Computing Department
Lancaster University

# Service creation combining programmable networks and open signalling technologies

Mark Richard Banfield
B.Sc.

Submitted for the degree of
Doctor of Philosophy
March 2001

# Abstract

The deregulation of the telecommunications sector, and the resulting multi-operator competitive marketplace, has reduced the potential profitability of simple voice traffic. To return to the high dividend levels to which shareholders have become accustomed, operators are forced to turn to other sources of income. Future profitability is likely to come from content delivery and associated value added services, the mergers between various Internet Service Providers (ISPs) and content-creators being championed as the beginning of a new trend within these industries.

Faced with a competitive market, network operators are increasingly offering value-added services to differentiate themselves from the competition. At present this is taking the form of concentrating on specialist markets, for example the provision of Virtual Private Networks (VPNs) or multipeer services.

This thesis explores the technology necessary to program the network to provide specialist network layer services to meet distinct customer requirements. This begins with a critical examination of existing service provisioning approaches from both the "open-signalling" and "active-networking" domains.

The Alpine framework, A Lightweight Programmable Internet Environment, is presented and discussed. This is a model for providing network programmability to third parties, combining programmable and active network technology. Current emerging industry trends and standardisation are drawn upon in the framework. The Alpine framework is compatible with the programmable interface standardisation activity in the IEEE P.1520.3 IP sub-working group. The resulting implementation of the "Active Router" component can thus be viewed as an early interpretation of IEEE P.1520 within a novel active networking context.

Finally, an evaluation of the architecture is presented, based around an analysis of the mechanisms for service deployment. Central to the architecture has been the adoption of distributed middleware technologies, the performance of which is measured and discussed with respect to applicability in the active networking domain. The architecture and

router design fundamentals are qualified through comparison with a state-of-the-art family of commercial routers.

# Acknowledgements

I would like to express particular thanks to my colleague Steven Simpson for his support and constant attempts to understand my ideas and ground them in common sense. Thanks go to Chris Edwards, Paul Smith, Tom Fitzpatrick, and Laurent Mathy for their support, encouragement, and involvement in technical discussions. I'm also very grateful to my Ph.D. supervisor, Professor David Hutchison, for his encouragement and guidance throughout my studies.

Special thanks are also due to the many colleagues from industry that I have been lucky enough to work with as a Research Assistant at Lancaster University. Particular acknowledgement should be given to Ian Marshall from BT for allowing me to combine personal research with my activities on the EURESCOM P926 Caspian project. Finally, thanks to Jan Laarhuis from KPN Research, Huw Oliver from HP, Nicolas Mercouroff and Nathalie Charton from Alcatel for the many discussions that have over the years challenged and helped form the ideas presented in this thesis.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Electronic communications has had a huge impact on post-industrial society during the twentieth century. From the earliest telegraph networks through to universal provision of telephony, television and radio, these technological advances have led to what is now termed the "global village". Crucially, the communications infrastructure is the essential enabling factor in construction of a single world marketplace for goods and services.

Against this background, the telecommunications industry has fast been changing, from state-owned monopolistic PTTs (Post, Telegraphy, and Telephony provider) to a competitive deregulated multiservice market. Along with this change of ownership, the nature of traffic carried has changed; for example, today British Telecom carries more data than voice traffic over its core networks.

However, with increasing competition, the profit raised from carrying simple voice/data is under pressure, and network operators may turn to other revenue raising services. Within the PSTN (Public Switched Telephone Network) and GSM (Groupe Spéciale Mobile) mobile networks within the UK, operators are increasingly offering value-added services to differentiate themselves from the competition. Examples of these services include call waiting, "voicemail", text messaging, and traffic report services. The challenge considered by this thesis is to introduce a mechanism into IP networks, to enable Internet Service Providers (ISPs) to offer similar value-added services to raise revenue, and differentiate their services from the competition to attract/retain customers.

## 1.1 Intelligence in the network versus the end terminal

The intelligence for provisioning a value-added service may be located either within the network or at the end-terminal device. However, at present there is no consensus as to the proper location. The logic for the "call waiting" service resides in the IN (Intelligent

Network) fabric of the PSTN (Public Switch Telephone Network), while pre-programmed fast-dial numbers are held at the terminal.

Indeed, the massive advances in the computational power of PC based terminal devices makes the end-terminal an attractive and scaleable place to locate value-added communication service logic. This, coupled with the low cost of service creation and deployment in the open PC software marketplace, has made the end-terminal PC Web browser a preferred location for introducing services in the Internet.

By contrast, data-communications networks have not incorporated value-added service logic. The trend has been towards simplification of network elements to achieve higher throughput. This trend is best examined with the rejection of inter-node error-correction with the migration from X.25 to ATM (Asynchronous Transfer Mode), and the recent acceptance of connection-less IP over the heavyweight connection set-up of connection-oriented protocols.

This background could draw into question the appropriateness of the active and programmable technologies discussed in this thesis. The market for introducing new services into the network elements has remained closed, the vendors unwilling to allow third parties access to their proprietary APIs (Application Programming Interfaces).

In recent years this scenario has been changing. Equipment vendors under pressure from their biggest customers (the network operators) and international standardisation have begun to provide limited third party control. The IETF (Internet Engineering Task Force) GSMP working group is charged with standardisation of the General Switch Management Protocol (GSMP) (Doria *et al.*, 2000), a protocol designed for high-performance third party connection management. The IEEE standardisation project P.1520 (Biswas & et al, 1998), a group dedicated to the standardisation of programmable interfaces for network elements, has the support of major router vendors (Cisco and Nortel Networks).

Although the computational power of the desktop PC has risen considerably, it is predicted that many future terminals will be characterised as low-power consumption specialised mobile devices. The end-terminal is therefore a less attractive location for value-added services since the service must operate on devices of low computational power and memory, and heterogeneous hardware, operating systems, and APIs. Second, locating value-added intelligence at the end-terminal might not provide optimal resource utilisation, as it is difficult for the distributed instantiations of the service to co-operate.

A suitable example of this is the Video Cassette Recorder (VCR). Most homes have a VCR installed, and at any one occasion there will be multiple VCRs around the UK making recordings of the same programme. This is obviously inefficient in terms of storage capacity, and involves most homes in the UK purchasing an expensive VCR approximately every five years. An alternative solution (van der Merwe *et al.*, 2000) based on CATV (Cable TV) is to provide programme recording as a value-added service. Only one copy of the programme need be stored, and the end-user will not find it necessary to upgrade VCRs to keep abreast of advances in technology.

## 1.2 Thesis Aims

The main aim of this thesis is to propose a mechanism for introduction and provision of communications oriented value-added services. This is considered from the viewpoint of an Internet Service Provider or network operator, which, due to an increasingly competitive market for provision of IP connectivity, must increasingly look to differentiate itself from the competition. The solution presented in this thesis is to introduce customer-oriented services, following a trend already taking place with ISPs tailoring themselves to family and business oriented connection and content provision.

The main aims of this thesis are as follows:

- *Customer-oriented control and management*

  Traditional network management services are grouped under the five FCAPS management functions (*f*ault, *c*onfiguration, *a*ccounting, *p*erformance, and *s*ecurity), each distinct from call-control functionality (route selection, bandwidth management). The logic behind this division was the distinct real-time requirements posed by call-control compared to the weaker time constraints inherent on management functions. However, it is now generally accepted that this impacts upon the potential for service provision by hampering the cross-functionality properties intrinsic to many novel services.

  Instead, the solution offered in this thesis suggests that novel services can be created by offering interfaces to the customer without regard to the control and management functionality they exploit. No longer do separate protocols enforce this separation, as seen today with the use of SNMP for management, and a separate suite of protocols (e.g. RSVP, SIP) for call-control.

- *Third-party element programmability*

  The introduction of open PC hardware and software architecture has enabled the existence of a buoyant IT industry consisting of many small companies specialising in low volume niche market PC software. In contrast, network equipment remains a largely closed environment, the industry being dominated by a small number of vendors (chiefly Alcatel, Cisco, Ericsson, Lucent Technolgies, Nortel Networks, and Siemens). These vendors introduce new services into their equipment only when they can benefit from large economies of scale. This thesis adopts the view of the OPENSIG community (Opensig, 2000), that by opening the switching system software to third party programmability, operators and software houses will be able to develop a richer set of network features designed to satisfy an individual ISP's requirements.

- *Fusion of programmable and active networking technologies*

  The domain of programmable element research can be considered as two distinct technologies. Firstly, the "programmable network" approach typified by Xbind (Chan *et al.*, 1996) and the XRM (Lazar *et al.*, 1994), and championed by the OPENSIG community, defines programmability at the element, network and service domain. At the heart of the approach is the broadband kernel and the BIB (Binding Interface Base), a collection of interfaces enabling component intelligence residing off-board the network element to administer/control/manage the network.

  The second approach, "active networking" has emerged from notion of agent-based computing. At its most extreme is the capsule (Tennenhouse *et al.*, 1997) approach, whereby every packet has the potential to contain as payload both data and active code. Unlike the "programmable" approach where intelligence resides on a secondary Operations Support System (OSS) platform, in this case active code is executed directly at the network element itself.

  An important goal of this thesis is to fuse together these diverse approaches thereby adopting the characteristics of both. The solution exploits the high-performance and flexibility of executing code at the network element, combining the secure network centric and device independent properties of "programmable networks".

- *Utilisation of programmable interfaces* The solution to third party programmability adopted in this thesis is through the construction of programmable interfaces. These are assembled into a hierarchy of interfaces, providing abstractions appropriate to the user/programmer viewpoint. This solution differs from that commonly adopted by active networking approaches. Conventionally the notion of programmable interface is not distinguished from that of the Execution Environment (EE), and thereby tied to programming language rather than support for hierarchical layered programmable abstractions.

- *Compatibility with emerging industry trends and standardisation* As well as incorporating the latest research directions in service programming, the framework presented is compatible with the emerging standardisation within this domain, that of IEEE P.1520 (section 3.4). The framework should be considered as an interpretation of the P.1520 reference model in the context of IP networking, the results of which have been presented to the P.1520.3 working group (Banfield *et al.*, 2000a) and published to the community (Banfield *et al.*, 2000b).

- *Significance of middleware technologies for support of active services* It may be argued (section 7.3.3) that the active node architecture (Node OS, Execution Environment (EE), and Active Application (AA)) replaces the role of conventional distributed middleware technologies (e.g. CORBA, DCOM, RMI) in service creation. However, useful services may be constructed composed of many individual co-operating active component elements. The potential dislocated and heterogeneous properties of these active components mandates access and location transparency through the provision of distributed middleware.

- *Contribute full service provisioning context to active network technology research* Active network services, unlike their close relative "agent based computing" (Breugst *et al.*, 1998), lack support for full service and component lifecycles. To satisfy the requirements posed by a multi-component service environment, a lifecycle is defined that consists of states and transformations through which an active service must progress.

## 1.3 Thesis Structure

The structure of the thesis is as follows. Chapter 2 begins by considering the impact that three major trends, deregulation, transport technologies, and modern software engineering approaches, have had on the telecommunications and information technology industries. Whilst the former trends may be classified as technological and regulatory push, the resulting convergence of the telecommunications, information technology, broadcasting and entertainment industries have resulted in 'requirements pull' arising from the demand for multimedia applications and services. Next (section 2.1) a definition of types of multimedia applications and services of the future is considered.

The second section of Chapter 2 takes an historical consideration of recent technologies within the telecommunications industry with regard to provision of advanced services. Three technologies of significance are highlighted, viz. the Intelligent Network, Broadband-ISDN, and the Telecommunications Information Networking Architecture (TINA). Of these three, the most ambitious, TINA, is considered in greater detail, providing an overview of the enterprise, computing, and network resource architectures and mentioning its apparent failings and limited impact on the industry. Finally, the emergence of the Internet as an alternative platform for communication services is introduced along with the challenges ahead with regard to future services including multicast and Quality of Service (QoS).

Chapter 3 considers the state-of-the-art in distributed programmable networks as a mechanism for novel service deployment. The goal here is to introduce the approaches that have both influenced and provided alternatives to the programmable service creation environment discussed in this thesis. These approaches are classified into two categories, namely: examples of *programmable* and *active* networks. First, the approaches in respect to *programmable* networks, that is techniques adopted from both TINA-C and the OPENSIG workshop are considered. These range from ReTINA an extension of the TINA NRA, through to the XRM broadband kernel and finally directions towards standardisation including IEEE P.1520, and Parlay.

In section 3.6, the second "*active networks*" (AN) approach is examined. This technology is far less mature than the preceding approach, and has not yet been considered for standardisation. To aid understanding of the technologies used subsequently in this thesis, especially in regard to the common misunderstanding with regard to what constitutes an active network, definitions are given for the terms *integrated* and *discrete* ANs. Many early platforms (e.g. ANTS and Switchware) follow the *integrated* approach where an in-band capsule-based technique to code transport is applied. Many of the more recent platforms (e.g ALAN) including the framework discussed in this thesis, take the alternative out-of-band code distribution technique inherent in the *discrete* approach. Finally, several examples of *discrete* active networking are introduced.

Chapter 4 introduces Alpine, **A** **L**ightweight **P**rogrammable **I**nter**N**et **E**nvironment, and considers the motivation and requirements behind the framework proposed in this thesis. It adopts a number of well-established practices and toolsets, RM-ODP, OMG IDL, UML, and IEEE P.1520 to aid presentation of the fundamentals of the framework. In particular Alpine is described from 4 viewpoints, namely enterprise (business requirements), element programmability, deployment, and information representation.

This leads naturally in Chapter 5 to the definition of the active DPE architecture necessary to meet the requirements of the Alpine framework. Initially, discussion centres on the role of signalling in the management of a modern network. Once this has been clearly defined the architecture of the Alpine Middleware Inter-networking Environment (AMIE) is described. The architecture provides a novel interpretation of the IEEE P.1520 reference model in the context of IP based service deployment. An extension to the reference model is suggested, incorporating a new vertical plane, the Signalling Transport Service Provider (STSP) and corresponding 'S' interface reference point.

Chapter 6 describes the partial realisation of the framework previously defined in Chapters 4 and 5. The emphasis is on the programmable element component of the framework, with the implementation of an example Active Router node. A description of the underlying technologies (ALAN and LARA) is provided. Next, the implemented components, corresponding interfaces, and interaction model are summarised. Given the heterogeneous programmable requirements and node features/architectures, it is asserted that a single extensible programmable abstraction is difficult (perhaps impossible) to define. The solution favoured to assure compatibility between active applications (AA) and heterogeneous active nodes is to adopt an XML based meta-language for requirements specification. The Component Compatibility Mark-up Language (CCML), an XML based meta-language, is parsed pre-execution at service instantiation time. Based on local knowledge, the active node is best able to detect requirements mismatch and fetch the appropriate AA implementation.

Chapter 7 presents an evaluation of the proposed Alpine framework. Since the implementation concentrates on the Active Router component, two approaches, quantitative and qualitative, are taken to evaluate the architecture. First the fundamental requirements posed by the Alpine framework are summarised along with a description of success or failure of the resulting implementation to comply. A qualitative evaluation technique is used to consider the framework beyond the scope of the implementation. To aid this, a case study and a comparison to a state-of-the-art router architecture are introduced.

Evaluation of the prototype Active Router implementation is considered according to two criteria: service instantiation latency, and interface performance. Service instantiation latency is affected by two key factors, the mechanism to determine the appropriate service implementation (the Component Compatibility Mark-up Language (CCML) XML based meta-language), and the approach taken to download and execute the chosen implementation.

The primary purpose of dynamic upload and instantiation of code to the active router is to provide an interface (at the IEEE P.1520 'L' reference point) to allow customer centric network control from off-board Operations Platforms. The mechanism by which this interface is constructed is not mandated, and the selection of an appropriate scheme is a matter for the active code implementation. To discuss the effect this selection has, two remote prodecure call protocols are considered, the established and proven IIOP used in most CORBA ORB implementations, and the state-of-the-art SOAP protocol leveraging the Internet friendly HTTP and XML technologies. The selection of these two approaches highlights the trade-offs to be considered by the active service developer/provider in terms of code implementation, interface performance, and size of on-the-wire messages.

Finally, Chapter 8 concludes the thesis. The contributions made to the research community are summarised, along with suggestions for future directions and developments.

# Chapter 2

# Motivation

It is evident that the telecommunications and data communication industries are converging, enabling the creation of an Information Highway, forming a new society nerve structure in this first decade of the 21st century. Information technology industry convergence is not a recent phenomenon but a trend dating back to the late 1980's. This chapter looks back to the early stages of this trend and tries to take stock and learn from the mistakes and false assumptions that were made. The results of this analysis are used later in the thesis as input into the requirements of the future multi-service Internet.

During the late 1980's and 1990's three important trends can be observed within the telecommunications, data communications, and information technology industries; deregulation, the emerging wide variety of transport media, and the development of new distributed and object oriented software engineering technologies and processes.

### 2.0.1 Deregulation

The history of the provision of telecommunications services within Europe and the US has been based on single operators having a vertical monopoly (i.e. a monopoly spanning multiple tiers of the telecommunications business). In most European countries, for example the United Kingdom, telephony was provided by the state-owned General Post Office and integrated with other forms of communications (Post, and Telegraphy).

These monopolistic operators owned the sole licence for telephony service provision over all tiers of the business, from running the network infrastructure, ownership/installation of Customer Premises Equipment, to provision of value-added network services. Although politically unfashionable now, this monopoly enabled the provision of the universal access that we enjoy today; telecommunications services were installed based on benefit to society as opposed to operator profit. Indeed there is much still to be made of this argument today for the provision of broadband services.

Deregulation has been a policy followed by most major European governments over recent years (and mandated by the European Commission at the beginning of 1998) to introduce competition into the telecommunications sector. The idea is that a large number of reactive small and diverse companies will provide a greater benefit to society (through lower prices and more innovative services) than large public utility companies.

In the United Kingdom the government was committed to privatisation and deregulation both for ideological reasons and to reduce government borrowing. According to the White Paper entitled "The Future of Telecommunications in Britain" (DTI, 1982) and the statement to the House of Commons by the Secretary of State:

> "Monopoly power has allowed British Telecom to raise prices without doing all that could be done to increase efficiency. Around 90% of British Telecom's investment programme, around £2200 000 000 this year (1982) has been self-financed by customers."

However, deregulation is a misnomer, because far from the removal of regulation, there is an increased role for regulation. In order to enforce the liberalised telecommunications market, against the background of dominance by one operator, it has been necessary to create the new office of the Telecommunication Regulator (the Office of Telecommunication, OFTEL, within the UK). It should be noted at this point that transparency of market conditions to enable effective regulation should be an important function of any telecommunications model such as the Alpine framework discussed within this thesis.

Networks of the future are unlikely to be composed of a single operator owned infrastructure, instead they will be built from the federation of many networks owned by many companies and scattered over many geographic domains. In the past, networks could adopt their own control and management schemes and protocols, since interconnection was limited to infrequent international calls. Inter-operability is a key function of future telecommunications architectures and protocols, increasing the role of international standardisation bodies such as the ITU (International Telecommunication Union), ETSI (European Telecommunications Standards Institute), and the IETF (Internet Engineering Task Force).

### 2.0.2 Heterogeneous Transport Technology

Developments in fibre-optic and wireless communications in addition to the forces of deregulation have spawned a multitude of data network transport technologies. These have changed the availability, price and scale of bandwidth and the location of use. Two new network features, broadband and mobility, have had a considerable effect on the communications services available today.

The emergence of broadband networks can be characterised by the availability of SDH/ATM transport technology. Whereas the previous narrowband networks carried data in a 64kbit/s channel, broadband networks efficiently provide potentially many gigabits/s data transport.

The Synchronous Digital Hierarchy (SDH) technology provides a data-link-layer mechanism to enable high data throughput based upon optical networks. It allows much higher bit rates to be switched than with the previous Plesiochronous Digital Hierarchy (PDH) due to its much more efficient channel multiplexing.

The introduction of ATM (Asynchronous Transfer Mode) during the early 1990s provided over the SDH data-link layer a high-speed, low delay, low jitter transport ideally suited to multimedia data. ATM is based on cell-switching technology, each 53 byte cell containing 48 bytes of user data. Central to ATM is the provision for Virtual-Paths (VP) and Virtual-Channels (VC). A physical link is composed of multiple VPs, which in turn may contain multiple VCs. Figure 2.1 demonstrates the multiplexing of Virtual Paths and Channels in ATM networks. This separation, coupled with the ability to allocate scheduling resources to these VPs/VCs, enables the construction of virtual private networks, each with its own Quality of Service (QoS). Because of this, ATM has been widely adopted by Internet Service Providers (ISP) as the basis for their high-speed backbones.

Unlike the connection-less IP datagram protocol, a virtual circuit must be established before data can be transferred. These virtual circuits are either Permanent Virtual Circuits (PVC) established manually at configuration time, or Switched Virtual Circuits (SVC) established at run-time by a signalling mechansim.

The combination of SDH and ATM has enabled data-transmission rates over fibre-optic up to several gigabits per second. However to meet growing demands for bandwidth beyond that of traditional TDM (Time Division Multipexing) new technologies such as Dense Wavelength Division Multiplexing (DWDM) has been developed that multiplies the capacity of a single fibre sixteen fold, to a throughput of 40 Gb/s. Using these ultra-high-speed cross-connects, future networks are abandoning the intermediate ATM


Figure 2.1: ATM VPI/VCI Architecture

layer, instead running IP directly over SDH and DWDM.

### 2.0.3 Distributed Object Technology

The design and development of software systems has in recent decades been revolutionised by the emergence of object oriented methodologies. These range from object oriented system requirements capture and design using techniques such as UML (Unified Modelling Language), to object oriented programming languages (e.g. C++ and Java) and component frameworks (e.g. Microsoft's COM and Sun Microsystem's JavaBeans).

