**

Following the plugin election process, it is required to distribute the elected plugin by requesting the nodes that possess it to distribute it to their immediate neighbours that don't have it. This technique minimises the network traffic and convergence time to achieve complete flooding of a particular plugin. The actual plugin transfer is achieved using the Trivial FTP (TFTP) protocol. Compared to FTP, TFTP is less complex and demanding on network resources. TFTP has no user authentication, which spares time and traffic in a trusted environment, but most importantly it
uses only one connection contrary to FTP that required two connections, one for control and one for data traffic.

A key aspect of plugin flooding is that a terminode should not receive the same plugin twice. This is deliberately prevented as the plugin size might be considerable and prove costly to the network. In order to prevent this from happening, the transmitting node will have to first probe its peer if it has already acquired the plugin and then only transmit it. The plugin distribution includes apart from the actual plugin transfer, the transmission of the plugin characteristics.

5.5 Platform Evaluation

In this section we present the evaluation of the programmable platform that was carried out, in both, a real ad hoc network and a simulator.

5.5.1 TestBed Experimentation

For the purpose of evaluating the programmable platform in a real ad hoc network, we deployed the former in our ad hoc TestBed. This consists of three laptops and one personal digital assistant, running the Linux Debian operating system. The ad hoc TestBed supports the 802.11b wireless standard for all the required communication between nodes. Packet routing is achieved using the AODV-UU, user space routing daemon by Uppsala University, which implements the Ad hoc On-Demand Distance Vector (AODV) routing protocol [AODV]. In order to create custom network topologies without having to place the nodes far from each other, we used a MAC address based filtering tool to simulate an “out of reach” situation. In this case, nodes discard incoming packets from predefined source MAC addresses, as for example AODV “hello” messages from specific nodes [AODV]. Several network topologies were used and test cases were carried out in order to validate and evaluate the ad hoc programmable platform. The scenarios implemented were different static network topologies and a case where a link between two nodes breaks and AODV has to construct an alternative route on-the-fly — this verified that the ad hoc routing protocol worked as expected.

The initial measurement taken was the response time and traffic generated by a single XML-RPC method call. The selected method was the “advertisePlugin”, which is invoked by the Cluster Head to the Terminodes for requesting advertisements for candidate plugins. The selected method call carries as argument the “advertisePlugin” string and has no return type. The traffic generated for that method was measured to be 1,211 bytes and the response time was 6.5 milliseconds when the call was made between two laptops and 23 ms when it was made between a laptop and the PDA. As expected, the PDA exhibits a much slower processing time and therefore the parsing of
XML messages needs considerable more time than on the laptops. The latter appear to be around 3.5 times faster.

The next set of measurements that were recorded, involved the complete system run for different topologies of the ad hoc network. For the first test case, the topology shown in Figure 5-3 was realised. The CH in this case is manually selected to be node “ares”, as it is powerful and is a single hop away from every other node. During the plugin advertisement procedure, all four nodes will have a number of plugins to advertise but the plugin with the highest identifier is deliberately selected to be the one that belongs to node “zeus”. The elected plugin is a simple echo server and its size is 1,450 bytes. For a complete system execution, as described in 5.4.1, the measurements taken were the overall time and the traffic generated by AODV, the interaction between Terminodes and the Cluster head modules and the transfer of the elected plugin from the owner to all other member nodes within the cluster. The time taken for a complete system operation within the considered cluster, from the request for plugin election to the activation of the elected plugin, was measured to be 2.56 secs over a number of samples. If “ares” had the plugin, which is a more central node, the convergence time is 2.40 secs. During this period, the overall AODV traffic observed across the whole network was 528 bytes, which mainly includes “hello” messages. The traffic when the Cluster Head module is involved includes the request for plugin election (1,084 bytes), which initiates the system, the advertisements of candidate plugins by the Terminodes module (3,475 bytes), the request to the elected plugin owner to distribute the plugin (1,095 bytes) and the plugin installation confirmation by the Terminode modules (2,096 bytes). Overall, the Cluster Head related traffic was recorded to be 7,750 bytes. On the other hand, there is the Terminode related traffic, which includes the communication between member nodes only. This was recorded to be 15,114 bytes and includes the flooding messages for requesting the plugin advertisement (3,633 bytes), the method for querying an adjacent Terminode module if it has the plugin under distribution (4,392 bytes), the distribution of the elected plugin and its characteristics (3,429 bytes) and the flooding messages for requesting the plugin activation (3,660 bytes). Finally, there is the plugin distribution traffic, caused by the TFTP file transfers. In the specific test case, three plugin transfers are required and the overall traffic generated was measured to be 5,253 bytes.