Naturally the same object oriented approach could be used for the development of distributed systems containing components distributed over multiple geographically remote processing elements.

A good definition of what constitutes a distributed system is hard to determine, however a useful definition may be found in the book "Open Distributed Processing and Multimedia"(Blair & Stefani, 1998):

> "A distributed system is a system designed to support the development of applications and services which can exploit a physical architecture consisting of multiple, autonomous processing elements that do not share primary memory but co-operate by sending asynchronous messages over a communications network."

Amongst the many distributed systems standardisation activities there have been four important initiatives:

1. *ISO/ITU-T Reference Model for Open Distributed Processing (RM-ODP)*
   The Reference Model for Open Distributed Processing (RM-ODP) is an international standard for distributed systems modelling developed by the ISO and ITU-T. Rather than define an "on-the-wire" protocol standard for distributed communications, RM-ODP defines an architectural framework (ITU-T, 1995b).

2. *The OMG's Common Object Request Broker Architecture (CORBA)*

   The Object Management Group (OMG) was founded in 1989 by a collection of IT and telecommunications vendors with the aim *"to create a component-based software marketplace by hastening the introduction of standardised object software"*. The conformance to these specifications will make it possible to develop a heterogeneous computing environment across all major hardware platforms and operating systems.

   Today the OMG has over 800 member companies, and is responsible for the development of CORBA and maintenance of the UML design language.

   The Common Object Request Broker Architecture (CORBA) is a mechanism to allow applications to communicate, no matter where they are located, what platform and operating system they are executing on and in what programming language they are written. CORBA is a standard; it is up to vendors to develop object request broker (ORB) implementations that are CORBA compliant. The ORB is the *middleware* that establishes the client-server relationship, and provides the transparency that allows a client to invoke methods on a server. These requests are passed between ORBs using the General Inter-ORB Protocol (GIOP), specialisations of which have been developed to suit specific network environments (e.g. Internet Inter-ORB Protocol (IIOP) for IP networks).

   CORBA 2.0 (OMG, 2000c) adopted in December 1994 is the accepted base standard, and implementations have been made by many vendors including Iona, Inprise, and Sun Microsystems. The OMG have worked on a third generation of CORBA specifications (CORBA v3) which was released during the year 2000.

3. *Distributed Computing Environment (DCE)*

   The Distributed Computing Environment (DCE) (Open Group, 1990) is a distributed client-server RPC (remote procedure call) service, but unlike CORBA does not provide an object-oriented environment. It was developed by the Open Group, which is a non-profit de-jure standardisation body similar to the OMG, set up by IT systems vendors.

   DCE provides a set of services providing a distributed processing environment built upon the core DCE RPC mechanisms. These services provide distributed time, naming, distributed file systems, security and management. The aim is to provide a robust distributed platform capable of providing security and reliability necessary in many critical systems domains. DCE is commonly found in large-scale systems in the finance and banking industries.

4. *Microsoft's Distributed Component Object Model (DCOM)*
   Distributed Component Object Model (DCOM) (Microsoft Corporation, n.d.) is an extension of Microsoft's successful COM technology enabling inter-node object communication. COM is the basic 'componentware' that forms the glue between components and enables software components on the Microsoft Windows family platform to interact. It is an evolution of the Dynamic Data Exchange (DDE) and Object Linking and Embedding (OLE) first introduced to enable application data exchange and compound documents (e.g. embedding Excel spreadsheets in Word documents).

   The goals of DCOM are similar to that of CORBA, i.e. to provide distributed object binding and communications. Both claim to offer location independent object binding, object model and language independence, and similar support services (e.g. transactions and persistence). However although DCOM implementations do now exist for Unix derived platforms, in reality DCOM applications are heavily dependent on the Microsoft Windows family of platforms. For this reason, due to the number of legacy systems deployed within telecommunications systems, DCOM is probably a less appropriate distributed technology for this particular application domain.

The use of distributed system technology for the provision of telecommunications services can lead to more efficient service provision. The definition of interoperability standards is especially important within the telecommunications domain when we consider the heterogeneous network technologies deployed and the increasing operator interoperability required by the forces of deregulation. It is also claimed that object oriented techniques increase software robustness and enable code reuse.

The distributed component interoperability opens the market for value-added service provision. Rather than an operator providing all horizontal market planes, third parties can be granted access to the network infrastructure to enable them to provide competitive services. Initiatives include the OMG Telecommunications Domain Task Force's Open Services Marketplace Model (Schenk *et al.*, 1999), and the interface definition activity of the Parlay Consortium (Parlay Group, 1998).

## 2.1 Multimedia Applications and Services

Traditional telecommunications and information technology applications and services are limited to a single medium. The classical voice telephony service provides a single media channel for voice communications, whereas a bulletin board service such as Prestel from British Telecom is purely text based.

Significant advances have been made in computational devices, particularly the advent of high powered desktop personal computers incorporating high resolution colour visual display units, and high capacity storage media, for example CD-ROM and DVD drives. This has enabled the creation of applications using multiple media (sound, video, text, etc). A good definition of multimedia applications is given in the ITU-T Recommendation for audio-visual/multimedia services (ITU-T, 1996):

> "A Multimedia Application is an application that requests the handling of two or more representation media (information types) simultaneously."

Advances in communication technology (see section 2.0.2) have increased the end-to-end transmission bandwidth available from the 64 kbit/s voice channel to many megabits/s through technologies such as ATM and Gigabit ethernet. Increasing multimedia is not confined to distribution on CD-ROM and DVD for standalone PC applications but can be used to provide multimedia telecommunications services. A good definition of multimedia services can be found in the ITU-T Recommendation for audio-visual/multimedia services (ITU-T, 1996):

> "Multimedia services are services that handle several types of media in a synchronised way from the user's point of view. A multimedia service may involve many parties, multiple connections, and the addition or deletion of resources and users within a single communication session."

Multimedia services potentially require significant quality of service (QoS) from the telecommunications network. However, the QoS demands are quite diverse, and to further understand these requirements, multimedia services are categorised below. The categorisation examined in this thesis is taken from the ITU-T B-ISDN Service Aspects Recommendation I.211 (ITU-T, 1993).

Multimedia services are divided into two broad categories, interactive and distribution services (figure 2.2). Distribution Services are generally multiparty uni-directional services, where a single source transmits multimedia data to many sinks. The most common type of distribution service today is RF broadcast television but cable (e.g.

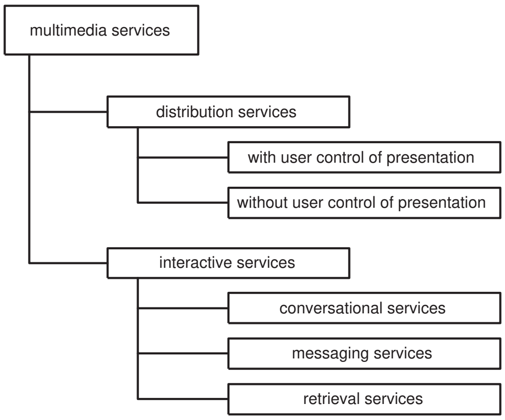


Figure 2.2: Multimedia service categories

CATV) is just as suitable a transmission medium. Distribution Services are not limited to audio-visual, Teletext services are broadcast to many terminals regardless of demand and it is up to the end terminal to display the page required by the user. This category may be further divided into two subsections. First, services on which the user has control over presentation, for example, selecting the desired camera view of a football match or commentary language from a given list, as recently available for digital TV customers. The second subsection, are those traditional broadcast services where the user has no control over presentation, for example, conventional radio and TV broadcasts.

Interactive Services are those with bi-directional feedback. However, this does not preclude those services from being multiparty. Typically, interactive services are subdivided into three further categories, conversational, messaging, and retrieval.

1. Interactive Conversational Services: These services can be characterised as real-time and bi-directional. Classical telephony services are an example of an interactive conversational service involving only a single medium. It is important to understand that despite the label "conversational" this does not limit the service to voice communication and could include video and textual media. The service may be either two-party bi-directional or may contain many parties. However, unlike distribution services, participants in conversational services are normally equal peers rather than broadcaster or receiver.

2. Interactive Messaging Services: Whereas conversational services provide support for real-time communication, messaging services provide an off-line style of communication. The most common type of messaging service is that of electronic mail. Today email messages can contain pictures, video clips and audio samples as well

as textual information.

3. Interactive Retrieval Services: In an interactive retrieval service, users access multimedia sources stored in multimedia library archives. Similar to distribution services, the participating entities are not equal peers, but instead multiple users (information sinks) and one or more multimedia archives (information source). Video on demand (VoD) is a much-quoted example of an interactive retrieval service. With a VoD application the user first selects a film choice from a menu, then at an agreed time (possibly much later) the video is streamed from the archive to the end user's terminal.

## 2.2 The search for broadband service provision

The forces of market-driven pull for ever advancing demands for multimedia applications, coupled with the push of continual increases in broadband bandwidth, are posing new challenges for service provision. As the market for network services becomes more competitive and is subject to increasing regulation, network operators must look to raising revenue from the sale of value-added services.

Unfortunately no solutions to broadband service provision can be adopted from the current best of technology (Brendan *et al.*, 2000). Over the next two subsections, two legacy approaches are examined, first current narrowband solutions, and second the B-ISDN approach, considered by many during the early 1990s as the model for future broadband network provision.

### 2.2.1 Legacy narrowband solutions

The introduction of computation on stored program controlled (SPC) telecommunication switches during the 1970s opened the possibility for the PTTs (Post, Telegraphy and Telephony) to provide "supplementary services" to the ubiquitous voice transmission service. However introducing new services represented a very high cost due to the need to introduce these services into the wide variety of switch control software in use.

The Intelligent Network (IN) concept (Berman & Brewster, 1992) was developed during the 1980s as a means to deploy cheaply and rapidly services on the PSTN (Public Switched Telephone Network). This has been used to provide the "0800" (freephone), "1471" (BT last number called service), calling card, call waiting, and call holding services routinely expected from modern telecommunications networks. The IN architecture is comprised of a set of distributed functions that interact during call origina-

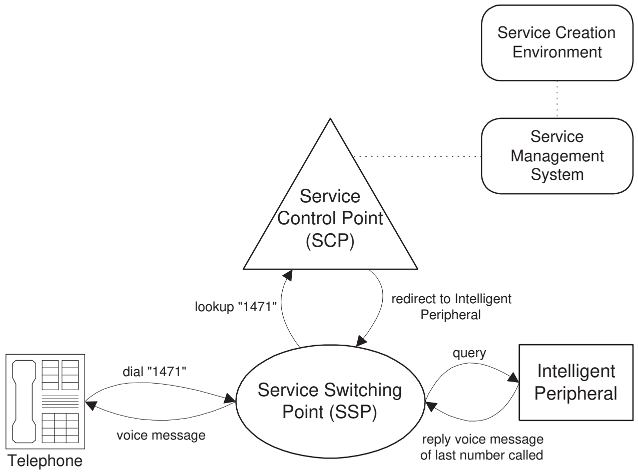


Figure 2.3: IN Architecture

tion/termination to provide customised services. Fundamental to IN is the separation of switching from network intelligence. To achieve this, IN distinguishes between Service Switching Points (SSP) and Service Control Points (SCP). An SSP is a modified telephone exchange incorporating trigger tables to detect IN call processing events and consult the SCP. The SCP is a specialised computer distinct and, possibly, remote from the SSP that runs the specialised IN service logic. The SCP invokes this service logic in response to INAP (Intelligent Network Application Part) signalling messages originating at an SSP. In addition IN makes use of dedicated equipment known as Intelligent Peripherals (IP) to play announcements, produce tones, and queue calls.

To enable service creation and management, two other entities, the Service Management System (SMS) and the Service Creation Environment (SCE), are provided. The SMS is an offline facility used to prepare network and customer specific configuration information and download then to the network SCPs. The SCE platform is the development and testing suite used to develop and debug new IN service software.

An example IN architecture and case study of how these components might interact to provide a "1471" service is shown in figure 2.3. In this example the customer has failed to answer a previous call, and therefore to obtain the phone number of the originator, dials "1471" into their telephone. The local exchange (providing the SSP function) detects that this number belongs to an IN service and directs a request for further information about how to process this call to the SCP. The service logic at the SCP finds an appropriate Intelligent Peripheral (IP) to provide this service, and replies to the SSP with the location of the IP to which to re-direct the call. Once the call has been

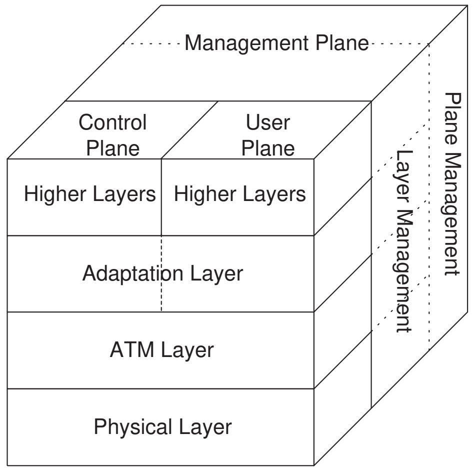


Figure 2.4: B-ISDN Reference Model

established the IP replays a voice message containing the information requested by the customer.

### 2.2.2 B-ISDN

According to ITU-T Recommendation I.113 ("Vocabulary of Terms for Broadband Aspects") (ITU-T, 1991a), the term broadband is defined as

> "A service or system requiring transmission channels capable of supporting rates greater than the primary rate (up to 2 Mbits/s)"

B-ISDN (Broadband Integrated Services Digital Network) is an evolution of the ISDN network beyond the 64kbit/s channel capability. It is a model for broadband communication provision and does not itself mandate an appropriate transport technology. However according the ITU-T Recommendation I.121 (ITU-T, 1991b) candidate technologies should:

> "Support switched, semi-permanent and permanent, point-to-point and point-to-multipoint connections and provide on demand, reserved and permanent services. Connections in B-ISDN support both circuit and packet modes, in both bidirectional and unidirectional configurations"

.

ATM (Asynchronous Transfer Mode) was chosen as the most likely candidate transport technology for B-ISDN although not fully compliant with ITU requirements. The

B-ISDN reference model (see figure 2.4) has been developed to logically separate functionality into a three dimensional structure. Three vertical planes are defined, a user plane to transport user information, a control plane composed of signalling information, and a management plane. Additionally a fourth functionality, plane management exists responsible for the management of the three former planes. Each plane is composed of a layered protocol hierachy similar to the OSI and TCP/IP protocol stacks. However the mapping between the B-ISDN reference model and the well established OSI model is not obvious.

## 2.3 TINA - a complete broadband telecommunications architecture

Current telecommunications networks have been built upon a number of different business models and software systems for each part of the network (e.g. control software, management software and billing software). The interaction between systems is hence severely restricted. Combined with this constraint, network operators have come under increasing competitive pressure to reduce costs and provide more advanced services in order to retain customers. The process of standardisation through international standardisation bodies (e.g. the ITU, ETSI, etc) is a slow process, taking many years to adopt a new standard. Network operators can forge ahead and define their own proprietary protocols and solutions. However, in the increasingly multi-network operator environment this is not an attractive solution.

The Telecommunications Information Networking Architecture (TINA) (Inuoe *et al.*, 1999), dubbed *"a co-operative solution for a competitive world"* and developed by the TINA Consortium (TINA-C), aimed to provide a complete framework for the provision of telecommunication services. The TINA-C was established in late 1992 by a collection of leading telecommunications operators and equipment vendors, to develop a service delivery framework and corresponding software specifications. The TINA Core Team, a group of researchers and engineers from the member companies was established the next year and ran until 1997, to perform the bulk of this effort. Since 1997, TINA has relied upon input from the many auxiliary projects, and a new procedure for developing specifications based upon the issuing of "Request for Proposals" for specific problems, similar to the procedure used in many other fora such as the Object Management Group (OMG). TINA is in any case not a true standards body, aiming instead to influence other appropriate bodies (ITU, ETSI, IETF, etc).

There are 5 essential principles that formed the basis of the development of TINA:

1. Competitive business model: defines a model where provision is divided into separate functions, each of which may be provided by a separate independent agency. For further details of the TINA business model see section 2.3.1.

2. Adopt software-engineering approaches: embrace modern approaches to software design and development from the IT and computing industries. This should enable cheaper and faster development of systems that are both more reliable and better extendible to meet future evolution.

3. A single computation platform: traditionally there has been a separate platform for supporting user services (e.g. IN) from that of management (e.g. Telecommunications Management Network (TMN)). Not only does this increase costs but also restricts interaction between different parts of the system. TINA uses a single platform exploiting distribution for load balancing and reliability. Functional entities can consequently be easily relocated to appropriate parts of the architecture.

4. Isolate network entity: the concept of services is separated from that of the network. The network itself may consist of a federation of many networks from many different operators. Networks may be composed of heterogeneous technologies, some may support third party control and management, while others may perform all control and management internally (e.g. IP networks).

5. The notion of services and sessions: as noted above, sessions are an independent entity within the TINA architecture, and are controlled autonomously of the underlying network. The concept of the session is introduced for service control; a session models all the resources used in the provision of a service to a given group of consumers (group may change members during lifetime of session) for a given lifetime.

It is important to understand that TINA does not attempt to recreate or redefine the existing wealth of research and development on distributed systems methodologies. Contemporary design and specification techniques are adopted to provide a common language to reach a common understanding of the TINA architecture. These originally included the Reference Model for Open Distributed Processing (RM-ODP), the Object Modelling Technique (OMT), and quasi-GDMO (Guidelines for the Definition of Managed Objects). More recently the OMG's Unified Modelling Language (UML) and

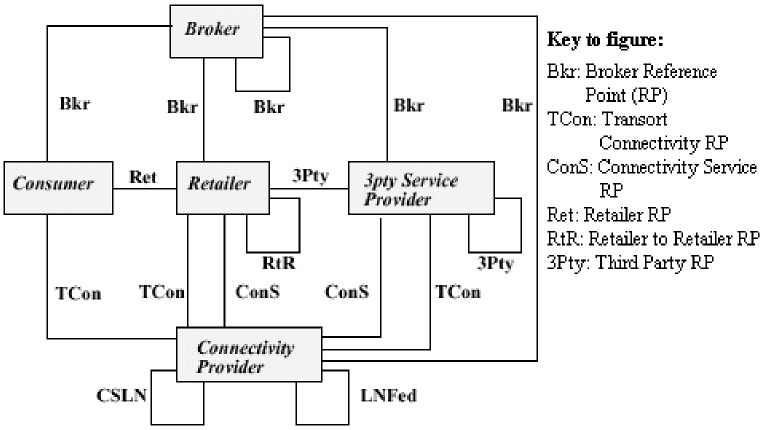


Figure 2.5: TINA Business Model

Interface Definition Language (IDL) have been adopted as the primary formal specification methods.

### 2.3.1 Business driven Reference Points

Central to the object oriented design and analysis used in TINA is the adoption of RM-ODP modelling concepts. RM-ODP uses five viewpoints: enterprise, computational, information, engineering, and technology, which are used within the TINA specifications to describe the system from a particular perspective. In this section, the TINA Business Model (TINA-C, 1997a) is considered, which is based upon the RM-ODP enterprise viewpoint.

Five business roles are defined (figure 2.5), which stakeholders within the telecommunications industry may perform. These are defined as follows:

1. Consumer - a stakeholder in the consumer business role takes advantage of the services provided in a TINA system. Unlike other business roles, the consumer has no interest in earning income from a service. It is possible that a stakeholder may perform multiple roles, for example a network operator in order to test a network may be both consumer and connectivity provider.

2. Third party service provider - provides wholesale services to the retailer business role. A classic example of a third party service provider is that of a content provider. The important distinction between this role and that of the retailer is that the third party service provider does not have a direct relationship with the consumer.

3. Retailer - is the agency responsible for selling a TINA service to the consumer business role. The retailed services are based on customer support and value adding to the service provided by the third party service providers. It is the point at which the customer makes payment for the services it consumes, and where appropriate the retailer will pay (using a different accounting model) the third party service provider and connectivity provider. It is likely to be very common for a stakeholder to perform both third party and retailer business roles.

4. Broker - the mission of the broker is to provide stakeholders (e.g. consumers) with information that enables them to find services within the TINA system. It aims to be an enabler for open free market supply of TINA services by providing an important facet of free markets; perfect information about the supply of goods and services amongst infinite (perhaps a large number within the international marketplace) suppliers and consumers.

5. Connectivity Provider - owns and manages a physical network. This can be a transport network to support user stream bindings or a kernel transport network (kTN) enabling service control and administration. A separation from service provision is drawn so that each may develop at its own pace in line with technological advances.

### 2.3.2 Computing Architecture

Contrary to the common perception, TINA does not require the adoption of an ORB (Object Request Broker) based on the OMG's CORBA (Common Object Request Broker Architecture) standard to form the basis of the TINA computing architecture. Instead, TINA specifies the requirements to which the Distributed Processing Environment (DPE) must conform (TINA-C, 1997b). This draws heavily on the background of other DPE platforms (e.g. DCE, and ANSA) that were available in the early 1990s. A TINA compliant telecommunications software system can potentially be built using OMG CORBA, Microsoft's DCOM, or even Java distributed technology.

Nevertheless the TINA requirements are closest to being met by the OMG's CORBA standards. The aims of the OMG are however far broader than those of TINA; CORBA based DPEs may be found in many additional domains other than telecommunications, including those of education, medical, industrial, and transport. The telecommunications domain requires a specialisation of the general DPE architecture, an example of which is the TINA computing architecture. The OMG have many internal bodies considering the requirements of specific domains, one such being the OMG Telecommunications Domain Taskforce that has on many occasions adopted results from TINA.

The adoption of the TINA DPE as the universal communications medium should offer greater service interoperability. Legacy systems have deployed many separate communications media, Signalling System 7 (SS7) signals for connection control, INAP (Intelligent Network Application Protocol) messages for network intelligence, CMIP (Common Management Information Protocol) messaging for TMN, and ISDN UNI for user to network signalling. The challenge for the TINA DPE is to provide for these broad usage requirements within the telecommunications domain.

The ISO/ITU-T RM-ODP prescribes nine types of distribution transparency that may be found in a distributed system. These are detailed in table 2.1. Current DPE

| Transparency | Effect |
|---|---|
| Access | Hides different representations of data and invocation approaches |
| Concurrency | Co-ordination of many simultaneous interactions |
| Failure | Detect failures and recovery in a way masking the failure from the user where possible |
| Federation | Masks that invocations may take place between multiple administrative domains |
| Location | Enables invocations to be made on objects whether the client and server are located locally on the same DPE node or dislocated and connected over a computer network |
| Migration | Disguise from the perspective of the invoking client that the location of a server object has altered |
| Replication | Mechanisms are provided to allow object/service replicas to be constructed and consistency maintained |
| Persistency | Hides the mechanism to maintain data between deactivation and reactivation of objects/services |
| Transaction | Provides a mechanism to enable rolling back of transactions to maintain consistency of data upon failure of a transaction |

Table 2.1: RM-ODP Distribution Transparency

standards do not provide support for all nine types of transparency; however, the TINA DPE stipulates only access and location transparency as mandatory, both of which

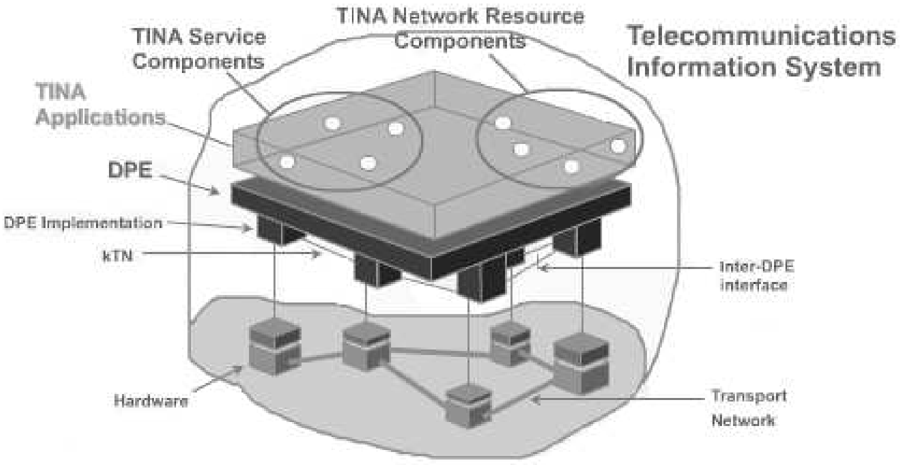


Figure 2.6: TINA Software Architecture

are provided for by CORBA. TINA applications providing TINA services are deployed upon a DPE infrastructure as depicted in figure 2.6. The main elements are the DPE implementations, the kTN, the Native Computing and Communication Environment (NCCE), and the Transport Network. An important distinction is made between the Transport Network and the Kernel Transport Network (kTN). The Transport Network is owned by the Connectivity Provider, and is used to provide end-to-end carriage of user stream based traffic. The kTN is a network used to interconnect all DPE nodes enabling software components to communicate. Use of the kTN is not limited to communication between software components within the Connectivity Provider business role but could also include access and service session negotiation carried out between the Consumer, Broker, Retailer and Third Party Service Provider domains.

### 2.3.3 Network Resource Architecture

The Network Resource Architecture (NRA) (Steegmans & et al, 1997) combines the roles of control and management of the underlying network infrastructure. Traditionally these two roles have been separated, e.g. through SS7 signalling and TMN management in telecommunications networks, and RIP (Routing Information Protocol), OSPF (Open Shortest Path First), and BGP (Border Gateway Protocol) routing, and SNMP (Simple Network Management Protocol) network management within the Internet.

The combination of these roles in a single architecture based upon a single DPE should lead to a more uniform design enabling component reuse, lower cost through a single DPE infrastructure, easier interactions between control and management functions, and sharing of information.

Unlike most conventional approaches to control and management, the network of the future will be constructed from a range of heterogeneous technologies. In parallel the type of network services will expand, future demands will include guarantees of Quality of Service (QoS) and support for multiparty, multipeer communications. The Network Resource Information Model (NRIM) (Natarajan & et al, 1997), was developed as a means to abstractly define network resources and network usage in a generic manner independent of underlying network technology.

As such, NRA defines the components that performed the tasks of the Connectivity Provider business role from an RM-ODP computational viewpoint. NRIM defines the information manipulated by those components from an RM-ODP information viewpoint.

With NRA, there are four separate functionalities, Connection Management, Network Topology Configuration Management, Fault Management, and Accounting Management. To date, Connection Management has been considered in the most detail, with a number of validations through implementation (e.g. Alcatel's ALCIN).

From a computational viewpoint Connection Management is composed of a hierarchy of computational objects. This divides the problem of management of a large network of heterogeneous technologies (possibly in different ownership domains) into a manageable problem. The top-level component, the Connection Co-ordinator (CC), exposes the ConS (Connectivity-Service) reference point between the Connectivity Provider and the Retailer. It delivers an abstract view of network resources in the form of physical connection graphs, and offers functions to enable: network flow connection set-up/release, addition of branches to multipoint flows, modification of QoS parameters, and activation/deactivation (move from locked (resources undergoing configuration/management) to unlocked state (in use for transport)) of flows.

Beneath the CC, the overall connectivity network is composed of one or more layer networks (subnetworks of heterogeneous technologies and owned by multiple agencies). A layer network denotes a network based upon a single technology, for instance an ATM VP Layer, an ATM VC layer network, or frame relay. The notion of a layer network is not new to TINA, originating from ITU-T Recommendation G.803 (ITU-T, 1992a) for SDH networks, and later adopted as a generic model in ITU-T Recommendation G.805 (ITU-T, 1995a). The Layer Network Co-ordinator (LNC) is the computational component responsible for the management of layer networks.

Within the Network Management Layer (NML) the subnetworks that form a layer network are further decomposed through the composition of a hierarchy of Connection

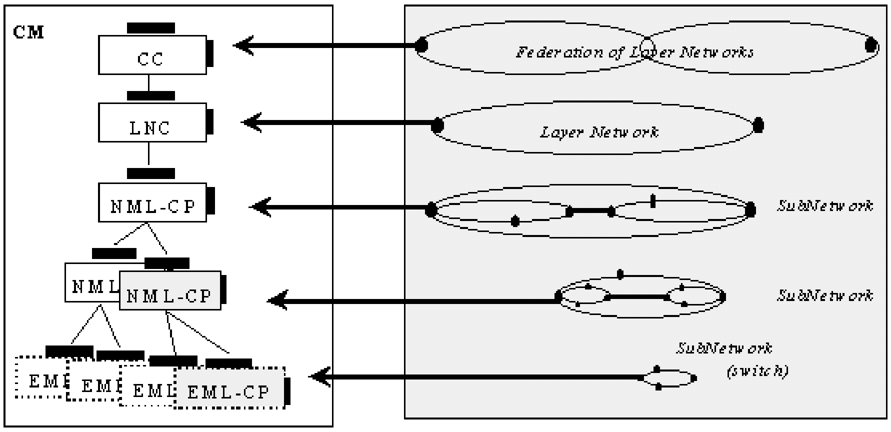


Figure 2.7: TINA Connection Management Architecture

Performers (CPs) acting over child subnetworks (figure 2.7) until, at the lowest level (Element Management Layer (EML)), the open interfaces to switch resources are reached.

## 2.4 Discussion on the impact of TINA

Although many now write TINA off as a failure this is simply because they do not fully understand its objectives. It is certainly true that eight years since the creation of TINA-C, a network operator cannot purchase a fully TINA compliant system; however much has been learnt and is being fed into other fora. In this section we discuss why TINA has failed to make a dramatic impact and where its successes are being quietly adopted.

The most obvious criticism to make is indeed that of lack of adoption. In fact in many areas the TINA specifications are still currently incomplete, for example within the Network Resource Architecture (NRA) only the ConS reference point has been specified in detail. Any implementation would therefore be forced to fill in the gaps, but this would likely lead to incompatibilities between multiple vendors' software, a problem that TINA was supposed to solve. With the TINA Consortium activity having ceased at the end of the year 2000, many of these incomplete specifications will never actually be completed.

The high complexity of the TINA specifications is another frequently raised issue. The Core Team based at Red Bank, USA, were mainly concerned with building an all-encompassing telecommunications model. Inevitably the result of this activity was fifteen or more specifications, each generally at least 200 pages long. Although much manpower was devoted to formally validating the resulting architecture, only limited effort was put into implementation. No complete reference implementation has been made available so that the design can be practically validated and potential customers (network operators) can test whether TINA is beneficial for their business. Instead, multiple closed implementations have been made of small subsets of the system through numerous TINA trials funded mainly by Network Operators (e.g. the GlobalOne Alliance TINA Trial), the European Union ACTS and RACE programmes, and EURESCOM.

The lack of an industrial strength Distributed Processing Environment (DPE) is certainly a headache for any software vendor wishing to build TINA compliant telecommunications software. Although CORBA is seen as the basis for the TINA DPE, it does not yet provide the feature rich ORB that is required. Notable omissions include support for real-time communication, non-functional requirements, and a much simplified object lifecycle model. For example, the CORBA v2 standard (as implemented by most major ORB vendors) was based upon a synchronous RPC mechanism. Asynchronous communications was only provided through the one-way invocation mechanism, and stream support was not included. This has since been addressed by the OMG's Telecom Domain Taskforce, with the adoption of the Control and Management of A/V stream specification (OMG, 1998). Unfortunately at present not all ORB vendors have implemented this, and many are still using proprietary solutions.

Non functional requirements, such as performance, fault tolerance, and availability are also poorly provided for by CORBA. Performance requirements such as timeliness of invocations have not been mandated; current ORBs have been found to vary widely depending upon vendor, platform, and language binding (Charles University, 1998).

Second, in regard to the DPE, TINA mandates that customer's end user terminals are part of the TINA DPE. However, in general, the terminal devices we use today do not include a DPE implementation and hence could not be part of a TINA network. The increasing trend is towards smaller low power consumption portable devices (e.g. WAP phones, Psion and PalmPilot organisers); these do not have the computational power nor memory necessary to run a fully featured DPE ORB. The only option is to terminate the TINA network in proxies within the network and for those proxies to translate between legacy protocols and the TINA system.

Until recently TINA could easily be criticised for failing to take account of the changing business model for telecommunications, and the rapid rise of the public Internet. The TINA business model (see section 2.3.1) correctly recognised the advent of competition and multiple parties involved within the provision of services, and while still valid, it does not represent the rich business relationships developing within the Internet community. The Internet supports a multitude of business roles, for example Internet Service Providers (ISP), Internet Content Providers (ICP), Internet Application Service Providers (IASP), Internet Portal Providers (IPP), Core Network Operators etc. The emerging electronic commerce business model encompasses both customer to business (c2b), and business to business (b2b) transaction models.

The TINA Service Architecture is considered by TINA devotees to be one of the key features that should survive. It defines the use of generic service objects so as to enable component reuse and speed-up and lower the cost of service deployment. Central to this is the definition of the session notion, divided into three parts, the access, service and communication sessions:

1. Access Session - the initial authentication and profile selection between the user and the TINA system

2. Service Session - negotiation of participation in a particular service

3. Communication Session - co-ordination of network resources used in order to provide the service.

With the recent dominance of the “Web” as the predominant paradigm for communications, it is questionable whether the TINA Service Architecture will ever be realised.

The Network Resource Architecture (NRA) is another contentious issue, although supposedly technology independent, it is based on a connection-oriented third party control model. This has been shown to map easily to ATM through numerous demonstrators such as those developed in the ReTINA (see section 3.2) and VITAL projects, funded within the EU ACTS programme. However, a mapping to connection-less technology (e.g. IP) is non-trivial, and is currently the subject of ongoing study (see section 3.3.2).

The NRA can represent a heavy overhead for connection management because of the potential to construct a deep hierarchy of computational objects, through which control requests must pass. The alternative construction of a broader, less deep hierarchy results in excessive computation to be executed at a single DPE node. The DPE itself represents an additional overhead; the act of marshalling/de-marshalling invocation requests and data, and the transmission over the kTN all further impact on connection management performance.

Naturally, a hierarchical network model has a root computational element to which all control and management operations must be made. This impacts upon the potential for concurrency of network control and management, the obvious solution of replication of the root node brings instead problems in maintaining consistency amongst replicas. Potentially more serious is the existence of a central point of failure within the system software; the outage of the root computational element renders the entire network unavailable, clearly an unacceptable risk for a telecommunications network.

At the element layer, the switching fabric is under the command of the Connection Management software. However, current switches do not include an ORB, and hence are not part of the TINA DPE. Indeed support for third party switch control is very poor; vendors see their proprietary control solutions to be an integral part of a switch. The retail value of a complete switching package including both physical hardware and control and management software is seen as being greater than the sum of its parts. There is therefore, resistance to opening switches to provide interfaces for alternative control software. Although open protocols now exist, for example the IETF is standardising the General Switch Management Protocol (GSMP) (Doria *et al.*, 2000), few switch vendors implement it or have any plans to support any of these protocols.

Lastly, network operators have invested vast sums in legacy network technologies, and any new technology must ensure backward compatibility and offer a solution for easy migration. The OMG's Telecom Domain Task Force has already considered mappings from legacy signalling protocols such as CMIP, SS7 and INAP to a DPE based system. Implementations are still hard to find, but the advantages of having a single computing architecture are clear and will remain attractive.

## 2.5 Emergence of the Internet

The Internet emerged from a United States government and military initiative to develop a network to interconnect computers in a manner that was robust to attack. Since then the Internet first became used as an academic network, and more recently a consumer and business network forming the nerve structure of the information society.

Applications used on the Internet have up to now had only weak Quality of Service (QoS) requirements. Indeed applications such as email, FTP, and the World Wide Web (WWW) have in the main been concerned only with integrity of transfer as a measure of service quality. Future applications are unlikely to be so unconcerned about the quality of service provided by the network. Internet Telephony, Video on Demand, Co-operative working applications, and multiparty games result in tight requirements on delivery of data, especially timing.

The Internet Engineering Task Force (IETF) is the body responsible for the design and development of new protocols and standards. It is not responsible for the implementation of these standards, nor the running of the Internet; the success of a standard depends on whether it becomes widely adopted. Although there is no formal membership within the IETF and anyone is welcome to contribute, it has become heavily vendor driven, and network operators (e.g. core network provider and ISPs) appear to have limited impact.

It is a commonly stated fallacy that, since the connection-less IP protocol lacks an identifiable unique call set-up stage, therefore signalling (see section 5.1.1) has no place in the Internet. Further, this leads to the false view that since the Internet is a multi-operator environment with autonomous dynamic routing protocols (e.g. RIP, OSPF, BGP), there is no place for traditional network management.

In this section, the emergence of management and associated signalling mechanisms are considered for the provision of QoS for multimedia applications. Two main approaches have emerged as possible solutions to the provision of guaranteed and statistical network QoS: Integrated (IntServ) and Differentiated (DiffServ) Services. Although offering distinct solutions, most QoS architectures share four objectives:

1. To control the network service such that the response to a specific service element is consistent and predictable.

2. To control the network service such that a service element is provided with a level of response equal to or above a guaranteed minimum.

3. To allow a service element to establish in advance the service response that can or will be obtained from the network.

4. To control the contention for network resources such that a higher priority service element is provided with a superior level of network resource.

These two approaches belong to two distinct families of signalling mechanisms, IntServ belonging to flow state based signalling and DiffServ being an example of a state-less approach.

The development of the Integrated Services concept is the responsibility of the IETF's IntServ Working Group. It starts with the premise that the Internet is composed of *"a heterogeneous set of routers, running different routing protocols and using different forwarding algorithms"*. The working group aims to *"define a minimal set of additional router requirements which ensure that the Internet can support the new service model"*. This accepts the continued existence of a connection-less network under the control of no single agency, and simply aims to extend the best effort delivery mechanism to provide a guaranteed and controlled load service. These two types of service require further definition. A guaranteed service (Shenker *et al.*, 1997) implies delivery of a data stream at a defined bandwidth and bounded delay, very similar to the QoS provision typically supported on ATM networks. The Controlled Load Service (Wroclawski, 1997) attempts to provide end-to-end traffic behaviour approximate to that received on an unloaded or lightly utilised best effort network.

Integrated Services is an abstract model for QoS provision, and assumes the definition of an OSI layer 3, QoS signalling mechanism (comparable to UNI/NNI ATM layer 2 signalling protocols). Although a number of protocols have been developed, the Resource Reservation Protocol (RSVP) (Braden *et al.*, 1997b), is the only IntServ QoS reservation protocol to achieve realisation in commercial IP routers.

Unfortunately, as noted in RFC2208 (Baker *et al.*, 1997), there still remain serious questions as to the scalability of per-flow based QoS reservation approaches and therefore large scale adoption in the public network appears unlikely. The resources required (computation and memory) for flow classification, and the maintenance of flow state increase in direct proportion to the number of supported streams, preventing deployment other than at the edge routers or in small private networks.

The alternative Differentiated Service (DiffServ) approach (Blake *et al.*, 1998) provides QoS by classifying traffic into a number of classes that are treated with different levels of priority. Unlike the soft-state RSVP approach, DiffServ has no per-flow signalling and no per-flow state for routers to maintain and hence should scale better and be more appropriate for installation in the public network core infrastructure. However, the service outcome is only approximate, and since there is no explicit negotiation between the application and the network, it is the responsibility of the application to respond if the service is not delivered.

The result of these two competing schemes is that there is a confused service model for QoS provision, there is no single approach that possesses both accuracy and scalability. Both approaches suffer from the contradiction between the demands of QoS guarantee and the Internet's best-effort routing schemes. Most routing schemes generate routes on a lowest metric (normally based on a function of the number of hops) routing algorithm, and routing may vary on a per-packet basis.

## 2.6 Summary

The aim of this chapter was to highlight the important factors influencing the provision of telecommunications over the last decade. This has been achieved first by observing three important trends: deregulation (section 2.0.1), advances in transport technologies (section 2.0.2) and distributed computation (section 2.0.3).

Bandwidths have grown from multiples of 64kbit/s ISDN channels at the start of the 1990s, through to speeds today in excess of 40 Gigabits per second using DWDM (Dense Wavelength Division Multiplexing). Increased capacity beyond 64kbit/s voice channels has turned attention to other types of electronic communication.

While these changes have influenced the telecommunication industry directly, advances in the Information Technology sector have indirectly impacted on the provision of communications. Section 2.1 examined multimedia services and the effect low cost multimedia terminals have had on the types of multimedia applications.

The chapter went on to explain the technologies that have resulted from these trends in the communications and IT industries. Three technologies of significance were highlighted, the Intelligent Network (section 2.2.1), Broadband-ISDN (section 2.2.2), and the Telecommunications Information Networking Architecture (TINA) (section 2.3). Of these three, the most ambitious, TINA was considered in greater detail, providing an overview of the enterprise and computing architectures. As will become evident in chapters 4 and 5, many of the goals of TINA and approaches selected have been influential to the Alpine framework described within the thesis. In common with the Alpine framework, the Network Resource Architecture (NRA) and Network Resource Information Model (NRIM) described in section 2.3.3 share the aim for third party control and management of network elements through a distributed middleware infrastructure.

However, as discussed in section 2.4, unfortunately after 8 years the consortium disbanded at the end of December 2000 with probably limited impact on the industry. The section discussed the many reasons behind the lack of TINA adoption, but argued that many of the ideas and principles have been quietly adopted in other fora, in particular the OMG.

Finally, in section 2.5, the emergence of the Internet as an alternative platform for communication services was introduced along with the challenges ahead with regard to future services including multicast and Quality of Service (QoS).

# Chapter 3

# The state of the art in distributed programmable networks

This chapter addresses the developments over the past couple of years within the field of network programmability. It builds upon the historical account of developments within the telecommunications marketplace discussed in chapter 2 and relates how the issues have been addressed by the research community.

The developments described here are crucial, since they highlight the important developments that have shaped current programmable and active network research and have influenced the work contained within this thesis.

## 3.1 The role of Network Programmability as a means for novel service deployment

As discussed earlier, the market for provision of information services is increasingly competitive; the future of a Service Provider hinges on the speed at which it can bring novel services to market compared to its competitors. This has resulted in new challenges both within the application service creation domain, and within the enabling technologies. Not only does the network need to be able to handle higher capacity but also to provide potentially radical new levels of transmission quality and network services. The solution put forward in this thesis is to engineer an open, extensible, highly configurable, programmable network infrastructure. It is argued that this will form a vital basis for future new service provisioning.

The thesis proposes that this environment can be met by separating the cell/packet forwarding fabric from the control software. Typically today, network operators and service providers do not have access to switch/router control software, because it is fully integrated with the hardware environment (e.g. Cisco proprietary IOS). Through the definition of a programmable network API, third parties could develop and deploy new services on network equipment. At present an operator or ISP wishing to introduce a new service must accept the slow time-cycle for persuading vendors to include the features within the next release of their control software. The effect this would have on software development should be akin to the effect that the UNIX and Win32 APIs have had on the software marketplace compared to old proprietary, horizontally integrated mainframe software.

There are two schools of thought as to how to enable network programmability, characterised by the Open Signalling (Opensig) and Active Networking communities respectively.