Figure 5-3: TestBed configuration - Test case 1
For the second test case, the requirement was to examine the ability of the programmable platform to follow the sudden changes in the topology of the ad hoc network. The original topology of this scenario was the same as the previous case, but with the difference this time that during the plugin election process, the link between nodes “ares” and “Poseidon” was deliberately broken to form a “chain-like” topology. That would mean that all packets from “ares” to “Poseidon” and vice versa would have to be routed via “apollo”. It was found that this function is well supported by the AODV-UU daemon and was indeed effortless. The same holds for the programmable platform, which successfully completed its operation regardless of the sudden topology change. The time taken for a complete system run in this case is 2.71 sec, which is 150 ms longer than the first case. This was expected as AODV had to discover a new route and any packets going from “Poseidon” to the CH and vice versa would have to do an extra hop. Due to the fact that the AODV daemon had to construct a new route, the AODV traffic was also increased to 576 bytes. On the other hand, the Cluster Head related traffic remained the same (7,750 bytes), as it is mainly dependent on the number of nodes in the network and the ones owning plugins. The Terminode related traffic however, which is heavily depended on the network topology and the number of neighbours of every node, was reduced to 14,016 bytes. This is because the traffic caused by the message flooding in this chain-like topology, had one strict route. Finally, the plugin distribution traffic remained the same (5,253 bytes), as again there is one plugin owner (zeus).

Key results to note from the experiments are the following. First, the overall amount of control traffic required is in the order of few kbytes per node and it is actually more for the terminodes than for the Cluster Head. In the next section we validate through simulation that the control traffic per node does not increase with the number of nodes, which guarantees scalability. In addition, the transmission traffic required for a 1.45 Kbytes plugin is 3.43 Kbytes because its characteristics are also flooded to the adjacent node through an RPC. The plugin size has obviously no impact on the control traffic, but for a 64 Kbytes plugin, which is a typical size of applications in Java Micro Edition, the transmission traffic becomes 66 Kbytes.

Another important result is that for a network of 4 nodes, the overall convergence time is 2.56 secs with two hops and 2.71 secs with three hops of uploading a 1.45 Kbytes plugin. Increasing the plugin size to 64 Kbytes brings those times close to 3.5 and 4 secs respectively, mainly because of the additional transmission time. This means that the average latency per node is approximately 1 sec for a 64 Kbytes plugin in a 4 node network. In the next section we validate through simulation that this value increases almost linearly with the number of nodes, which again guarantees controlled convergence times. For example, for a network of 20 nodes the average latency is 2 secs per node, which means 40 secs overall convergence time (Figure 5-5).
5.5.2 Simulation

While TestBed experimentation provided an initial idea about the relative performance, traffic overhead and convergence time of our system, as well as values for key elemental interactions, simulation allowed us to experiment with much larger node populations and topologies and assess in much more detail the performance of our approach. The simulation works attempts to investigate the performance of our plugin election process in terms of the average control cost incurred per node and the latency involved for the complete plugin deployment process. We performed our simulations using the GloMoSim simulation package [GMS] in which we implemented the associativity-based CH election heuristic of [Siva04b] - note that in the TestBed experiments we hard-assigned the CH role - and the already described plugin election process. The transmission range of each node is set to 100 m, and the link capacity takes a value of 2 Mbps. The simulations were performed for a stationary MANET and the simulation parameters are similar to those of [Siva04b], with key values used as measured in the TestBed experiments. The scalability of our plugin election scheme is assessed in terms of i) increasing node-count, ii) increasing average node-density. In the first-set of simulations, the scalability of the clustering protocols is measured in terms of increasing node-count. In order to assess the effect of increasing network size on the clustering and plugin election schemes, the terrain-area is also increased with an increase in the number of nodes, so that the average node-density is kept constant in the first set of simulations. The number of nodes in this case is varied from 25, 100, 225, 400 and 625. The terrain-area size is varied so that the average node degree remains the same and accordingly 200X200 m², 400X400 m², 600X600 m², 800X800 m² and 1000X1000 m² are selected for each scenario. Figure 5-4 shows the average control cost incurred per node during the clustering as well as plugin election processes as a function of increasing number of nodes. As can be inferred from Figure 5-4, the average control cost per node does not depend on the increasing node count, and hence both the clustering and plugin election schemes are scalable.

![Graph](image)

Figure 5-4: Average control cost incurred per node as a function of increasing node count.
Figure 5-5: Average latency as a function of increasing node count

Figure 5-5 depicts the average latency involved per node for a complete plugin deployment process as a function of increasing nodes. The latency involved is actually the time the plugin process takes from the point when a terminode receives the plugin election trigger message from the CH until it finally receives the plugin activation message. It can be seen from Figure 5-5 that the latency increases almost linearly with the node count. This is somehow expected as we assume that the number of plugin "owner" nodes also increases proportionally with network size. Although these nodes are randomly distributed, they appear to be reasonably well distributed as the network grows in size, avoiding "empty" areas, hence the almost linear latency increase.

In the second-set of simulations, we measure the scalability in terms of increasing node-density. In this case, the terrain-area is kept constant at 1000X1000 m^2, while the number of nodes in the given area is increased. From Figure 5-6, it can be seen that the average control cost incurred per node for both schemes decreases with increasing node density initially up to a point, and beyond that it stays constant. This is due to the fact that when the node-degree is low, there is less connectivity among all the nodes and as a result we have more than one unconnected ad hoc networks, each with its own CH. More than one CH within a given region means increased amount of unnecessary clustering and hence plugin related cost. On the other hand, when the network becomes denser there is increased connectivity, resulting in a uniform network with a single CH. This results in decreasing control cost with increasing node-density. Another aspect is that in a sparse network, plugin "owner" nodes are also sparsely distributed, resulting in more control cost to share the plugins, as it is evident from Figure 5-6 where control cost per node appears to be high for a few node scenario. On the other hand, when the node density increases, the likelihood for many nodes to have a common-plugin as their optimal plugin is high and the burden to distribute it to those that do not have it is equally distributed.
5.6 Conclusions

In this chapter we presented a programmable middleware platform that can align the capabilities of the nodes of an ad hoc network through the use of loadable plugins. This is crucial in a heterogeneous environment if a common communication infrastructure is to be deployed that could achieve, for instance, quality of service based communication across the ad hoc network. For the platform communication we used the lightweight XML-RPC message-oriented protocol, where relevant management information is encoded in XML and transferred over HTTP. The platform was implemented in the Java 2 Micro Edition programming language, which can cater for small to medium devices, such as PDAs.

Our initial performance evaluation seems encouraging, with a few seconds required for convergence in a small network and linear increase with node count. In addition, high node density does not seem to have adverse effects until a threshold, above which it results in increased 802.11 collisions and performance deterioration, which is expected. Given the fact that we have adopted Java-based plugins for platform independence and XML-RPC for communication due to the easy integration with XML-formatted data, the overall performance seems encouraging.

We have adopted a centralised management approach, with cluster heads administering geographical clusters and one of them nominated as network head, administering the whole network. The approach could be centralised, with the network head taking all decisions, or partly distributed, with management decisions reached through collaboration among cluster heads. For the time being, the platform focuses in a single cluster only, with the cluster head being also the network head. Cluster-to-cluster interaction will be investigated in the future.
Furthermore, given the peer-to-peer nature of the platform, it is valid to state that the whole ad hoc network could be in risk if a loadable plugin was an engineered computer virus. In this case, a secure mechanism for verifying the advertised plugins would be a requirement in untrusted networks. Last but not least, the plugin election and distribution process could benefit from additional contextual information, gathered from the terminodes and their behaviour in the mobile ad hoc network.