The Open Signalling community emerged out of the work at Columbia University on the Extended Reference Model (XRM) (Lazar *et al.*, 1994; Lazar *et al.*, 1995) and their resulting Xbind (Chan *et al.*, 1996) architecture, combined with activities in distributed technologies (e.g. the OMG and TINA) towards telecommunications. Initial emphasis was on developing open interfaces to ATM switches, a number of early attempts include VINCE (Vendor Independent Network Control Entity) (Naval Research Laboratory, 1995), Columbia University's Open API for ATM Switch Control, and the Ariel interface deriving from the work on Switchlets (Van der Merwe & Leslie, 1997; Van der Merwe *et al.*, 1998; Rooney, 1997) at the University of Cambridge. Very significant is the development of the General Switch Management Protocol (GSMP)(Doria *et al.*, 2000), a protocol to enable control of label switches (e.g. ATM or MPLS). Since 1999, an IETF working group has been established to standardise the development of the GSMP protocol. More recently the community has turned to consider the applicability of the open signalling approach to IP technologies.

The Active Networking community shares many of the same aims as Opensig, but has a more dynamic model allowing new services to be uploaded onto network elements at runtime. At its most extreme is the capsule (Tennenhouse *et al.*, 1997) approach, whereby every packet has the potential to contain as payload both data and active code. The model of service deployment is quite distinct from that of the Opensig community: new network services may be deployed in the network at the behest of user applications, simply by the user injecting packets containing active code into the network. Of course such an approach introduces serious reliability and security questions which are covered later in this chapter.

This thesis does not confine itself to one community, drawing upon the best of both approaches. From the Open Signalling viewpoint, the evolution of TINA inspired Connection Management schemes is considered, along with more recent control architectures from IEEE P.1520 and the Parlay Consortium. The inclusion of Active Networking technologies is more selective. Those approaches based upon capsules (i.e. packets whose payload contains both data and active code) are not included, since they are considered outside the scope of this thesis. Although there are many claimed advantages of these schemes, at present there is no proof that they scale well, can be made secure and reliable and are fit for deployment in public networks.

## 3.2 The ReTINA project

From the mid to late 1990's, TINA relied upon a collection of independently funded TINA "auxiliary" projects to validate, and to stabilise the TINA specifications. Many of these projects, including ReTINA and VITAL were funded by the European Union under the ACTS (Advanced Communications Technologies and Services) Programme.

The ReTINA project (AC048) is one such ACTS funded TINA "auxiliary project", a consortium of software vendors, equipment manufacturers, telecom operators, and academia. The project was launched in September 1995 and initially contained the following partners: Chorus Systems (now part of Sun), Siemens, HP, CSELT, France Telecom, British Telecom, Telenor, APM, O2, and Broadcom. Lancaster University joined in March 1996 to work on Quality of Service (QoS) issues, while GMD Fokus joined for the one-year project extension granted for 1999.

The goal of the project was to "develop and demonstrate an industrial strength open distributed programming environment (DPE) with support for distributed multimedia applications over emerging broadband networks". This embraces many areas of distributed systems research including end-system control, network management, DPE services and software engineering tools. Important results from the project were made with respect to the DPE, development of DPE services (notification service, persistent object service) and their impact upon OMG standardisation, and the validation and extension of the TINA CMA (Connection Management Architecture).

It is this last issue, the work of the third workpackage on TINA CMA validation, that is of relevance to this thesis. TINA CMA is the part of the Network Resource Architecture (NRA) (Steegmans & et al, 1997), responsible for the establishment and release of network flows. The ReTINA CMA is therefore an implementation only of the connection management features of TINA NRA, other features such as configuration, fault and performance management were beyond the scope of the project.

The ReTINA CM (Connection Manager) is based closely upon the TINA CMA specification, and adopts a similar architectural model. It was based upon an evolution of the ALCIN software package from Alcatel (Flores & van Landegem, 1996), the ReTINA CMA implementation constituting a DPE service itself, and forming part of the ReTINA DPE. The ReTINA CM is only a live implementation of the NML (Network Management Layer) and EML (Element Management Layer) layers; the CC (Connection Coordinator) and LNC (Layer Network Coordinator) components are merely dumb implementations. The network components (computational) and informational models are very similar to those of TINA, and are not redefined here (refer to chapter 2).

Beyond validation of the TINA CMA, a second aim was to extend the architecture to support the constraints implied by real-time telecommunications. This meant focusing on developing scalable real-time QoS control and negotiation mechanisms compliant with the spirit of the TINA NRA.

The large semantic gap for representation of QoS between the client applications of the ReTINA CM package, and the physical ATM infrastructure was bridged through the Generic Quality of Service (GQS) (Banfield & Oliver, 1997) specification. This was a three layer QoS model (figure 3.1), consisting of Application Service Classes, a generic traffic specification, and switch specific representations. The ReTINA CMA is independent of network technology and it is the responsibility of the implementation of the EML-CP (Element Management Layer Connection Performer) to map between generic and switch QoS representations. The generic traffic representation, as used for inter-component (within the NML layer) QoS representation is drawn from the notion of Flow Specifications as envisaged by Partridge (Partridge, 1992).

Amongst the interesting properties of GQS was a division between application driven, network provisioned, and non-functional parameters. The GQS contract distinguishes between parameters that are controlled by the source application, for example maximum transmission rate and burst size, and those that are the responsibility of the network (e.g. delay and error rate). Full details of the GQS parameters and their respective contract types are given in Table 3.1.

Since the implementation of the ReTINA CM package was engineered to control an ATM based network infrastructure, proof of the appropriate choice of GQS parameters was considered on their ability to map down onto the ATM Forum UNI3.1 traffic classes (Table 3.2) current at that time.

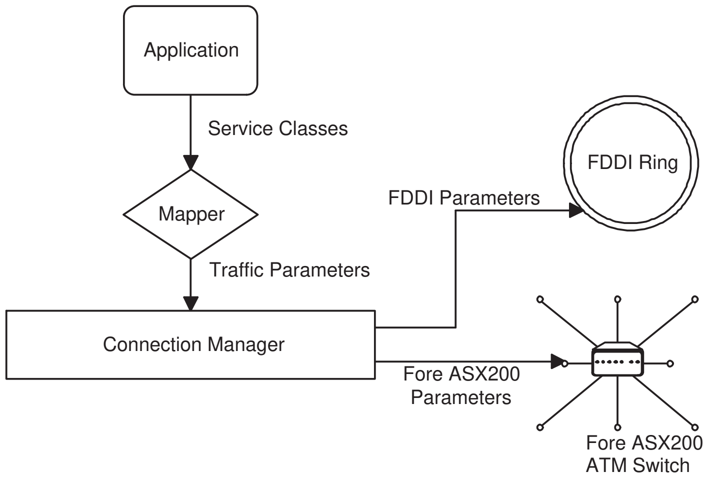


Figure 3.1: GQS, three layer QoS model

The ReTINA CMA extends the TINA CMA computational model with additional QoS components (Figure 3.2). It was noted that keeping the routing tables within each CP up-to-date in respect of both topology and QoS in real time was unrealistic. The aggregation of QoS status from lower level subnetworks is computationally expensive since it requires the calculation of all the optimal routes through this subnetwork in terms of use of the lower level child subnetworks. Instead, two representations of the routing table are held within a CP, a real-time accurate topological table and a QoS estimate table. Such a solution works well because QoS status is merely an estimate to aid higher-level route decomposition. Accurate QoS guarantees can be made at the switch (EML-CP) level and roll back could occur in the unlikely event of great inaccuracy in the high level QoS estimates. To achieve this the role of the Connection Performer (CP) is devolved into three functions: router, admission tester and QoS broker.

1. The Router functions as a traditional NML-CP component; it has the job of selecting possible routes from point to point through its given subnetwork. In the case of point to multipoint connections, a route is selected to support each branch of the connection. The router differs from the non-QoS aware design in that after a valid route has been found it must pass this on to the Admission Tester for evaluation of available QoS on that route. Only once the Admission Tester has passed the route may it be established.

2. The Admission Tester is the component responsible for establishing whether a given traffic flow over a given route can be admitted so that the network can provide the requested guarantees to the requesting application. Secondly, it must

| Application Driven | Network Provision | Non functional |
|---|---|---|
| Average Transmission Rate (ATR) | Minimum Transmission Rate (MnTR) | Guarantee Type (GT) |
| Maximum Transmission Rate (MxTr) | Maximum Delay (MD) | Security Required (SR) |
| Maximum Burst Size (MBS) | Maximum Delay Variation (MDV) | Maximum Cost (MC) |
| | Maximum Error Rate (MER) | |
| | Consecutive Bit Error Rate (CBER) | |

Table 3.1: GQS Parameters

| Traffic Class | MxTR | MnTR | ATR | MBS | MD | CBER |
|---|---|---|---|---|---|---|
| CBR | Yes | No | No | No | Optional | Optional |
| UBR | No | No | No | No | Optional | Optional |
| rt-VBR | Yes | No | Yes | Optional | Low Value | High Value |
| nrt-VBR | Yes | No | Yes | Optional | High Value | Low Value |
| ABR | Yes | Yes | No | No | Optional | Optional |

Table 3.2: Mapping of GQS to ATM UNI3.1 traffic classes

ensure that the new traffic will not break any of the existing QoS traffic contracts. The Admission Tester has no QoS status information of its own, being purely a QoS evaluation function. The proposed route is collapsed into its constituent parts on which the available QoS is checked with the broker. If the aggregation of the available QoS on the constituent parts of the route is greater than the required QoS then the route can be considered accepted.

3. The Broker is responsible for maintaining QoS capacity information of an associated resource (either a Subnetwork or Link). Initially only bandwidth, delay and error rate were supported but this is easily extensible. The Broker hierarchy follows the network hierarchy, and the Broker is also responsible for initiating the update of its parent broker if necessary.

The Router, QoS Evaluator and Broker run in the same process for efficiency but

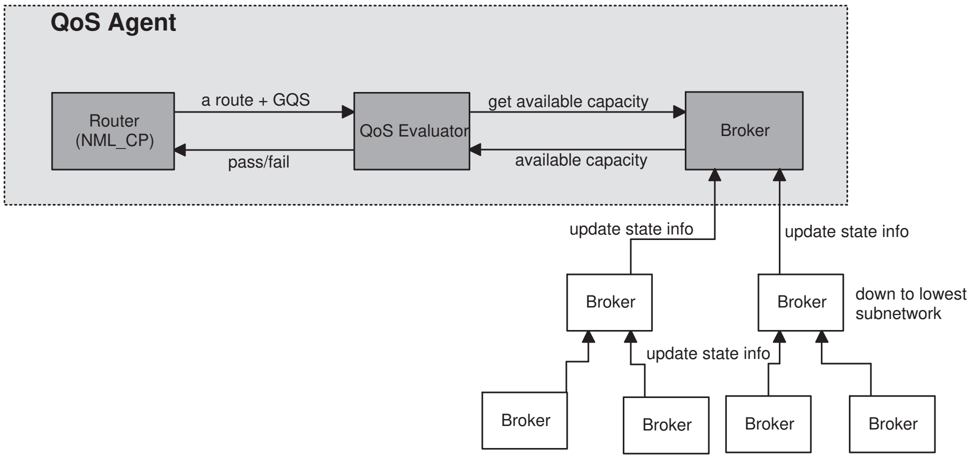


Figure 3.2: ReTINA Connection CP functional Architecture

the Broker runs in its own execution thread. This allows QoS estimate updates to occur as a low priority in the background but newly arrived connection requests have immediate access to the latest status figures.

An initial simplistic approach (Banfield & Hutchison, 1997; Oliver *et al.*, 1997; Oliver, 1997) co-located these functions within the same component, the xNML-CP (eXtended Network Management Layer Connection Performer). This enabled the routing and QoS brokering functions to share the same topological database, having the dual advantages of maintaining consistency of topological information, and simplifying configuration by keeping the same process as for the non-QoS extended Connection Manager. However, less than satisfactory performance (Banfield *et al.*, 1999) was found to result from the combination of routing and QoS estimation within the same component. The solution adopted by the final evolution of the ReTINA CMA was to separate the functions of routing and brokering into two separate components, which given the properties of access transparency provided by the ReTINA DPE could be located on separate nodes. Logically this was justified since the accuracy and time-scale of routing and QoS estimation are distinct. The routing functionality is invoked on receipt of a CM operation request and requiring a fast and accurate result. QoS brokerage is a continual process of inaccurate estimation, invoked synchronously on a given timecycle and asynchronously given emergency conditions (e.g. change of topology, QoS estimation failure). This enables the construction of an alternative hierarchy for QoS brokering from that of the CP hierarchy, and provides for the dynamic reconfiguration of brokering algorithms simply by redeploying components in the QoS broker component hierarchy. The final Connection Management Architecture was shown to scale well (Banfield *et al.*, 1999), not only was the impact of size of network negligible, but through replication of CPs, greater concurrency and reliability could theoretically be assured.

## 3.3 Integration of IP into the TINA Network Resource Architecture

As was noted in Chapter 2, one of the reasons for the lack of success of TINA is its association with ATM and other broadband telecommunication technologies and a perceived lack of relevance to the Internet. Although it was not an objective of TINA, it is fair to say that the current Network Resource Architecture (Steegmans & et al, 1997), associated Network Resource Information Model (Natarajan & et al, 1997), and ConS Connectivity Provider Reference Point (TINA-C, 1997c) consider only the abstraction of connection-oriented network technology.

With the adoption of the TINA Business Model as the major input into the OMG Open Service Marketplace model (Schenk *et al.*, 1999), the development of a connectionless Connectivity Provider architecture and reference point remains an important but unsolved challenge. The following two sections, outline two important recent activities within this domain.

### 3.3.1 Managing TINA Streams that use Internet Transport

The role of the Connectivity Provider is to offer through the ConS (TINA-C, 1997c) interface a DPE service, enabling others to configure end-to-end connectivity for transporting information between TINA applications. The different types of connectivity are abstracted to the concept of stream flow connections; a binding between two or more end points (stream flow end points). One or more stream flow connections together form the communication session enabling unitary session control. This notion naturally fits with a connection-oriented network model, but connection-less networks (e.g. IP based networks) are not addressed in current TINA specifications.

An initial approach (Steegmans *et al.*, 1997) to bring the Connectivity Provider business role and the NRA into line with IP technology was made by members of the core team as early as 1997. This proposed a solution to map TINA streams onto IP transport technology by extending NRA without changing the connectivity management interface (ConS), so that the service architecture could remain unaltered. This is done by taking advantage of the NRA modular network architecture, which naturally supports heterogeneous network technologies at the layer network abstraction. The layer network in the current TINA NRA is based upon the concepts defined in the ITU-T G.805 (ITU-T, 1995a) and M.3100 standards, which are only applicable to connection-oriented network modelling. The solution is to generalise layer network connectivity, an instance of which is defined as a trail, into a neither connection-less nor connection-oriented layer network binding (LNB). The LNB can be mapped directly to a trail for connection-oriented networks, but in the case of an IP network is simply the specification of a relationship between two or more Layer Network Access Points (LNAP). These LNAP map directly to IP transport layer network addresses, normally either UDP or TCP port addresses. Additional binding characteristics, QoS parameters, topology (point-to-point, point-to-multipoint) may be associated to a Layer Network Binding. For an IP network no attempt is made to define a further level of decomposition below the layer network. The IP routing and management infrastructure is maintained using existing internet protocols (RIP, ICMP, BGP etc.).

The solution is, however, incomplete. It has not been adopted as part of the NRA (indeed, the NRA has not been updated since this time). Furthermore, it does not address the issues of internet control and management, leaving this to current internet protocols. Instead, it provides a simple solution to map TINA based streaming technology to a connection-less network. Streams are by their very nature connection based, and although they have been shown to map to connection-less protocols (e.g. UDP/IP), these protocols are still being used in a session-based usage paradigm.

### 3.3.2 The IP Control and Management (IPCM) working group

A second, more ambitious, approach in the TINA-C is that of the IP Control and Management working group (IPCM). Created in April 1999, the group's main interest is in the development of a simplified TINA Network Resource Architecture and information model providing a generic delivery process of reusable components to provision of services over an IP network. The group is concerned solely with the Connectivity Provider business role, and although results are only likely to be achieved at the end of the TINA life expectancy, they will be highly relevant to other successor fora including the OMG Telecom Task Force.

Unlike the previous approach, it is accepted that it will be necessary to redefine the specifications of the ConS, LNFed, and CSLN reference points even though this may impact upon other layers of the TINA architecture. Indeed, the group has rewritten the approach to network federation replacing the LNFed peer-to-peer, and CSLN client/server federation with two new inter and intra-domain relations. This is more naturally compli-

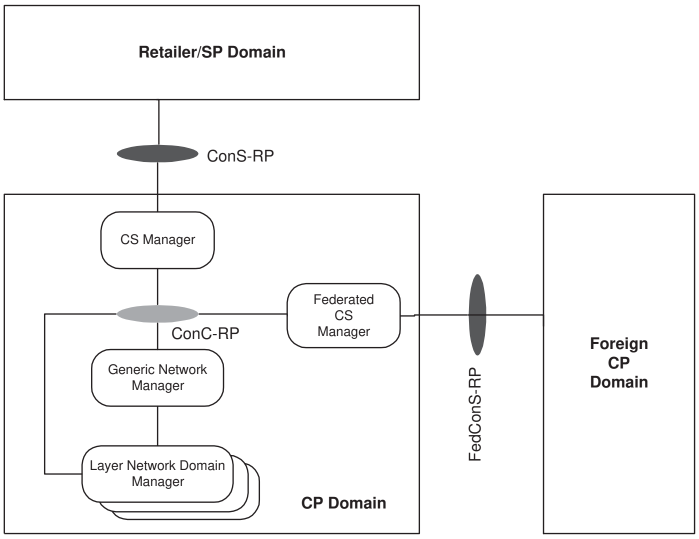


Figure 3.3: The TINA IPCM Reference Model

ant with the TINA business model, distinguishing between internal network federation and external agreements made to provide connectivity between separate business entities. These two new reference points (figure 3.3) are named Connectivity Connection (ConC-RP) and Federation Connectivity Service (FedConS-RP) respectively.

## 3.4 IEEE P.1520

The IEEE P.1520 initiative (Biswas & et al, 1998; Biswas & et al, 1999) formed in 1997 is a standardisation project aiming to develop a set of standard programming interfaces for communication networks (PIN) and network elements to enable rapid service creation and open signalling. Although it shares a similar goal to that of TINA, P.1520 instead emerged out of the Open Signalling research community and does not adopt the TINA model.

Central to this is the emergence of open interfaces enabling control and management to enable services to be introduced that could not otherwise be realised using today's proprietary network control software. This is achieved through the definition of programmable network interfaces, rather than specific algorithms or protocol semantics. This opens up the telecommunications software marketplace enabling the engineering of telecommunications services tailored to the customer domain.

A very simple business model forms the basis of the four-layer P.1520 Reference Model (figure 3.4). The traditional model for telecommunications provision is based

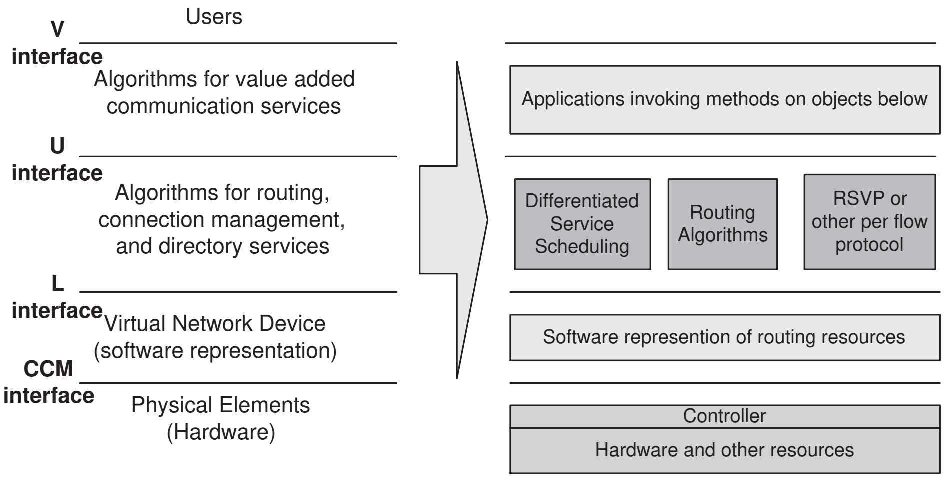


Figure 3.4: The P.1520 Reference Model

upon a vertical domain monopoly operated by a single telecommunications provider, and where inter-working between operators has been necessary (e.g. for international links), this has been achieved through bilateral peer-to-peer relationships. The P.1520 Reference model recognises the move towards the division of telecommunications into horizontal markets, consisting of multiple network operators and value-added service providers.

Starting from the network fabric, the lowest layer in the horizontal hierarchy is that of the Physical Element (PE), be it a switch, router or terminal. Immediately above in the hierarchy is the Virtual Network Device (VNDL), a logical representation of the state of the PE entities, and as such provides a vendor neutral but technology specific abstraction of physical resources.

While the PE and VNDL represent a one to one mapping with local physical resources, an abstract network wide view is formed at the Network Generic Services Level (NGSL). Here, algorithms are located that deal with the functioning of the network as a whole, for example, topological configuration, QoS management and routing functions. Finally at the top level plane is the Value-Added Service Level (VASL) where algorithms and policies reside provided by value-added service providers. It is with those value-added services that the end user applications interact directly.

### 3.4.1 The Programmable Network Interfaces

Each of the four entities identified is separated by four programmable network interfaces. These are the ‘V’, ‘U’, ‘L’ and ‘CCM’ interfaces, described below:

1. 'V'-interface: - user-level programming interface providing access to value-added services. This interface is highly personalised to end-user requirements, in general not dealing with the communications medium directly but accessing features to enhance the value of the communication experience. The interface is not necessarily an RPC based request/reply interaction nor defined in IDL. An example of an alternative 'V'-interface interaction could be the asynchronous posting of XML based policy specifications.