The work presented in sub-section "5.5.2 Simulation" was undertaken as a collaboration between S. Sivavakkeesar and the author. The latter provided key input in the specification of the simulations.
Chapter 6

6 Summary and Conclusions

The summary provided in this final chapter aims to present an overall picture of the thesis with respect to the issues addressed. A discussion of thesis contributions highlights the approach followed, the work performed and the extent reached towards the thesis objectives. This is followed by conclusions section, presenting the main findings and results. Finally, we discuss potential future research directions identified through this work.

6.1 Thesis Summary

The thesis investigated the use of middleware for managing emerging network architectures. A comparison of four middleware technologies was initially carried out in terms of performance and usability. The metrics of the evaluation were the response time, bandwidth overheads, memory usage and the usability of the selected platform of each technology. The metric of the latter was the author's experiences from the development and the lines of code written. For the further investigation of middleware technologies for emerging network architectures, two case studies were carried out in the context of this thesis. The first case study investigated a middleware system, based entirely on mobile agents, for configuring and monitoring Quality of Service in IP DiffServ capable networks. The system was evaluated in a heterogeneous TestBed, containing both Linux and Cisco network elements. The second case study investigated an XML-RPC based infrastructure for allowing the seamless programmability of mobile ad hoc nodes. The latter can vote, retrieve and activate required protocol and service software dynamically for the purpose of aligning the nodes' capabilities across the MANET. The thesis contributions are presented in more detail in the following section.

6.2 Thesis Contributions

6.2.1 Comparison of Middleware Technologies

In the context of this work, we carried out an extensive comparison of selected middleware technologies in terms of capabilities, performance and usability. We recorded the overheads of a common operation with respect to response time, bandwidth utilisation and memory usage. It was
shown that SOAP and XML-RPC are indeed promising technologies but, being XML-based, have more bandwidth overheads than Mobile Agents and CORBA. On the other hand, being XML-based is also their biggest attraction, due to potential easy integration with other applications. However, Mobile Agents exhibits significantly higher performance costs in terms of response time and memory usage than the other middleware approaches in our comparison. In the case where mobile agents need to traverse a resource constrained environment, special care needs to be taken as the cost of migrating, presented in 3.4.2.5, is expensive in terms of both bandwidth and latency.

An important finding is that XML-RPC appears to exhibit relatively good performance in terms of all the evaluation aspects and especially in terms of memory usage, where it outclasses all other platforms. This justifies the argument of being lightweight, which means that it can be suitable for resource-constrained environments, such as devices in ad hoc networks. It was also discussed that XML-based approaches are more flexible and user friendly than CORBA in terms of software development, since their usability is similar due to the stub-based approach most platforms follow.

6.2.2 Assessment of Mobile Agents within a QoS Configuration and Performance Management System

For this case study we developed a configuration and monitoring management system for a QoS-enabled network, based on mobile agents. The objective was to assess the suitability of mobile agents in such a scenario and. It was shown that using mobile agents as the basic design and implementation technology for our system helped to decentralise configuration and monitoring tasks and promote good software design and relatively easy implementation of such a complex system. However, it was clear that mobile agent platforms are rather immature and not yet streamlined for efficiency. Despite that, the savings for complex configuration and monitoring tasks are significant in comparison to client/server protocol-based or distributed object technologies.

6.2.3 Proposal of XML-Based Programmable Middleware for Ad Hoc networks

In this case study we proposed an XML-RPC based approach for achieving programmability in ad hoc networks. The proposed platform makes use of loadable plugins for aligning the capabilities of ad hoc. Our performance evaluation seems encouraging, with a few seconds required for convergence in a small network and linear increase with node count. In addition, it was shown that high node density does not seem to have adverse effects until a threshold, above which it
results in increased 802.11 collisions and performance deterioration, which is expected. Given the fact that we have adopted Java-based plugins for platform independence and XML-RPC for communication due to the easy integration with XML-formatted data, the overall performance of our approach is encouraging.