2. 'U'-interface: interface used by value-added applications to access/control network services. Commonly this would be requests to establish/release network connections, in the form of point-to-point or point-to-multipoint connectivity graphs. Since this is the interface to the NGSL (Network Generic Services Level), operations exposed over the 'U' interface are independent of algorithm, connection set-up procedure, or switch fabric of the underlying network.

3. 'L'-interface: the API to directly access and manipulate local network resource state. Resources exposed here are not technology independent (separate interfaces for ATM, MPLS, IP Differentiated Services Routers etc.) but should be vendor neutral.

4. 'CCM'-interface: is a collection of protocols that exchange state and control information between the device and external VNDL (Virtual Network Device Level) agent. In the ATM domain, the GSMP protocol resides at this point in the P.1520 Reference Model. Information passed over this interface is very low level, and may use proprietary protocols.

Within the IEEE P.1520 project, there are three working groups dedicated to the study of the applicability and definition of programmable interfaces for ATM, SS7, and IP networks. Both the ATM and IP working groups have made good progress, and the ATM group is ready to complete first standardisation of the lower level interfaces. To enable standardisation to continue in both groups at independent timescales, early in the spring of the year 2000, two new projects were created by the IEEE, that of P.1520.2 and P.1520.3. The former is dedicated to the definition of ATM, while the latter is aimed at IP based programmable interface standardisation.

### 3.4.2 Programmable Interfaces for IP Network Elements

The rapid evolution of the Internet is enabling new unforeseen services to be deployed, from eBusiness, to IP telephony. The rapid developments in application software in both server and end user terminals, through technology like Java and ActiveX, has not been matched by advances in the data communication network. Although many new protocols have emerged supporting features such a multicast and QoS guarantee, there is a slow time lag before these features are adopted. Indeed despite standardisation there are still no concrete plans for mass deployment of protocols such as IPv6 and RSVP into the public Internet. In order for the data network to keep pace with end-system developments, a mechanism must be developed to allow alternative policies, control algorithms, and signalling protocols to be rapidly deployed into the public network (Biswas *et al.*, 1998). This is the goal of the IEEE P.1520.3 project on open programmable network interfaces for IP network elements.

However, current IP routers are designed to do very fast routing of large numbers of packets using fixed pre-programmed control algorithms. In the main, they do not expose any form of session control as exhibited by ATM and other connection-oriented technologies. It is generally assumed that it is impractical to maintain per-flow state on the core network routers, which may handle gigabits of data per second. Instead, the aim of the P.1520.3 standardisation is to define interfaces to influence routing, and scheduling policies of network elements. Unfortunately, there are many competing approaches to IP transport and mechanisms to ensure better than best-effort QoS. These include Differentiated Services, Multi-Protocol Label Switching (MPLS), and the Resource Reservation Protocol (RSVP). It is therefore not possible to define a single P.1520 Reference Model 'L' interface to capture the diverse nature of IP network elements. Instead, a set of interfaces is being defined and later hopefully standardised to capture contemporary IP transport technologies and protocols. Inevitably, this set of 'L' interfaces will be extensible, and it must be a continual activity of P.1520.3 to keep them up-to-date. An example of possible 'L' interface abstractions and mapping onto low-level resources is given in figure 3.5.

Beyond the 'L' interface there has been no concrete interface definition to date. Typical operations likely to be found on IP network element 'L' interfaces include:

1. Adjustment of resource allocation. This could be per-flow packet processing for IntServ based network elements, or per service class for DiffServ based elements.

2. Dynamic routing - routing policy information can be configured into routing elements to implement new routing schemes based on customer preferences.

3. Traffic shaping - configuring of policies to re-shape packet profiles and drop packets.

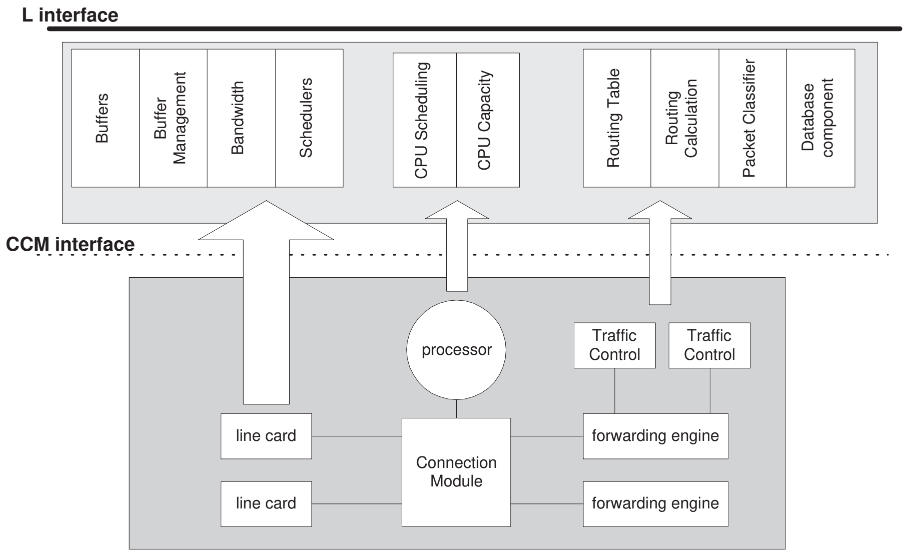


Figure 3.5: IP Router Resource Abstractions

4. Configuration of physical resources - the programmable interface enables the network operator to bring up and configure physical resources. A typical example is the configuring of IP addresses associated with a new line-card.

5. Monitoring - provide a much more sophisticated approach to monitoring of status of network elements than the ubiquitous MIB (Management Information Base) traversal of SNMP (Simple Network Management Protocol) (Case *et al.*, 1990; Case *et al.*, 1996).

### 3.4.3 The Broadband Kernel

The heart of the P.1520 software architecture is the broadband kernel, the underlying platform for broadband (Chan *et al.*, 1996) and mobile (Angin *et al.*, 1998) network programmability. Although similar in role to the DPE in TINA, the broadband kernel has evolved instead from the XRM binding model. The XRM (eXtended Reference Model) developed within the COMET group at Columbia University (New York, USA) comprises a communications architecture for networking and multimedia computing platforms. This is based upon the view that the management of the broadband network and multimedia applications are naturally related and must be brought under a common architecture for efficient provision. To achieve this the "RGB" decomposition of the XRM is introduced (Lazar, 1997), dividing service provision into three components (figure 3.6), the broadband network, the multimedia network, and the Services

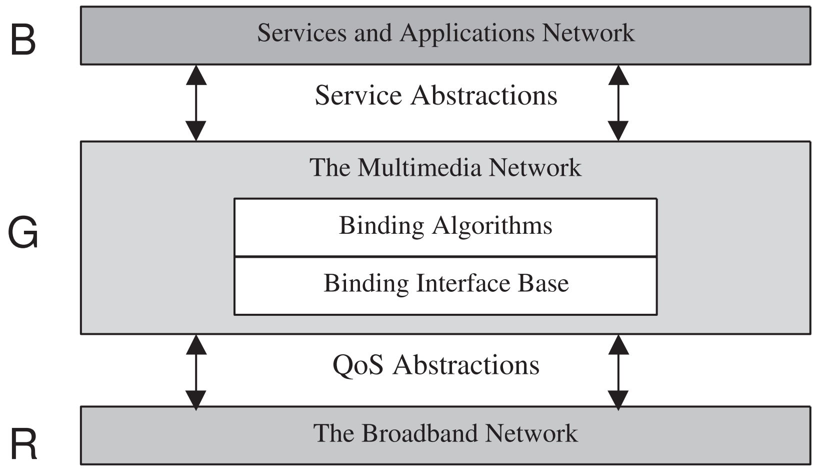


Figure 3.6: RGB decomposition of the XRM

and Application Network.

Upon the physical infrastructure ('R'-plane) resides the multimedia network whose function is to provide the middleware support needed to realise multimedia services. The primary purpose of this architecture is the provision of services requiring end-to-end QoS guarantees. The multimedia network provides a programming model that allows resource reservation through distributed state manipulation.

Rather than the hierarchical Management Information Base (MIB) of current management solutions, the broadband kernel incorporates a series of interfaces collectively entitled the Binding Interface Base (BIB) (Lazar *et al.*, 1994; Lazar *et al.*, 1995) (figure 3.6). The middleware BIB located at the 'G'-layer is subdivided into five further planes (for illustrative simplicity not shown in figure 3.6). The 'D'-plane is a repository of information where information about components and state in the network is held. Within the other four planes reside the algorithms and control interfaces to manipulate the state held in the 'D'-plane. The 'N' and 'M' planes contain the management and resource control functions respectively. Connection Management algorithms (interacting with the 'M'-plane routing functions) are found in the 'C'-plane, with finally the 'U' plane providing multimedia protocol abstractions.

Although similar to the system services provided by an operating system (OS) kernel, the broadband kernel is distributed across the entire network as opposed to conventional OSes that reside on a single host. A distribution mechanism is not defined, being the choice of the broadband kernel implementers to choose between the state-of-the-art distribution technologies available at the time of implementation. To date the Xbind (Chan *et al.*, 1996) kernel for control and management of ATM networks has been the only XRM inspired broadband kernel implementation. Xbind adopted CORBA as its distribution technology through the use of the Orbix ORB from Iona.

## 3.5 Parlay

Unfortunately, current telecommunications networks are a long way from the utopia of open programmable frameworks described so far. A less radical approach is emerging from the Parlay Group, an eleven company consortium, to design and promote the Parlay Application Programming Interface (API) to access the core capabilities of public telecommunications and data networks. Originally formed in March 1998 as a group of five companies (British Telecom, Microsoft, Nortel Networks, Siemens, and Ulticom) the impact of the group has been such that by May 1999 six further members had joined the consortium (AT&T, Cegetel, Cisco, Ericsson, IBM, and Lucent Technologies).

The group's sole aim (Parlay Group, 1998) is to define a set of interfaces, known as the Parlay API (figure 3.7), to enable network operators and independent software vendors to write applications that provide services over wireless, IP, and PSTN (Public Switch Telephone Network) networks. This should enable a new generation of dynamic telecommunications applications to be designed with customers' own private requirements, and created and maintained by the networks' customers themselves. The Parlay Group is not a standards body itself nor does it aim to provide commercial products or implementations of the interfaces themselves, instead hoping that successful interface definitions will be incorporated into relevant standards bodies.

The Parlay API is not designed to directly open up the networks' signalling for public usage, instead providing a platform to enable value-added service provision. It represents a move away from the operator controlled mass-market applications (e.g. the 0800 "freephone" service) where complex network engineering could be used for service creation due to economies of scale. Instead, the API makes visible those network features that are useful to enterprise value-added service applications, exposed in a manner that is technology independent while maintaining the integrity, performance and security of the network.

Since the Parlay API will be used by value-added service provider programmers lacking a background in telecommunications, the interfaces are clearly defined using standard software engineering tools. The resulting API can be characterised as, a multimedia enabled, extensible, object-oriented interface enabling secure access by value-added service

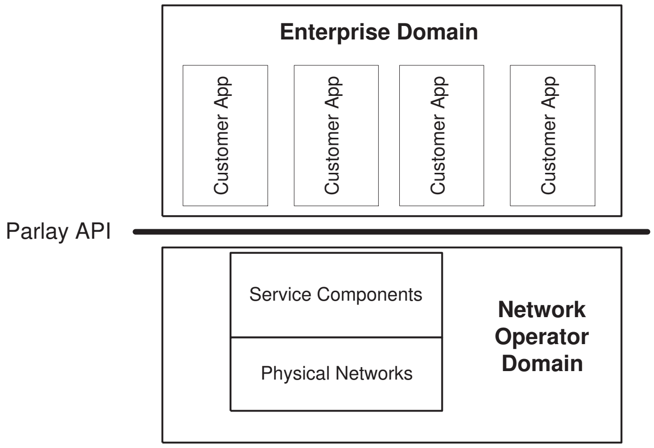


Figure 3.7: Parlay API Architecture

layer applications to the fundamental network functions. The interface definition has been achieved in a technology independent manner by adopting the Unified Modelling Language (UML) and both OMG and Microsoft dialects of IDL. This enables the API to be defined independently of distributed deployment technology, leaving the choice between today's COM, CORBA, and Java technologies until implementation.

The Parlay API can be subdivided into two further categories of interface:

- Service interfaces: offer access to a range of network capabilities and information.
- Framework interfaces: provide the supporting capabilities necessary for the service interfaces to be secure, resilient, locatable, and manageable.

The services are assumed to be co-located with the framework. They are considered trusted, such that once an application has been authorised to use particular services by the framework, then no further authorisation is required when those services are used. In some scenarios, however, additional authentication may be required for the application to use certain service features. Examples of service interfaces in the Parlay API v2 (Parlay Group, 2000) are:

- Generic and Enhanced Call Control Service
- Conference Call Control
- Multimedia Call Control and Conferencing Service
- INAPI Call Control Service

- Camel (Customised Applications for Mobile Network Enhanced Logic) Applications Part (CAP) Call Control
- Generic Messaging Service
- Generic User Interaction Service
- Connectivity Manager
- User Location Service
- Emergency User Location Service
- User status

The Parlay Group is not trying to compete with the work of other groups (IETF GSMP working group, MSF (Multiservice Switching Forum), SoftSwitch Consortium etc.) developing element control APIs. The Parlay service layer API should map down onto third party "Resource APIs" of network-based resources and physical equipment, including legacy protocols such as INAP and SS7. At present, off-the-shelf Parlay compliant telecommunications management applications are not available; output from the Parlay Group itself has been limited to the "Wake-up call" demonstrator. The recent release of the Parlay COM SDK (Software Development Kit), which, although unfortunately only supporting the Parlay version 1.2 interface, now provides a basis for further demonstrators and future commercial implementations.

## 3.6 Active Control Plane technology

The second group of state-of-the-art technology relevant to this thesis is in the newly emerging contentious field of Active Networking (Tennenhouse *et al.*, 1997). The term Active Networking is a widely (mis)used term, generating equal enthusiasm and mistrust from the research community. There are many different definitions but for the purpose of this thesis, it is defined as follows:

> "An Active Network (AN) is a communications network which contains computational capability enabling the execution of mobile code that adapts the communications network to provide novel specialised services".

Using this definition we can distinguish between open signalling programmable switch technologies, which enable third party connectivity control applications, and mobile code

AN systems which allow the dynamic downloading and movement of code to specific network nodes.

Network element control in AN is achieved by executing code in an Execution Environment (EE) rather than remote calls to a programmable interface. These Execution Environments create virtual machines where AN code can execute securely, unable to access resources to which permission has not been granted. These virtual machines are normally bound to specific high-level languages, either general purpose (e.g. Java) or specially design AN languages (e.g. O-Caml and PLAN (Hicks & Keromytis, 1999)). The AN code executes locally on the active nodes, as opposed to an open signalling solution, where algorithms may reside remotely on DPE workstations. However, unlike open signalling which uses DPE technology (e.g. CORBA) to expose language independent interfaces, active solutions are tied to those programming languages supported by the EE.

There are two broad categories of Active Networking technologies, the so-called *integrated* and *discrete* approaches. In the *integrated* approach, every transmitted packet (known as a capsule) potentially may contain both data and AN code. This code within the capsule may be executed at each active node, the result of which may influence the routing, forwarding of the packet or even the contents of the data contained within the packet. Generally the initiation of an active service is made by the end-user not the network operator or value-added services provider, since it is the end-user application that injects packets containing active code into the network.

The *discrete* approach separates active code out-of-band from the main data traffic. Mechanisms are provided to allow third parties (end-users or value-added service providers) to upload code to active nodes. It is assumed that this is done on an irregular occurrence, but allows for security and consistency checks that are more sophisticated than are possible with a stream of capsules containing active code.

There are undoubted advantages in the *integrated* approach within an Internet context. The IP protocol is itself a connection-less best effort protocol, and an important property of Internet routing is that there is no guarantee that the route will be the same for two consecutive packets bound for the same destination. Scatter routing schemes (Tateson, 2000) actually utilise this property to distribute link load by forwarding packets over routes other than the one with the lowest hop count. With the *discrete* approach, unless the active service is uploaded to all AN elements, and that every possible route contains at least one AN element, it cannot be guaranteed that the active service will operate on all packets in a stream. Unless route pinning is adopted within the Internet, this condition must be designed into all *discrete* active services. For example given a video transcoder (e.g. MPEG to H.261) AN service, the receiver must expect to receive a stream containing packets encoded in both MPEG and H.261 formats. These services can be termed "best-effort active services", highly consistent with the best-effort Internet model.

The *integrated* approach does not suffer from such a problem, since AN code is contained within the IP packet and will therefore be available to execute when the packet passes through AN nodes. However, this still assumes that there will be an active node on every possible route that a packet may traverse, an unlikely guarantee in a large heterogeneous inter-network. Similarly, this assumes that every packet must contain active code, this representing an overhead both in terms of extra bandwidth consumed for carriage of AN code, and time and processing power required to authenticate and invoke code on a per packet basis. Considering the very high data rates now required, the trend to achieve this has been to design simpler routers/switches (e.g. the move from X.25 to ATM removed unnecessary error checking) rather than to slow down packet forwarding with excessive computation. There is general distrust of the potential for end-users to both maliciously and accidentally send code that is detrimental or even damaging to the running of the core network. Solving the security and feature interaction problems is currently a "hot" research issue; current approaches of sandboxed execution environments and secure programming languages have failed to satisfy concerns of network operators.

Much of the early active network research has been funded by DARPA, and US centric. Amongst the early AN research are three influential AN platforms, ANTS, Switchware and the NetScript toolkit.

Developed at MIT, the ANTS active networking environment (Wetherall *et al.*, 1998) is an example of an *integrated* approach AN platform. Active code is transported between ANTS nodes in capsules injected on demand by the end user application. The ANTS platform is focused primarily on providing core functionality to enable active services, and relies upon the general purpose Java programming language and its associated security model. These core services include mechanisms for transporting mobile code, demand loading and caching of active code, and processing and forwarding APIs.

The DARPA funded Switchware (Gunter *et al.*, 1997) AN architecture developed at the University of Pennsylvania emphasises safety and security through type checking, and program verification. This draws upon language provability research; instead of using general purpose programming languages (e.g. Java) active code is written using the purpose-designed Camel (Rimy & Vouillon, 1998) and PLAN (Hicks & Keromytis,

1999) languages. The fundamental building block of any AN architecture is the active router infrastructure; for Switchware this is the Secure Active Network Environment (SANE). SANE provides the essential services for system integrity and the infrastructure for cryptographic code authentication.

The NetScript project (Yemini & da Silva, 1996) at Columbia University (New York, USA) has developed a programming language to enable the creation of Virtual Active Networks (VAN). The NetScript language includes constructs and abstractions that greatly simplify the design of traffic-handling software. These abstractions hide the heterogeneous details of networked systems. The language is characterised as being high-level and strongly typed and is designed specifically to operate on packet streams. The network is viewed as a collection of Virtual Network Engines (VNE) which are interconnected by a series of Virtual Links (VL). This may only be loosely coupled to the physical network; not all network nodes will support a VNE, while others may include multiple VNE environments. NetScript programs are known as agents, which are dynamically dispatched to and executed at remote nodes that support a VNE.

NetScript is useful for building packet stream filters, routers, packet analysers, multimedia stream processors and new private network management protocols, but can also be used to interwork with legacy control and management systems. Inter-working with the ubiquitous SNMP standard is seen as especially important. NetScript agents can be used to dynamically create new SNMP MIB to work with current SNMP based management applications and provide a backend to the programmable infrastructure.

There is general acceptance that no single AN solution is capable of realising the full set of requirements necessary to build a fully featured AN system. It is likely that AN enabled networks will consist of many different types of AN node, each tailored to its specific task. In a network of heterogeneous AN solutions, there will be many active programs in transit for many different types of active execution environment. The Active Network Encapsulation Protocol (ANEP) was developed by University of Pennsylvania and MIT to allow multiple active environments to co-exist on a node, and to provide a mechanism for the node to de-multiplex active packets to the correct execution environment. According to the ANEP draft specification (Braden *et al.*, 1997a), ANEP is designed to:

> "Specify a mechanism for encapsulating Active Network frames for transmission over different media. The suggested format allows use of an existing network infrastructure (such as IP (IETF, 1981a) or IPv6 (Deering & Hinden, 1995)) or transmission over the link layer. In order to support ongoing

research, the proposed mechanism is as generic and extensible as possible. This mechanism allows co-existence of different execution environments and proper de-multiplexing of received packets."

The protocol defines the format of the ANEP header, how this header is used is not defined. It could be included as an IP header extension or held in the IP packet payload. The format of the header is designed so that an active node upon receipt of a packet can quickly determine for which environment the packet is destined. Since in a network of heterogeneous active environments, many packets cannot be processed at a given node because the execution environment is not supported, minimal processing time is taken on evaluating packets for which the node cannot support. Additional functionality other than de-multiplexation is also part of the ANEP header; these include forwarding rules, integrity checksums, and authentication. The ANEP header design is extensible, allowing the later addition of new features, possibly a security mechanism.

ANEP has gained widespread acceptance through use as the active demultiplexing protocol on the ABone (Berson, 2000). The ABone (*A*ctive Networks Back*bone*) is a DARPA funded active network research testbed consisting of multiple active nodes and heterogenous execution environments distributed across the world.

### 3.6.1 Darwin

The Darwin Project (Chandra & et al., 1998) has developed a set of customisable resource management mechanisms enabling telecommunications service providers to tailor network resources to meet their customers needs. Since these resource management requirements are quite diverse, Darwin considers that it is not possible to solve them with a single solution. Instead, the Darwin project has developed a suite of resource management solutions (see figure 3.8), comprising, the Xena resource broker architecture, the Beagle signalling protocol, and active code entities known as delegates. The resource broker, Xena (Chandra *et al.*, 1998b), performs global allocation of resources based on application-specific metrics submitted by either the end-user applications or the value-added service providers. The functions of Xena are firstly resource discovery, and secondly given knowledge of available resources the assignment of optimal usage.