### 6.3 Conclusions

Mobile Agents technologies (MAT) offer a relatively new software design and development paradigm for distributed systems. The use of mobile intelligent agent technologies in service creation, management and deployment is enhancing traditional approaches, making services programmable and dynamically customisable by end-users. Service control and management frameworks such as IN and TMN allow end-user access to management services. These services, however, are fixed in the sense that new features can only be added after a lengthy research-standardisation-deployment cycle. In the context of communication networks, mobile agents enable the transformation of current networks into remotely programmable platforms. The concept of “Remote Programming” using mobile agents is considered as an alternative to the traditional “Client/Server programming” based on the Remote Procedure Call (RPC) or the static distributed object paradigm (e.g. CORBA, Java-RMI). In contrast to “shouting” requests across the network from a client to a server, mobile agents transport themselves to the remote (server) computer, where action has to be taken.

Compared to static approaches, mobile agents can provide two major advantages:

- **Tactical advantage**: Improved performance, where client software can migrate and operate locally with the server software instead of communicating across the network. The performance advantage of remote programming through mobile agents depends partly upon the network: the lower the throughput or availability, or the higher the latency or cost, the greater the advantage.

- **Strategic advantage**: The strategic advantage of remote programming is customisation. Agents provide flexibility in extending the functionality offered by existing communication systems.

This means that mobile agents enable control tasks to be performed in a real distributed manner. In particular this concept enables telecommunications services to be provided instantly and to be customised directly at the locations where the intelligence is needed. An additional boon is that access to local information at a particular network node does not suffer from varying delays, as it is the case with remote access. As an example, the collected information from a monitored object can be more accurate while fine-grain intervals may be used in cases of periodic polling. Finally,
a key advantage of mobile agents is the provision of dynamic services in network elements that have not been pre-programmed with such facilities. The customisation of mobile agent behaviour can provide a powerful mechanism for “intelligence on demand”. In other words, clients are allowed to “push” functionality to a point offering elementary hooks, which can be accessed to provide derived, higher-level services.

On the other hand, there is a performance overhead to pay when using mobile agents. Remote method invocations can be four times slower than those in Java-RMI / CORBA and this difference could be more pronounced when comparing performance to the protocol-based CMIP and SNMP approaches. In addition, agent migration incurs a substantial overhead in terms of both latency and required data to be transported across the network.

Web services and SOAP can be seen as a distributed object technology; in fact, platform providers have been taking a CORBA-like approach with stub objects, which reinforces this view. Its use for network and systems management presents the same problems as CORBA, so exactly the same solutions can be adopted. Its usability is similar to CORBA due to the stub-based APIs and arguably better than SNMP. On the other hand, there is no security and notification support at present, which means this technology is not yet ready to be used for network management. The initial performance evaluation is encouraging but also highlights some expected problems. Information retrieval times are approximately twice those of CORBA, but the key problem is the amount of management traffic incurred due to the XML-based encodings, which can be up to six times that of CORBA. This can be reduced through compression at the expense of slower retrieval times.

XML-RPC, which can be regarded as a subset of SOAP, appears to be twice as efficient than the latter and in some cases very close to CORBA. The major advantage of the Apache/XML-RPC platform that was observed during the measurements is its very low memory usage compared to the other platforms. For this reason, it was considered suitable for the case study presented in chapter 5. The initial performance evaluation of the second case study was encouraging, with a few seconds required for convergence in a small ad hoc network TestBed and linear increase with node count. In addition, high node density did not seem to have adverse effects until a threshold, above which it results in increased 802.11 collisions and performance deterioration, which is expected.

In summary, SOAP and XML-RPC are promising technologies but, being XML-based, have more overhead than Mobile Agents and CORBA. On the other hand, being XML-based is also their biggest attraction, due to potential easy integration with other applications. Mobile Agents is probably the richest middleware technology in term of capabilities, but is let down by its significant resource requirements. The choice of the middleware technology to be used in a
particular application is not so straightforward and should be treated individually per application. Mobile agents offer the capability of code migration but are mainly suitable for environments with plenty of spare resources. On the other hand, XML-RPC offers very limited capabilities, but is lightweight and would be appropriate in an infrastructure-less, resource constrained environment. CORBA and SOAP follow the same stub-based approach but differ greatly. CORBA has many more features than SOAP as it is a "grown-up" technology, gradually extended and refined through the years. However, SOAP embraces attractive features, such as the XML representation of data and HTTP as the transport protocol. In addition, SOAP was found to be more user-friendly for the application developer. We believe that XML-based technologies still need a lot of work on various aspects, such as security and notifications, but will eventually become leaders of the middleware world.