The Beagle signalling protocol (Chandra *et al.*, 1998a) provides a mapping between Xena's abstract view of the network and the actual physical resources. Beagles operates on virtual network meshes, unlike traditional signalling protocols that handle network connections. A virtual network is a list of flows and delegates together with details about how these are distributed within the mesh. Upon receipt of a resource reservation

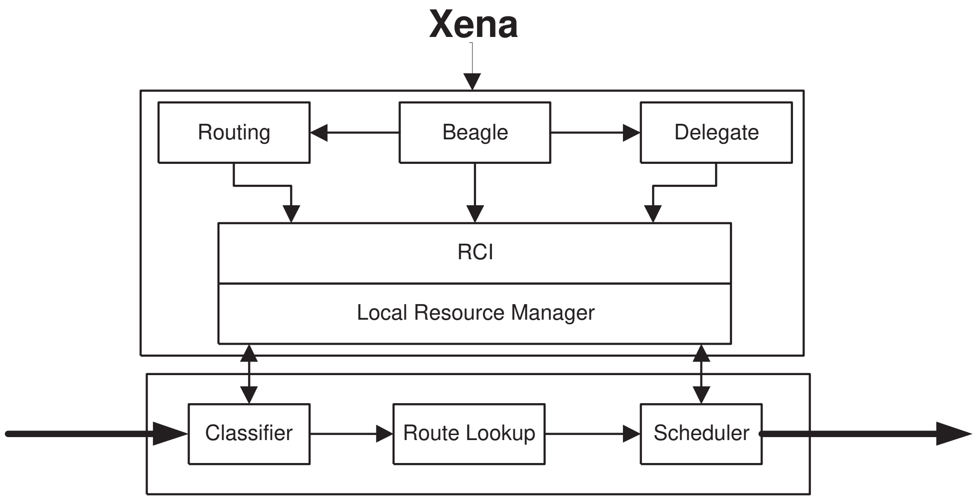


Figure 3.8: Darwin Node Software Architecture

request from Xena, a series of requests are made to the physical nodes, these instructions may include the uploading of new delegates onto network routers. This action introduces security constraints not considered by traditional signalling and is performed as a three step process, first the bytecode is submitted along with restrictions upon its resource requirements (including to which flow it may operate on). This is transported to the relevant nodes which must perform admission control to ascertain if they can supply the necessary resources required (runtime environment, CPU cycles, memory, etc.). Finally if the admission control stage is passed, then the delegate is executed, and resources allocated.

A delegate is an active code entity that is injected into the network and is executed on network elements to implement customised management of data flows. Unlike other Active Network architectures, delegates are a very focused application, and are restricted in functionality to traffic management. Typical functions delegates may perform include:

- Monitoring of network status
- Resource allocation
- Communication with other local and remote delegates
- Affect packet routing

Similar to the IETF QoS architecture for both Integrated and Differentiated services routers, at the heart of Darwin network routers is the packet scheduler. This is based upon a Hierarchical Fair Service Curve Scheduler model; each physical link is represented as the root node of a resource allocation tree. Leaf nodes representing actual data flow, and interior nodes flow aggregations. Resources allocated to the root node (i.e. a link) can be further allocated amongst a node's children, each of which can share that allocation within its subtree.

Delegates can influence packet forwarding through the Resource Control Interface (RCI). This consists of a series of functions that can be broadly categorised thus:

- Flow manipulation methods. Central to the resource control facilities exposed for use by delegates is the notion of flow. A flow is defined as those packets matching a flow specification consisting of a list of the contents of IP header fields. Upon the delegate invoking the `add_flow(flow_spec)` method, the supplied flow specification is used by the router's classifier to identify a flow and grant access for control of the flow to the delegate.

- Resource Manipulation. Once access has been granted to a flow, these functions allow the delegate to control scheduling of the flow forwarding. This is possible by enabling the delegate to insert/delete/modify a node representing a flow within a hierarchical scheduling tree.

- Flow re-directing methods. Enables the delegate to manipulate fields in the routing table and therefore control packet forwarding. Other functions include packet dropping and tunnelling options.

- Traffic monitoring and queue management. Used to monitor queue lengths and bandwidth data to gauge congestion.

- Delegate Communications. Use to set-up communication channels between both local and remote delegates. This enables delegates to react to global as well as local knowledge.

It has been impossible for the Darwin project to specify all possible user requirements on resource programmability functions that must be exposed. The RCI is therefore extensible beyond the five functionalities described above, and an example quoted in Darwin literature is recent support for Random Early Detection (RED) through additional functions to manipulate drop threshold variables.

Naturally, the introduction of router programmability through delegates introduces new safety and security concerns. These risks include both code safety (ability for delegate to accidentally or maliciously cause the outage of a network node) and privacy of third party traffic; these risks have been classified into two categories. The first category relates to security risks that are local to the router node, these include manipulation of traffic flows that the delegate is not responsible for, and interfering with scheduling of resources not belonging to a user. These concerns are solved in Darwin at runtime, upon the invoking of an RCI operation by a delegate, the node OS inspects access control lists to ascertain if the operation may legally be completed.

The second classification is the set of actions that can affect the global state of the network. The risks stem from the ability of the delegate to manipulate the routing tables and hence forwarding functions. Dangers arising include potential for denial-of-service of service attacks where flows are redirected to a target victim server (which becomes overloaded servicing incoming data) and congestion maliciously caused on critical links.

### 3.6.2 USC/ISI Active Reservation Protocol

The Active Reservation Protocol (ARP) project (Lindell, 2000) at the Information Sciences Institute (ISI) of the University of Southern California is developing a framework for deploying new control functions into the network. An active networking (AN) approach is used for the dynamic instantiation of new protocols but unlike other AN schemes, the ARP architecture is purely concerned with the control plane. Applications of ARP have been demonstrated for dynamically deploying the RSVP and RIP protocols.

Although IETF standardisation is considerably faster than that of the ITU, the time taken for protocol standardisation, inclusion in commercial grade routers and roll-out of services to market is considerable. The authors of the ARP architecture argue that their active control plane approach significantly reduces the need for protocol standardisation. Only a single implementation of a protocol is needed, the code of which in effect forms the defined standard.

The ARP project adopts a classical active node architecture (figure 3.9). Active Applications (AA) are executed within a secure Execution Environment (EE) sandbox. The underlying network node OS may support one or more EEs, each having its own levels of trust. The task performed by an Active Application may be further refined into activities; an RSVP AA handles many simultaneous sessions, while a routing algorithm will handle one stream per session.

A pragmatic approach is taken to the dynamic loading of active code, with the adoption of the out-of-band *discrete* approach. Active Application code to implement sophisticated protocols (e.g. the RSVP and RIP examples) is by necessity large, indeed too large to be fit in an active capsule, thereby ruling out the *integrated* AN approach.

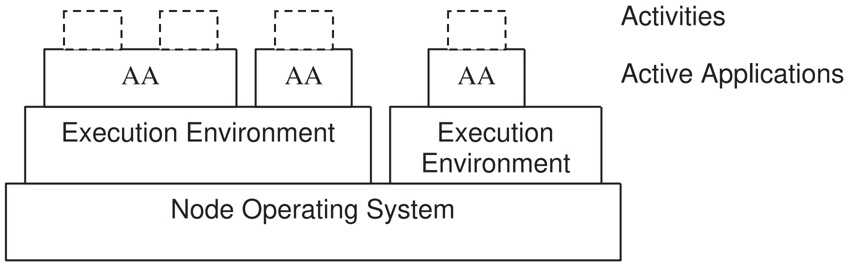


Figure 3.9: Active Reservation Protocol (ARP) Architecture

Instead, active packets may contain a reference to the location of active code, which can then later be fetched out-of-band from code libraries and caches within the network.

Active Applications for the ASP Execution Environment are written in the Java programming language. Java was chosen because it provides code portability across platforms (both processor families and Operating Systems), uses modern object-oriented terminology, allows dynamic class loading, and provides a flexible sandbox security model. The ARP EE extends the Java Virtual Machine with the Protocol Programming Interface (PPI). The PPI itself provides a safe interface to node operating system functions, and associated active support functions; these are defined in six categories:

- Process Management Interface - includes process and thread control functions.
- Soft-state container methods - provides data storage and timers for soft-state information.
- Network I/O - methods to allow AA to send and receive packets, including those not addressed specifically to the active node (i.e. promiscuously).
- User Application API - inter-process communication (IPC) mechanism between user application and ASP node. It allows user applications to send instructions directly to an ASP node.
- Interface to routing - allows AAs to both query and set entries in the routing table.
- Interface to traffic control - controls the active node packet classifier by defining flows and enforcing admission control. The identified flows can then be assigned priorities, and forwarding and processing resources reserved.

### 3.6.3 The Pronto Platform

There are many Active Network platforms currently under development, each specialised to provide a specific AN service model, and each functionally incomplete. The Pronto platform (Hjálmtýsson, 2000), developed at AT&T Laboratories (New Jersey, USA), is claimed to provide a commercial strength platform that may be used by the AN research community for experimental AN research. It is hoped that this removes the requirement for each institution to build its own research platform, and concentrate research on finding the so-called “killer Active Network application”.

The Pronto design recognised that in addition to the enhanced AN features, much of an Active Node OS functionality is identical to any standard OS. Indeed many potential AN applications (e.g. the active web proxy) require efficient and stable file system, transport protocol, memory, process, and thread support.

Those AN platforms that have been built from the ground up lack reliable well-maintained standard OS features and toolkits. Instead, the Pronto platform is built around the well-known Linux operating system. This enables the platform to take advantage of the well-maintained file, process, memory and thread support, and the wealth of freely available compilers, debuggers, toolkits, and libraries. The platform can develop in parallel to the Linux operating system, and still take advantage of enhancements made to the standard OS.

The debate as to where active programmability should reside, efficiently in the kernel, or in user-space, is resolved through a compromise by placing AN support functions in the kernel for performance, and giving a user level library providing an API to access these functions. The architecture consists of three major components, the generic kernel layer forwarding engine, the user-space Execution Environment (EE), and finally the interface between kernel and user-space.

The resulting implementation is highly modular, consisting of a number of small changes to the Linux kernel, presented in a patch file for post version 2.2 kernels. Since Pronto is built on a standard commodity OS, a standard Java Virtual Machine (JVM) can be adopted as an EE, although Pronto is EE neutral, and others could easily be applied. The Pronto router does not impose an AN programmable model, and supports four models that interact differently with the data-plane:

1. Programmable control-plane mode - in this mode active applications are only given access to control-plane functionality, and are not able to observe the data-plane itself. Active applications executing within this mode are the most efficient (i.e.

use the least processor and memory resources) and pose the least security risk.

2. Data-plane peeking mode - active applications are able to spy on packets and queues in the data-plane but cannot actually alter those packets. This provides finer grained control plane programmability, and can be used by AN applications to tailor policies to network congestion and types of packet.

3. Local multicast mode - packets are both processed by the normal data-plane forwarding engine and multicast to relevant active applications running in local execution environments. Although relatively computationally expensive compared to the two previous models, this is the first mode that allows the active application to alter packet contents. However, normal packet forwarding is not prevented, packets are forwarded by the standard process. Active applications offer merely a service enhancement, injecting additional modified versions of packets.

4. Programmable data-plane mode - in the preceding three service models, the execution of the active application is asynchronous with respect to the data forwarding function. In this mode, active applications execute directly upon every packet in the data-plane. While this mode is computationally very expensive (since the active application must process every packet) and represents the greatest security risk, it is required for applications that transform packet data (e.g. multimedia transcoders)

## 3.7 Summary

This chapter has introduced and put into context the most recent developments in programmable network technology. This is initially described from the perspective of the limitations of the closed network systems of today, combined with the perceived market driven advantages of migrating to an open approach.

At the core of this chapter is the division of programmable technologies into two schools of thought: "open signalling" and "active networking". This distinction is not drawn arbitrarily: they have emerged from different backgrounds over different timescales.

The discussion of the open signalling approach follows from the discussion in the previous chapter of TINA by exploring the recent activities in that community towards open network element control. One of the complaints levelled at TINA (as discussed in chapter 2) is that there has been a lack of reference implementations and experience gained prototyping. A recent TINA "auxiliary" project, ReTINA (section 3.2) is introduced, which has extended the open programmable network model with support for QoS and has put considerable emphasis on the validation of their proposed solutions through implementation and experimentation.

However, since this thesis considers applicability of programmable technology to the Internet and other IP based networks, the recent and currently on-going developments to apply TINA technology to IP are considered. Much of the work has been at a very high level, mapping the World Wide Web usage paradigm with the TINA service session model, an area beyond the scope of this thesis. Two particular approaches of relevance are introduced in section 3.3.1 and 3.3.2, first the work undertaken by the core team on mapping TINA streams to Internet transport, and second the on-going efforts of the IPCM working group.

There are, however, many open signalling technologies other than the TINA Connection Management Architecture (CMA). In Chapter 2 it has already been stated that it is unlikely anyone will deploy a network described as being TINA compliant. Instead, many of the central ideas have been introduced into newly emerging open signalling architectures, two of which (IEEE P.1520 and Parlay) have been discussed in this chapter.

In section 3.6 the second approach to programmable networks, that of active networking, is discussed. Active Networks (AN) are a new and currently extremely controversial technology. Since AN is a broad term, space is given to explaining the various types of active network, and to reference some of the early influential active platforms developed in the USA. General active network technology is not the topic of this thesis, and the use of active networking is limited to what is termed the "active control plane". With this in mind a number of active platforms, Darwin, ARP, and Pronto are described, each relevant and complementary to the Alpine framework discussed in the following chapters.

# Chapter 4

# Alpine - A Lightweight Programmable InterNet Environment

The Alpine control and management framework presented in this thesis provides a lightweight model for provisioning of IP network services. It is aimed at the network operator and value-added service providers' businesses, providing an open and transparent framework for competitive provision of telecommunications services.

## 4.1 Purpose of control and management architecture for IP networks

Communications networks (e.g. telephony and telegraphy networks) were developed for the provision of commercial communications services. The provisioning model is based on a network operator having full control of the system, allowing the communication service to be provided in a secure, reliable and authenticable manner.

The Internet Protocol (IP) and the resulting public Internet has emerged from a distinctly different non-commercial background of military and educational usage. At the heart of the Internet is the concept that the Internet is not a single network but a federation of multiple network domains. Each domain is autonomous, and may be owned and administered separately and may deploy different data-link layer technologies.

The success of the Internet has in part been due to the simple and ubiquitous IP protocol. Inter-networking between these autonomous systems is achieved through IP layer network peering, and the running of inter-domain control-plane routing protocols.

There are therefore many independent organisations involved in the provision of communication services over the Internet. This property makes it very difficult to introduce new network services into the Internet because of the end-to-end requirements inherent in many potential services.

The roll-out of a multicast IP service on the public Internet is a classic example of how the introduction of new services is being delayed. Currently, multicast traffic is supported by most modern routers, and many ISPs run multicast over their own network domains. However, despite the underlying technology being now widely deployed, there are still no serious wide-scale roll-out plans.

A similar impasse exists in the provision of Quality of Service (QoS) through a Differentiated Services mechanism. The technology for multiple priority queues is now built into many modern routers, and is already deployed in selective domains. However, unless all links of a route support packet prioritisation, the benefits may be lost end-to-end. Even given that all links in a path support Differentiated Services queuing, there is still the need for a mechanism to mark packets accordingly for each network domain.

Even within single operator domains, such as corporate Intranets, it is difficult to introduce new services. Within IP networks, the network control and management functions that are traditionally under the control of the network operator are to be found embedded within the IP network control architecture. Network layer (layer-3) equipment is a closed environment, giving the operator little control over their devices.

The IETF (Internet Engineering Task Force) is responsible for the standardisation of IP architecture and protocols. However, it is dominated by equipment vendors and US educational institutions, which has resulted in an explosion of ad hoc standards over the last few years. The comparative lack of influence by network operators (Byström, 1999) means that no cohesive fundamental architecture exists to enable the fitting together of increasingly complex protocols and standards.

From the pure management perspective, device management has been chiefly through proprietary protocols/interfaces or the ubiquitous SNMP (Simple Network Management Protocol). Network management has concentrated solely on the intra-domain, a solution obligated by the competitive business relationships between carriers, and the lack of scalability attributed to the SNMP protocol. In order for the provision of extra-domain services, it will be essential to develop scaleable secure multi-domain management approaches.

The Alpine (A Lightweight Programmable InterNet Environment) framework, the subject of this and subsequent chapters, is tailored towards network operator initiated service provision. This differs distinctly from user initiated services, the more common goal of Active Network solutions/architectures. From the perspective of the network operator, an efficient method of initiating new commercially oriented network services is essential for their survival. Since deregulation, the former PTTs face a competitive marketplace for communication service provision, resulting in lower profitability from their core transmission business. Many of these network operators are publicly listed companies, and must look to other business opportunities for raising revenue essential to maintain share price and dividend levels to which share-holders have become accustomed.

The provision of novel network services enables an operator to differentiate its services amongst the homogenous competition, and thereby raise extra revenue from these services. This demand has resulted in operators funding many new research initiatives within the area of active and programmable network technology. The EURESCOM funded Caspian project (Banfield, 2000), and the British Telecom University Research Initiative (BT-URI) both fund research projects that have provided impetus behind the Alpine framework discussed here.

## 4.2 Alpine Framework Model

The aspiration in the development of the Alpine framework has been to provide a mechanism enabling the provision of new advanced services within IP based networks. Achieving this requires management of the following issues:

- Physical resources: links, line cards, virtual paths/circuits, switching capacity, and storage capacity require configuration and co-ordination.

- Reservation: in the standard Internet, services are shared in a best effort manner amongst competing users. Where value-added services are provisioned, it is necessary to reserve resources so that they are for the sole use of the purchasing organisation.

- Scheduling: both user requirements and network performance alter by time of day and day of the week. A service provisioning mechanism must be able to plan ahead, responding and mediating between competing demands.

The Active Network (AN) technology (as previously described in chapter 3) is a promising approach to provision of advanced network services in IP networks. However, the technology has not yet reached maturity; there are still many unsolved issues. Amongst the network operators there is natural distrust of AN technology because of the potential risks associated with allowing end-users access to their shared core network infrastructure.

The Alpine framework takes a more pragmatic approach to IP services provision, fusing together the state-of-the-art AN technology with established distributed telecommunications management practices. This could provide a lead into further opening up of internal network control functions, if and when AN technology matures.

As has been alluded to earlier, it is important to understand that Alpine is a framework to support the dynamic provision of novel network services and not a full heavyweight telecommunications architecture. Although adopting practices from traditional telecommunications architectures, Alpine differs distinctly from previous programmable architectures such as TINA and Xbind. Alpine concentrates on providing a lightweight and flexible scheme for rapid deployment of network services on a connection-less IP infrastructure, which are tailored to customers unique requirements. By contrast, traditional telecommunications architectures as typified by TINA focus on providing a distributed object oriented environment for service deployment on broadband (e.g. ATM) telecommunications networks.

Central to Alpine is the view that the provision of a single programmable model to all business usage domains is senseless and illogical. It is inevitable that the type of programmable abstraction most applicable for use by end users is different from that of value-added service providers, trusted peer networks, or indeed the network owners themselves. Network control functions should be provisioned as programmable abstractions differentiated to specific business roles (defined in section 3.4). There are two possible approaches to this, one deriving from the AN research community, the other a product of traditional telecommunication management. The first solution is to provide multiple Execution Environments (EE) per node, each with its own security model applicable for its intended user. The second solution is to develop a hierarchy of programmable abstractions, where new network functions can be inserted at an appropriate point in the hierarchy consistent with the user's view of the network.

It is this last approach that forms the basis for the Alpine framework. As an example, the IEEE P.1520 Reference Model programmable interface hierarchy is adopted for consideration in Alpine. This enables different abstract views of resources to be projected, each appropriate to a particular organisation's position in the Reference Model hierarchy. This is not possible with the alternative of having specifically tailored Execution Environments on each Active Node, whereby each EE can merely constrain the view of that particular node's resources.

It is only the Reference Model that is adopted from P.1520, since this forms a good basis for discussion of the framework outlined in this thesis. No attempt is made to dictate (or prescribe) the actual instantiation of interfaces at these reference points since,
a) at present these interfaces are incomplete, considering only the 'L' interface;
b) interfaces should be tailored to the service demands of the customer.

### 4.2.1 Customer tailored interface definition

Traditional network service provision has divided management functionality into 5 distinct entities, known collectively as FCAPS (Fault, Configuration, Accounting, Performance, and Security) (ITU-T, 1992b). These are considered and often provisioned independently, and all management functionality is distinct from that of control (call set-up/release, route and bandwidth control). The rationale for the separation into the two classes of control and management is that control needs to operate in real-time while management operates over a slower time-scale. This separation is best highlighted in the current telephone network, where control is provided by the SS7 and IN signalling protocols/infrastructure, whereas network management is based upon a TMN architecture. A completely separate support system is used for the SS7/IN and TMN software systems.

Even the Internet makes this distinction, with control signalling through protocols such as RIP, OSPF, BPF, and RSVP, and management chiefly through SNMP. Interworking between these two functions is restricted to the very limited features exposed through the SNMP MIBs, generally only allowing the reading of attributes and insufficient influence of control plane functions. Such a division is inappropriate and disregarded for Alpine, since it is obvious that control functionality (e.g. routing and bandwidth control) must closely inter-operate with management functions such as faults, configuration and performance. Instead of the control and FCAPS separation, extensible components are specified which are similar to abstract classes and have no attributable meaning. These components are extended to provide the set of control and management functionality necessary to meet requirements of the particular customer's business domain.

### 4.2.2 Adopted Component Frameworks

The Alpine framework does not attempt to completely redefine component models, distributed systems and network control techniques, preferring to adopt the state-of-the-art in middleware technologies. Central to Alpine is the freedom to adopt the control technologies and interfaces appropriate to the customer's requirements, be that a COM instance of the Parlay API, a CORBA instance of TINA ConS, or an interface to a customer's proprietary system.

1. *ISO/ITU-T Reference Model For Open Distributed Processing (RM-ODP)*
   The Reference Model for Open Distributed Processing (RM-ODP) (ITU-T, 1995b) is an international standard for distributed systems modelling from the ISO and ITU-T. Rather than define an "on-the-wire" protocol standard for distributed communications, RM-ODP defines an architectural framework.
   Systems are inherently very complex with many entities and interactions contained within. Central to RM-ODP is the definition of five viewpoints to capture all aspects of a system; these viewpoints are, enterprise, information, computational, engineering and technology. Each viewpoint is not a complete analysis of the system, merely a projection of the system from one perspective. However by dividing the system up into five viewpoints (enterprise, computational, information, engineering, and technology) it is asserted that the properties of the whole system may be captured. This thesis adopts an approach based on multiple viewpoints inspired by RM-ODP to describe the Alpine framework. In this chapter the enterprise (described here as business requirements) and information viewpoints are discussed.

2. *OMG Interface Definition Language (IDL)*
   The CORBA Interface Definition Language (IDL) (OMG, 2000c; OMG, 1997a) is an OMG (Object Management Group) standard to define interfaces exposed by objects. There are several other IDL dialects including those from DCE (Distributed Computing Environment) and Microsoft's COM (Component Object Model).
   IDL allows a designer to define an interface independently of its implementation. Standard mappings from IDL now exist for the most popular programming languages, including C++, Java, Ada, and Smalltalk. Compilers exist to translate the language independent IDL into interface definitions in these general purpose programming languages.
   As well as allowing the decisions as to appropriate implementation to be made independent of the design, an important side effect from writing interfaces in a separate language is it encourages separate thinking about interface design. This discipline allows system interaction to be captured using conceptually coherent

interfaces.

3. *UML*

   The Unified Modelling Language (UML) (OMG, 1997b) is a graphical language for visualising, specifying, constructing and documenting traits of a software system. Originally developed outside of the OMG, it was offered to the OMG Analysis and Design Task Force in answer to their RFP (Request for Proposal) for a modelling language. Since November 1997, UML has been adopted by the OMG, and is currently undergoing its fourth minor revision (UML 1.4).

   Like RM-ODP, UML does not attempt to capture the whole system from a single viewpoint. Instead, a number of visual representations of a software system have been developed comprising classes, usage case, interaction, component, and deployment.

   Where diagrammatic notation is used to describe elements of the Alpine framework, UML is used as a common graphical design language for this thesis.

4. *IEEE P.1520*

   The IEEE P.1520 initiative (Biswas & et al, 1998) formed in 1997 is a standardisation project aiming to develop a set of standard programming interfaces for networks (PIN) and network elements to enable rapid service creation and open signalling.

   The P.1520 Reference Model (see section 3.4) recognises the division of telecommunications into horizontal markets, consisting of network operators and value-added service providers. This division into functional layers forms a central part of the justification for the Alpine framework described here.

   Interaction between layers is via three programmable interfaces (named 'V', 'U', and 'L'), and it is an aim of the P.1520 activity that each will be standardised. After two and a half years of P.1520 activity, only the 'L' has been considered in any detail.

   As such, the P.1520 Reference Model is useful in a similar manner to the OSI 7-layer protocol stack Reference Model. The networking research community regularly uses the model to provide a common language for discussion and understanding, while the actual OSI 7-layer protocol stack itself has never been widely adopted.

## 4.3 Business Requirements

To understand the computational and engineering decisions taken in the design of the Alpine framework, it is necessary to examine the business scenario from which it emerged. The most important tenet of the current Internet marketplace is the multitude of business roles serviced by many independent suppliers.

Two state-of-the-art business models for service provision, one from the OMG, the other from EURESCOM P926, have been considered. The first the Open Service Marketplace (Schenk *et al.*, 1999), currently under development within the OMG Telecom Domain Task Force, derives very strongly from the TINA business model (TINA-C, 1997a). It does not specifically consider whether services are actively deployed nor is it tied to an Internet scenario. It recognises the multiparty approach to telecommunications service provision, defining five roles: consumer, broker, retailer, third party service provider, and connectivity provider. Such is the similarity to the business roles defined by TINA that the details of the roles are not detailed further in this chapter, and are discussed earlier in section 2.3.1.

While these roles capture the functions performed in the provision of abstract telecommunications services, their mapping onto the Internet marketplace is non-trivial. In particular the Connectivity Provider, and the associated ConS, CSLN, and LNFed reference points, do not capture the Internet model of IP layer connectivity provision.

The second business model is taken from the EURESCOM funded "Caspian" P926 project, and is designed to discover all service management actions in the provision of what is termed the "my own network" service. The provision of the "my own network", a form of virtual private network (VPN), is the central purpose of active network technology in Caspian. From the highest perspective, three roles are defined[1]: consumer, owner (of "my own network"), and provider. The provider role (figure 4.1) can be further refined into the base connectivity provider and a number of actual service providers (at present security, quality of service, mobility, and group communications are defined). While well founded, there are a number of drawbacks not tackled by the model. First it does not fully capture the end-to-end nature of active services, the target is communication service provision from the perspective of the consumer. These active service providers do not produce a consumable service, merely transforming the communication properties. Those other agents involved in the communication session are excluded from the "my own network" model.

[1] developed by KPN Research as part of activity in EURESCOM P926

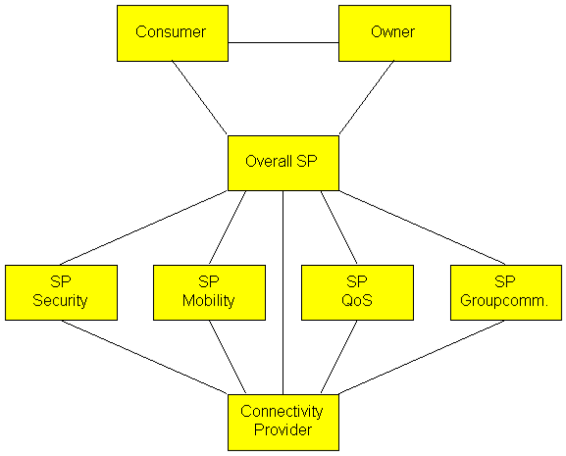


Figure 4.1: The active "my own network" business model

Similar to the OMG's Open Service Marketplace, the notion of the Connectivity Provider fails to represent the current model of IP layer service provision. Active Services are considered extra value-added features and do not enjoy a peer relationship with IP connectivity provision.

The Alpine business model defined for this thesis differs from these other models in that it concentrates on representing the current model of Internet provision rather than an abstract service deployment model. Figure 4.2 shows the current understanding of business roles and relationships between the actors involved. These roles are defined as follows:

- Client: the actor consuming communication services (i.e. end user).

- Customer: the agent responsible for the purchase of and paying for the Internet communication service. For example, Lancaster University as an institution performs this business role.

- Broker: an agent responsible for the location of Internet services. These services include what is termed White and Yellow pages services, and are currently performed by the Internet Search Engines (e.g. Yahoo, Excite, Alta Vista).

- Computation Provider: early World Wide Web models were based on a simple client/server role model, with the functions of the Computational and Content Provider being the responsibility of the server role. Increasingly the hosting of WWW servers and other forms of computation is handled by a third party.

- Content Provider: the agent responsible for the provision of the content that is

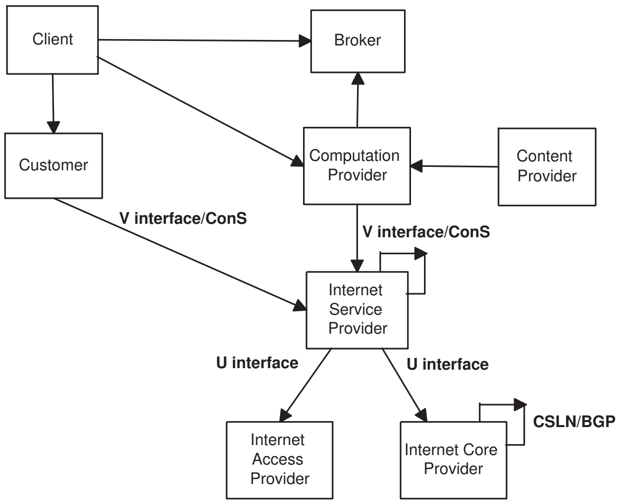


Figure 4.2: The Alpine business model with IEEE P.1520/TINA interfaces

being provided. Content is not merely multimedia served electronically over the Internet but could be physical goods dispatched as a result of the end-user interacting with an eCommerce service hosted by the computation provider business role.

- Internet Service Provider (ISP): located in the value-added service layer the ISP is responsible for providing the end-to-end connectivity service to the end-user. Included within the remit of the ISP is end-user support, in this way providing a middleman between the underlying physical network providers. Early ISPs provided little value-added service, instead concentrating on the Internet Access Provider business role. Modern ISPs are often virtual entities, constructed over infrastructures owned by others, and specialising in providing value over the communications service (e.g. family oriented ISPs blocking access to non-child-friendly sites).

- Internet Access Provider: responsible for the provision of the access network used to connect the end-user to the core Internet.

- Internet Core Provider: the public Internet is formed from a peering federation of core networks provided by agents responsible for the Internet Core Provider business role.

However, the Internet is a fast moving field, and new derivations of these roles are constantly emerging. An important facet of this market division is the partitioning into roles according to common functions. A stakeholder providing Internet services is not limited to a single business role; British Telecom for example combines the ISP, Access provider, and Core Network provider business roles.

It is important to note that the model does not distinguish a separate role for active and programmable network service providers. This is not considered since it is a technological deployment aspect and not a distinct role in itself. Active Network (AN) technology may reside with the Internet Service Provider, Access and Core Network Service Providers. However, AN is but one technology that may be used by stakeholders enacting these roles. At present it has not been possible to identify services that cannot be introduced through alternative means, the role of AN is simply to enable rapid, flexible, low cost service rollout.

One might question how this business model co-exists with the IEEE P.1520 reference model. The P.1520 RM differs by being a representation of programmable abstractions between devices and algorithms located in separate business domains. Both the 'U' and 'V' reference points for programmable interfaces are represented in the Alpine business model. However, the 'L' interface is an abstraction of the physical network devices, and has no consistent representation.

## 4.4 Element Programmability

Programmable network solutions may be considered as belonging to one of two broad categories, network layer and overlay programmability. Current IP layer appliances (e.g. routers) are closed environments, giving little potential for programming beyond limited configuration. An alternative solution is to leave the IP (layer-3) layer intact and to provide programmability in an overlay network superimposed over the non-programmable infrastructure. This has been shown to work (Fry & Ghosh, 1999) for adding active and programmable services to existing overlay technologies including the World Wide Web (WWW) and electronic mail (Aranda Gutiérrez *et al.*, 2000).

However, such overlay solutions are application specific and present an alternative set of requirements to layer-3 programmability. Table 4.1[2] summarises the characteristics of both approaches. Although it is potentially extremely easy to introduce new services using overlay technology, the severe performance limitations constrain its potential. Taking as an example WWW, the introduction of caching services has only proved feasible within the local domain and has not yet been made to scale to the core network,

[2]based on table adopted from (Juhola *et al.*, 1999)

| | Layer-3 programmability | Overlay programmability |
|---|---|---|
| Utility | High | High |
| Complexity | Moderate | Moderate |
| Manageability | High | High |
| Scalability | Moderate | Moderate |
| Performance | Potentially high | Low |
| Flexibility | Moderate - restricted due to risk | High |
| Applicability | High | Limited - only overlay networks e.g. WWW and email |
| Risks | High | Low |
| Availability | 5-10 years | 18 months |
| Integration with existing networks | Moderate - must change router software | High - very easy |

Table 4.1: Active Network Programmability Comparison

where caching would be required at gigabits per second.

Layer-3 element programmability promises greater performance and is more generally applicable. Traditional management of IP appliances has been through "open-loop" interaction using the ubiquitous SNMP (Simple Network Management Protocol) and proprietary protocol standards.

Control and management intelligence has been located off-board of the IP device (normally a general-purpose workstation running as an Operation Support System). In the case of the SNMP protocol, interaction with the switch has been on a master/slave relationship, the off-board SNMP agent walking a static MIB (Management Information Database), making "get" and "set" requests on variables in the MIB.

Even given an extended off-board programmable abstraction, this "open-loop" approach has inherent limitations. Additional bandwidth must be consumed by the programmable abstraction, which given that signalling and data channels share the same capacity detracts from the bandwidth available for the core data transmission service. "Open-loop" solutions including SNMP suffer from the so called "Catch 22" condition when connectivity has failed but the management agents are unable to achieve communications with the devices in order to restore connectivity.

It is intrinsically difficult for off-board management agents to retain an up-to-date view of device state. Constant device polling by the management agent is obviously not a scalable option, so inevitably the agent must accept a degree of historical inaccuracy. Fetching state on inspection by the off-board agent is an alternative, but is only applicable for applications that perform very limited state manipulation or monitoring. The trap handling approach practised by SNMP of sending event notification (given state passing beyond a given threshold) is the only approach shown to scale but is not suitable for a powerful programmable abstraction.

The approach followed in the Alpine framework is to adopt a "closed-loop" interaction mechanism whereby intelligence is located locally on the managed device. The solution is to create a programmable façade over the IP infrastructure, the Alpine framework targets this programmability at the P.1520 'L' interface reference point.

Control and management intelligence can be placed by an agency other than the equipment manufacturer (e.g. the network operator) dynamically onto the device. Interaction between this intelligence and the device is by an internal device loop, there is no need to consume signalling bandwidth between the device manager and the managed device.

Both approaches outlined have been concerned solely with the control and management operations resident in the control-plane. Most contemporary approaches to Active Networking (AN) consider active processing in the data-plane; this allows total reprogramming of device function. Obviously, the efficiency of this active code has a severe impact on the performance of the device. There has been considerable effort within the active network research community to engineer optimal AN Execution Environments (EE) based on kernel layer virtual machines (Cardoe *et al.*, 1999) and zero-copy kernel to userspace network I/O (Schmid, 2000).

Research into the requirements of layer-3 routers, both programmable and closed solutions, shows these designs to be inadequate. While software based routers (often based on Linux and BSD Unix) are regularly to be found in small local area workgroups such as those within Universities and Research Laboratories, they are increasingly being replaced with hardware based solutions for public backbone networks. The gigabit per second line rates and the increasing demand for real-time packet scheduling have forced manufacturers to adopt solutions based on optimised network CPUs and dedicated ASICs (Application Specific Integrated Circuits). Hence, the scope for programmability is limited to those features exposed by the underlying hardware.

The Alpine framework recognises this engineering reality, and concentrates on the

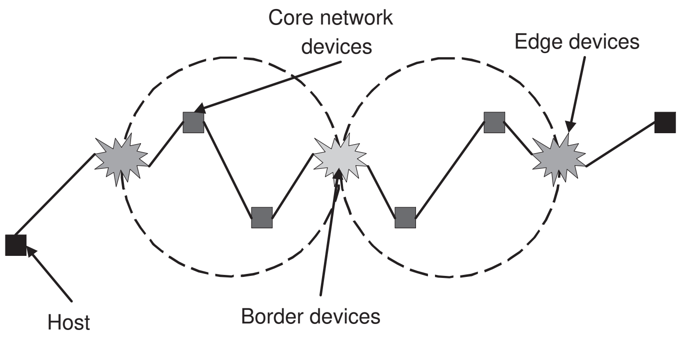


Figure 4.3: Potential points of programmability in Internet architecture

development of a control-plane active programming environment. This is not to disregard the possible advantages to be gained from data-plane active programming, which may be applicable in local area and intranet domains. Further consideration of the impact of router design on active programmability including a specific case study on a commercial series of Gigabit Ethernet routers is included in chapter 7.

### 4.4.1 Deployment

When designing an approach to active networking it is important to understand where in the network service programmability will be deployed. Many approaches are inadequate because they fail to recognise the different programmability requirements posed by different network domains.

There are many different types of IP network deployment, from the public Internet to large scale corporate Intranet WANs (Wide Area Network) to small workgroup scale IP LANs (Local Area Networks). Further consideration is given to the public global Internet IP network since this offers the richest and most diverse characteristics.

Fundamentally there are four places (figure 4.3) where programmability can be located in an Internet architecture, at the host (network), edge devices, border routers, and core switching fabric. Each of these is considered in detail in the following paragraphs. The first point of potential service deployment is the "host", be it an end terminal or a private network within the domain of the customer. Programmability enabling service deployment is well established within this domain, end terminals based on Personal Computers (PCs) are easily programmed to provide end system services. Conventional active networking nodes may be deployed within this private network where capacity is currently in the order of up to 100 Megabits per second. However, the future move from

10baseT,and 100baseT Ethernet LAN technology to Gigabit Ethernet may change this scenario.

It is the next three points of service deployment, located in the public network that are relevant to the Alpine framework from its operator perspective. These are examined in the order of controversy, from the low risk edge into the high-risk core.

The first point of programmability in the public network is the edge device located in the access network where it meets the core. Typically, bandwidth handled at the edge device is low (e.g. the connection from Lancaster University Campus to Net NorthWest MAN in August 2000 carries up to 155 Mbits per second). The edge device represents the point of least risk for introducing programmability, since a communication outage caused either maliciously or accidentally as a result of programming edge devices is less likely to impact upon other customers of the public infrastructure.

The second programmable location in the public infrastructure is the border router devices that inter-connect the autonomous systems from which the Internet is constructed. Outage of a border device due to incorrect programming is potentially serious since it can affect innocent network users. The Internet could potentially become segmented into disconnected islands, however such is the interconnectedness of the current infrastructure than any outage should only be temporary until an alternative route can be found. Placing programmability at the border device introduces new security fears since the ability to influence/inspect packets must be isolated to those belonging to the customer.

The final and most controversial point of programmability is the core network device. These are highly optimised devices which switch/route data at backplane speeds of many gigabits per second. The potential for programmability here is extremely limited, increasingly even the fundamental layer-3 (IP packet) forwarding processing is overweight. IP layer routing decisions are being taken at the border routers and cut-through layer-2 (MAC) routing is deployed internally. With the advent of optical switching technology, data rates of many terabits per second are possible without involving the IP layer.

The Alpine framework targets the introduction of programmability at the private and access networks and potentially at the borders between autonomous domains. These seem to be the most appropriate locations to introduce active services without impacting upon the overall performance of the network core. This is not to disregard the potential for future programmability at core network data rates, for example FPGAs (Field Programmable Gate Arrays) and other future technologies promise high-performance hardware programmability.

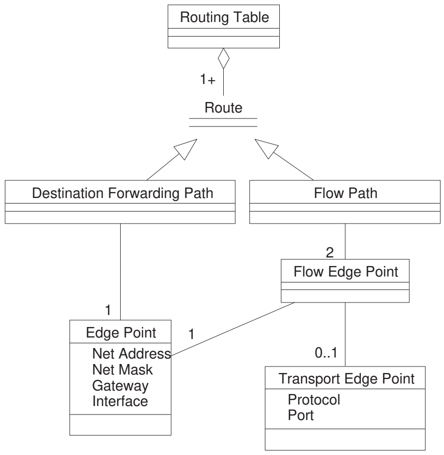


Figure 4.4: IP Routing Element Information Model

## 4.5 Information Model

Thus far, consideration has been given to what in RM-ODP would be considered the enterprise and computational viewpoints. To define programmable abstractions tailored to customers' viewpoints, it is necessary, together with the definition of the programmable API, to define a view of network resources. Such a perspective would be termed by RM-ODP an information model.

At the Virtual Network Device Level (VNDL), the programmable abstraction is primarily concerned with route control. There is already a well established route specification model used in current Internet routing protocols. However the route information passed is concerned solely with forwarding path to destination, and cannot differentiate routes upon other criteria, for example source addresses. This is increasingly inappropriate since for provision of per flow Quality of Service it is necessary to discriminate between packets belonging to different flows for resource allocation.

Instead, a more sophisticated model (figure 4.4) is introduced in this thesis, where a route may consist of a traditional destination-forwarding path (termed Destination Forwarding Tunnel), or a Flow Path. A Flow Path is able to differentiate resource to packets on a micro-flow level based upon the aggregation of network and transport edge point addresses. At the Network Generic Services Level (NGSL), where the network management functions reside, it is necessary to construct a global resource model. In traditional telecommunications management the network abstraction supported in the network management plane is termed the layer network. A layer network denotes a network based upon a single technology, for instance an ATM VP layer network, an

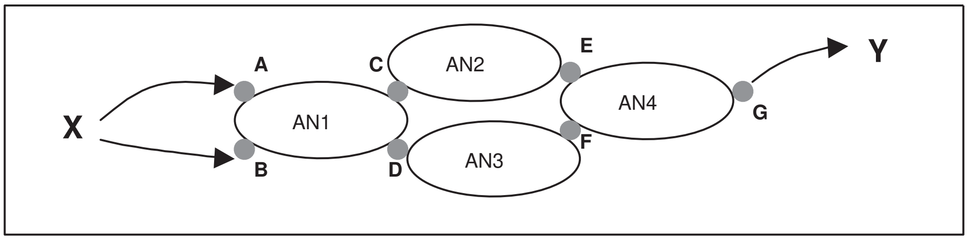


Figure 4.5: Autonomous subnetwork peering

ATM VC layer network, or a frame relay network. It was originally defined in the ITU-T Recommendation G.803 (ITU-T, 1992a) for SDH networks, but was later adopted as a generic model in ITU-T Recommendation G.805 (ITU-T, 1995a). Unfortunately, the trail (a subnetwork connection) concept is not applicable to connection-less networks such as the Internet and a new model must be sought.