6.4 Future Directions

The work on middleware technologies presented in this thesis opens several new possibilities for research. We focus on additional work that would enhance and complement the thesis.

6.4.1 Compression in XML-based technologies

XML's verbosity, which is due to its textual nature, imposes significant overhead on relevant technologies. During the communication of middleware components, a significant amount of traffic is incurred by the exchange of XML messages. A way of reducing the size of XML documents is by compressing them. However, this might lead to slower response times and computational resources requirements as every message would have to be compressed or decompressed every time it is being transmitted or received respectively. In addition, the use of compression mechanisms reduces the interoperability between applications as both sides need to be aware of the compression being used. XML compression however seems attractive in a bandwidth-restricted environment and more evaluation is necessary to balance potential traffic advantages against increased processing requirements.

6.4.2 Exploitation of Context in Middleware Applications

Middleware-based applications can collect and utilise contextual information from the network in order to adapt to a particular environment. For the second case study, presented in chapter 5, the plugin election and distribution process could benefit from additional contextual information gathered from the mobile nodes and their behaviour in the MANET. Context information, such as daily movement patterns of mobile nodes, electronic diaries, past history, etc., could be collected in order to assist in the election of cluster heads.
6.4.3 Distributed Management Approach in Ad Hoc networks

For the case study, presented in chapter 5, we adopted a centralised management approach, with cluster heads administering geographical clusters and one of them nominated as network head, administering the whole network. The approach could be centralised, with the network head taking all decisions, or partly distributed, with management decisions reached through collaboration among cluster heads. For the time being, the platform focuses in a single cluster only, with the cluster head being also the network head. Protocols and algorithms for peer-to-peer decision making among cluster heads need to be investigated and the approach to be validated in a larger scale TestBed.
Bibliography


IEEE Communications Magazine, June 2001


Grasshopper Mobile Agent Platform, IKV++ Technologies, http://www.grasshopper.de/


Management of Telecommunication Systems and Services: Modelling and Implementing TMN-based Multi-domain Management, Hall, J. (Ed), Lecture Notes in Computer Science 1116, Springer-Verlag, 1996


Overview of TMN Recommendations, ITU-T Recommendation M.3000, 1995


Information Agents (CIA '00), 2000


[Pavlou95] Issues in the integration of IN and TMN, Pavlou, G., Griffin, D., Bringing Telecommunication Services to the People, Proceedings of the 3rd International conference on Intelligence in Broadband Services and Networks, Springer-Verlag, 1995


[UDDI] Universal Description, Discovery and Integration (UDDI), http://www.uddi.org/


[WSDL] Web Service Definition Language (WSDL), http://www.w3.org/TR/wsdl

[X700] Management framework for Open Systems Interconnection (OSI) for CCITT applications, CCITT Recommendation X.700, 1992


Appendix A – UML Diagrams

Chapter 4 UML Diagrams

"Create VPrP" Sequence Diagram

This UML sequence diagram presents the complete operation sequence of the QoS configuration system for configuring a new connection.
This UML sequence diagram presents the operation sequence of the performance monitoring system for the point where the monitoring is initiated to the point where the first monitoring statistics are made available.
Chapter 5 UML Diagrams

"Election Process" Sequence Diagram

This UML sequence diagram presents the operation sequence of the programmable platform for the point where the Cluster Head receives a message for starting the election process to the point where the plugin is activated.
This UML class diagram contains the objects of the programmable platform along with their operations. The "ClusterHead", "PluginElecton" and "P1uginConfiguration" objects are part of the Cluster Head package and the remaining objects part of the Terminode package.