It is a fundamental property of connection-less inter-networking that packets are routed on a per-packet basis, and that it is possible that each packet may follow a different route, and indeed may be received out of order if at all. To model the path taken by a flow of packets from a source to a given sink, it is therefore not possible to simply specify an ordered list of traversed elements. Instead, when modelling a flow in a connection-less network, all that can be quantified is the set of all possible nodes that a packet may traverse. Any given packet will only traverse a subset of nodes in the full traversal set. This can be illustrated using the topology depicted in figure 4.5. Given this topology, packets of a stream emanating from point X (off the diagram) may enter the domain via two possible entry points A and B. A flow from X to Y can therefore traverse the following set:

`[X - Y]->[{X,A,C,E,G,Y},{X,A,D,F,G,Y},{X,B,C,E,G,Y},{X,B,D,F,G,Y}]`

The impact of this on provision of active service is immense. If the value added by the active service is to be applied to every packet then the following condition must be true:

> "To apply a value-added active service to a stream so that at least one packet of that stream is actively processed, instantiations of that service must reside at least once on every possible path between the stream source and the possibly numerous stream sink(s)."

Given a network with a fully interconnected mesh topology the number of nodes a service must be deployed on may be considerable. Fortunately, current Internet topologies are not fully connected meshes, following the topological model described in section

4.4.1. Conventionally a host is connected to the public Internet by a single access network, over which all packets must traverse. Deploying active services in the access network is an effective way to guarantee active processing. This combines with the advantage already noted in section 4.4.1 that introducing programmability on edge devices installed in the access network causes the least risk since erroneous actions are unlikely to impact upon others.

## 4.6 Summary

This chapter considered the introduction of new services into IP networks from the perspective of a network operator. Alpine (A Lightweight Programmable InterNet Environment) is introduced, a framework for introducing novel services based over programmable and active network technology. Alpine is defined from a business perspective; the consequent middleware platform and subsequent implementation are covered in chapters 5 and 6 respectively.

The chapter started by contemplating potential for service deployment in the current Internet, and highlighted the lack of roll-out of multicast and QoS network services. Sections 4.2 and 4.2.1 then went on to define the requirements that have shaped the Alpine framework. However, Alpine does not itself define new distributed software engineering technologies/techniques, instead building on well-established practices and toolsets. Four of these distributed systems tools, namely RM-ODP, OMG IDL, UML, and IEEE P.1520, have been highlighted as being of special relevance. These were chosen because they were engineering agnostic, leaving the choice of programming language, object model and microprocessor platform until implementation.

The importance of a business scenario to provide a secure grounding for the design was introduced section 4.3. Since the goal of programmable networks and Alpine is for rapid novel service deployment, two state-of-the-art services models were introduced. Firstly the "Open Service Marketplace", currently under development by the OMG, and secondly the "My Own Network" active service model developed by KPN Research in the EURESCOM P926 Caspian project. While useful, these models are highly generic and map uneasily to the current Internet scenario. A further Alpine business model was introduced representing a snapshot of the current Internet services marketplace, and tying this into the interfaces defined by IEEE P.1520 and TINA NRA.

Sections 4.4 and 4.4.1 considered the appropriate places and types of network programmability that may be exposed. The failings of conventional off-board "open-loop" element management as typified by SNMP were explained, along with the proposal to move to a "closed-loop" interaction model where intelligence and layer-3 functionality are co-located and interface via an internal device loop. The potential and consequent dangers associated with deployment of layer-3 programmability were discussed with respect to four points: host (network), edge devices, border routers, and the core switching fabric. Linked to this, the penultimate section (section 4.5) reflected on the difficulties of modelling connection-less traffic, which finally impact on the locations where active services might be deployed if they are to provide useful value.

# Chapter 5

# An Active DPE Architecture

The realisation of a new telecommunications service is not simply the implementation and execution of application components but a full service lifecycle. This should be encompassed by a full set of tools for requirements capture, service design, implementation, deployment, execution, and service withdrawal.

The initial stage of a service lifecycle, that of requirements capture, is beyond the scope of this thesis, and it is assumed current methods are adopted. Chapter 4 has already introduced the basis of the Alpine framework, and its impact on service design. However, the major emphasis of this thesis is on the design of a middleware infrastructure enabling the final stages of the service lifecycle, that of implementation, deployment, execution, and withdrawal to be completed.

This chapter and the subsequent realisation and evaluation chapters therefore contains the bulk of the contribution made in this thesis. It starts from an historical perspective, detailing the role of signalling in current networks, and re-examining the advantages to be gained replacing diverse signalling protocols with a uniform Distributed Processing Environment (DPE). Sections 5.2 through to 5.5 describe the Alpine middleware infrastructure founded on programmable node technology. Finally, the role of signalling in the Alpine environment is examined, resulting in a proposed extension to the P.1520 programmable reference model.

## 5.1 Role of signalling in Network Management

The correct operation of signalling is vital to the performance of the data transmission network itself. The demands of signalling are quite different from those of multimedia data transfer and the correct partitioning of resources between signalling and data is an essential part of the network provisioning procedure.

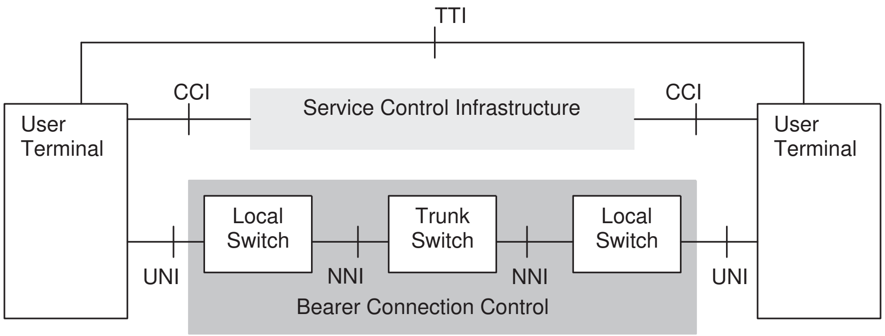


Figure 5.1: Decomposed Signalling Model

The introduction of advanced telecommunications services has extended the once simple call control model through the addition of non-circuit-related signalling. Examples of this include the interrogation of central databases for freephone and calling card IN services. Mobile networking, especially the sophisticated GSM (Groupe Spéciale Mobile) cellular networks, uses very advanced signalling for updating the cellular radio home location register (HLR). It is obvious, therefore, that signalling provision is at the heart of the deployment of advanced telecommunications services and requires further study.

### 5.1.1 Definition of the role of signalling

Signalling in all its many guises is a hidden but central part of all modern telecommunications networks. There are many different definitions of the term signalling, but for the purposes of this thesis signalling is defined as:

> "The system which provides the ability to transfer information between customers, within networks, and between customers and networks"

.

Conventional narrowband signalling was designed to enable the establishment and release of monomedia (voice) calls carried over an ISDN 64kbit/s channel circuit switched network. The emergence of more elaborate multimedia, multipeer services demands greater sophistication within the signalling system. To achieve this the monolithic approach to signalling is decomposed into service call control, and bearer connection control functions (Figure 5.1). This signalling decomposition is applicable to both B-ISDN and IP networking. The applicability to ATM as an example of B-ISDN technology is obvious, but a little more thought must be given in respect to the Internet.

Naïvely, it has been wrongly asserted that the Internet is a signalling-free network because the IP protocol is connection-less. However, signalling has always been a component of IP inter-networking, and is something that is increasingly important for the introduction of multimedia session oriented Internet services.

- Terminal to Terminal Interface (TTI) signalling is commonplace in the Internet, the most well known example of TTI signalling being the Transmission Control Protocol (TCP), which enables receiver oriented data transmission rate control, retransmission on the event of erroneous receipt.

- The Call Control Interface (CCI) enables communication between the user terminal and the network support infrastructure. A classic example of this type of signalling is domain name service (DNS) lookup, a service allowing translation between user-friendly service addresses to unintelligible numeric IP addresses. Other typical functions include multiparty session organisation, and terminal capability negotiation; the new IETF Session Initiation Protocol (SIP) (Handley *et al.*, 2000) and ITU-T H.323 multimedia protocol suite both embody these properties of Call Control signalling.

- The User to Network Interface (UNI) negotiates the bearer service between the Terminal Equipment and the local exchange, switch or router. In an Internet context, UNI signalling is embodied by network layer protocols, for example, the ICMP (Postel, 1981) and RSVP (Braden *et al.*, 1997b) protocols.

- The Network to Network Interface (NNI) encompasses all inter-network node signalling both intra and inter subnetwork domain. In addition to the ICMP and RSVP (also part of the UNI), signalling protocols at the NNI are generally route exchange based (e.g. RIP, OSPF, and BGP).

Generally each signalling functionality, be it service control, bearer control, or management, has its own signalling infrastructure. This may include separate signalling stacks, network addresses, security models, and in worst cases separate physical signalling networks. An example of this division can be found in the public telecommunication infrastructure of today, with the use of the SS7 protocol suite for call control, and network management through a CMIP based TMN platform.

Obviously, the cost of deployment and maintenance of multiple signalling functionalities is high, and it has long been a goal to bring all signalling under a single infrastructure.

Many modern signalling architectures (e.g. Xbind, and TINA) bring together all signalling under the umbrella of a Distributed Processing Environment (DPE) middleware communications platform. The main benefits the use of a DPE should yield are:

- In theory, lower infrastructure deployment and management cost because of the need only to purchase and operate a single middleware signalling platform.
- Interoperability between signalling functions: previously functions resident in separate signalling infrastructures could not inter-work, and services requiring the collaboration of such functions could not be realised.
- Lower cost of service creation due to access to modern software engineering methodologies and toolsets. This enables faster software implementation, component reuse, and greater software component reliability.
- Faster standardisation due to ease of defining new component interfaces and interaction models using object-oriented software design processes.

Current commercial switching and routing network appliances are not DPE elements, and do not provide computational capability. Examples of DPE based Telecommunications Management Systems (e.g. Xbind and the ReTINA Connection Manager) therefore place these functions offboard of the switches and routers, and often on workstation based OSS (Operations Support Systems).

This approach has the drawback of creating a large amount of signalling, both intra-DPE node and between DPE nodes and the switching appliances. Deploying functions off-board means that they can respond less quickly to changes in traffic conditions; at best it takes two round-trip times between the switch/router and the off-board DPE based management system. Off-board functions are forced to operate with incomplete or out-of-date information, and data must be transferred from switch/router to management platform either synchronously by systematic polling, or asynchronously on the occurrence of a trap event.

The approach followed by Alpine is to introduce the switch/router into the DPE infrastructure as a full DPE node. This enables functions that are driven by local resources to be performed locally, while those functions that need a global view can still be executed on powerful off-board workstations. The dynamic nature of this Active Network based middleware infrastructure permits the location of the deployment of a function to be under the full control of the management platform, as opposed to statically placed at the time of system configuration.

### 5.1.2 Signalling Characteristics

Earlier in this chapter, it has been established that there is a multitude of categories of signalling each with its own characteristics. Given the objective to move to a single middleware signalling infrastructure it is first important to understand the different requirements of signalling and data transmission network functionality prior to provision; these are now discussed according to seven distinct requirement categories.

- Bandwidth - it is obvious that signalling should be engineered to consume as limited bandwidth as possible so to minimise cost of provision of the signalling channel, and to maximise potential end-to-end data transmission bandwidth. A typical example of this is the ISDN 'D' channel typically used for signalling; this provides 16kbit/s which can be compared with the ISDN 'B' main data channel bandwidth of 64kbit/s.

- Delay - the delay characteristics of signalling information are difficult to capture since they are unique to each specific domain. However, while real-time multimedia data itself has very tight delay requirements this may not necessarily be true of the associated signalling. We routinely accept the several second connection establishment time associated with the set-up of a telephone call whereas delay of greater than 500 milliseconds to the voice channel yields unacceptable levels of service (ITU-T, 1998).

- Error rate - although it is well understood that multimedia data has a high tolerance for data transmission errors, the same cannot be attributed to associated signalling information. Where a high degree of errors occur in the network, error recovery techniques must be deployed at the transport level (OSI layer 4) through protocols such as TCP in IP networks and the SCCP and Intermediate Service Part protocols in SS7.

- Scalability - modern telecommunication systems are large, highly distributed, and span multiple transport technologies, and therefore scalability is a mandatory requirement. Some signalling functionality (such as accounting and billing) may need to scale larger than any single transport technology.

- Priority - a guaranteed level in respect to contention for communications resources is essential to maintain the correct behaviour of the network. For example, transmission of RSVP is potentially unreliable, as there is only a weak separation between control and data. This separation comes from a base recommendation that a minimum network bandwidth be configured for the processing of RSVP messages (Braden *et al.*, 1997b). Loss of RSVP signalling messages can result in a failure to provide QoS guarantees to the multimedia data transmission.

- Fault Tolerance - the job of signalling is to establish and maintain network connections; given the failure of a portion of the network the signalling must still be able to operate to repair the service for the lost data connections. Fault tolerance can be observed from two viewpoints; firstly to resist failure due to the rapid loss of resources (so called resource outage). Secondly, to resist failure due to erroneous interaction caused by faulty or incorrect equipment connected to the network. Total fault tolerance is impossible to achieve under conditions such as the unavoidable overloading of the signalling network. When such conditions occur, failure of the signalling network should be performed gracefully, for example dropping low priority messages so as to continue the essential network services.

- Federation - data transmission paths may span multiple networks possibly provided by different network operators. Where applicable the networks will need to provide channels and protocols for inter domain signalling. An example of this is the B-ICI specification (ATM Forum, 1995), which facilitates carrier-to-carrier connectivity, in order to allow end-to-end national and international B-ISDN services. It does so by defining the interface between public network providers, and specifying a number of functions required to operate a variety of inter-carrier services.

To show the applicability of these signalling characteristics we now look at how these apply to the specification of the ATM Forum UNI v4.0 signalling protocol (ATM Forum, 1996a). A Signalling ATM Adaptation Layer (SAAL) is specified to adapt the ATM service to the requirements of UNI signalling. It utilises the AAL5 "common part" protocol for un-assured transfer but adds a Service Specific Connection Oriented Protocol (SSCOP) (as specified in Q.2110 (ITU-T, 1994)) to provide recovery of corrupt or lost SDUs.

A unique virtual channel is reserved for signalling (default VPI=0, VCI=5), and by default the ATM layer service category should be either real-time or non real-time VBR (ATM Forum, 1996b). The minimum level of Quality of Service that must be reserved for the signalling channel for a given UNI interface of 8Mbit/s is shown in figure 5.1. This shows the importance of requirements imposed on signalling transport, and similar consideration must be given to the signalling requirements of any future Internet-based services, including the Alpine framework.

| ATM QoS Parameter | Value | |
|---|---|---|
| Peak Cell Rate (PCR) | 168 cell/s | ITU-T Recommendation Q.2120 |
| Sustainable Cell Rate (SCR) | 126 cell/s | At given 8Mbit/s interface rate |
| Maximum Burst Size (MBS) | 16 cell/s | |
| Cell Loss Ratio (CLR) | $10^{-6}$ | |
| Cell Transfer Delay (CTD) | Less than 500ms | ITU-T Recommendation Q.2110 |

Table 5.1: ATM Signalling QoS Requirements for 8MBit/s UNI interface

## 5.2 The Alpine Middleware Inter-networking Environment

The Alpine Middleware Inter-networking Environment (AMIE) provides the support mechanisms to enable the deployment of programmable services based upon the Alpine framework detailed in chapter 4.

As with the Alpine framework, the infrastructure draws upon the IEEE P.1520 reference model for inspiration. However, at present P.1520 provides very little guidance on implementation, based on the assumption that it is more appropriate to leave this choice to those responsible for implementation. The definition of AMIE detailed here, and its partial implementation (described in chapter 6), fills the gap between model and implementation, but it must be remembered that this is only one possible interpretation of the P.1520 architecture.

The Alpine Middleware Inter-networking Environment (AMIE) is founded upon the basis that network control functions should be deployed where they are most efficient in terms of computation and signalling performance. The P.1520 Reference Model provides a useful programming hierarchy within AMIE; functions are deployed within either of the three programming planes, depending upon function ownership and corresponding appropriate view of resources.

There are four components (figure 5.2) in the AMIE platform, namely the Active Router (AR), the Active Intelligent Peripheral (AIP), the Network Operations Services Platform (NOSP), and the Value Added Services Platform (VASP). These are defined as follows:

- Active Router - a device that receives and forwards IP packets, but extends the functionality of a traditional IP router by allowing the upload of programs to influence packet forwarding. These devices have limited computational power, and

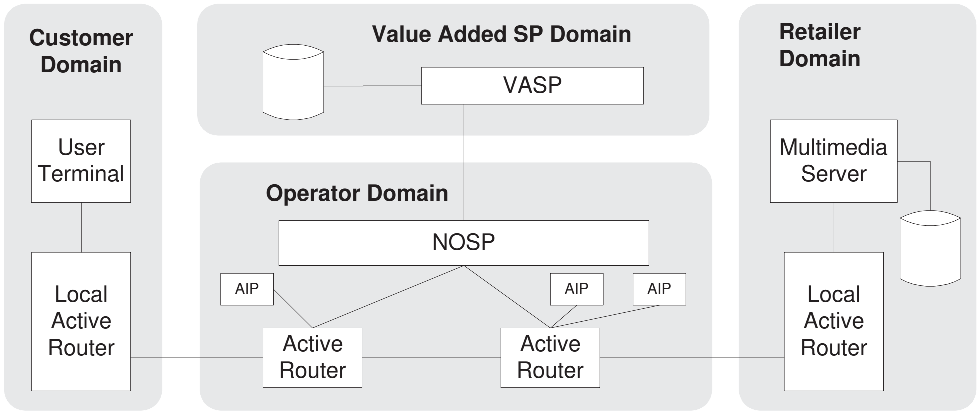


Figure 5.2: Alpine Middleware Inter-networking Environment

are only concerned with network layer (OSI layer-3) programmability. Most Active Routers reside in the network operator domain, but it is also possible for others to be part of the customer's local area network (LAN).

- Active Intelligent Peripheral - computational clusters within the network; normally a collection of workstations located in the vicinity of an Active Router. Each AIP has considerable processing power and supports all levels of network programmability above the network layer.

- Network Operations Services Platform - a collection of distributed DPE nodes interconnected together and to AR, AIP, and VASP via a kernel transport network (kTN). It is responsible for the execution of network layer services requiring a global view of resource, examples include connection management, admission control, and charging.

- Value Added Services Platform - a collection of hosts responsible for the execution of value-added network applications. The VASP is outside of the domain of the network operator and may be physically remote from the data network. For example a Value-Added Service Provider based in the US may retail a virtual private network provisioned over British Telecom's UK data network. A kernel transport network (in this example, the public Internet) is used to connect the VASP to the NOSP and data network.

The distinction between the Active Router and the Active Intelligent Peripheral is an important one. The Active Router, in common with all traditional routers, is a highly optimised device to store and forward packets at very high data rates. Conventionally they have a very high-speed backplane capable of transferring data rates in excess of the speed of an individual port. The need to provide complex packet processing as with many AN routers would have serious consequences for backplane speed and accordingly router throughput.

Therefore, active processing on the active router is limited to control plane activity (e.g. route control, and queue management) and not the manipulation of packet contents. Typical AR control plane functions include provision of quality of service guarantees by inserting new packet scheduling algorithms. Many other common examples of future "active applications" such as multimedia transcoders and active caches are not suitable for provision on Active Routers since they require significant computation (i.e. transcoders) or data storage (i.e. active caches).

For these more resource hungry "active applications", clusters of computational power are deployed into the network as Active Intelligent Peripherals (AIP). These are workstations equipped with active Execution Environments (EE) suited to active processing on the data-plane. These AIP are not to be found directly on the main data path, since they would not be capable of active processing at full trunk network data rates. These should optimally be deployed close to Active Routers, so that should an AR require greater active processing than it can itself perform, it can tunnel the stream to the local AIP cluster for active data plane processing.

An example of an AR and AIP working in tandem is for the maintenance of QoS guarantees on multimedia streams. An Active Router can monitor the dataflow, and try to assure QoS by preferential scheduling of packets belonging to the multimedia flow. On the event of failure by the AR to sufficiently provide the necessary QoS scheduling guarantees for maintenance of a multimedia flow, the AR can install a transcoder on the local AIP cluster, and redirect the stream via the transcoder application. The transcoder can convert the stream to a lower bandwidth format, thereby maintaining sufficient multimedia stream quality.

## 5.3 The Programmable Virtual Router

The programmable virtual router, located within the Network Generic Service Layer (NGSL), is responsible for the management of the network as a whole. This corresponds with the Network Management Layer (NML) in TMN and TINA, and supports the Operations Support System (OSS) functions.

The virtual router is an abstract representation of an autonomous subnetwork, the

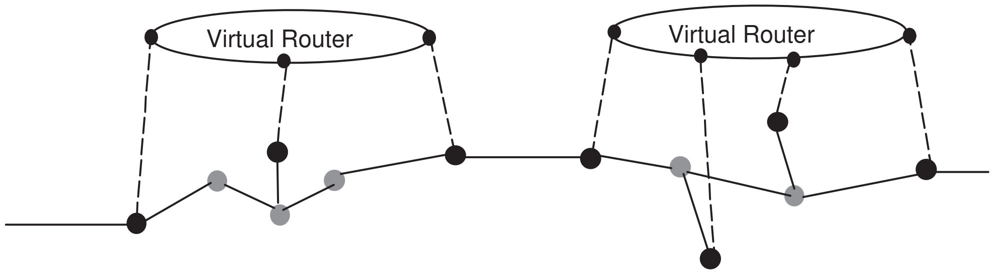


Figure 5.3: Virtual Router abstraction of underlying routing topology

internal topology of which is hidden. At the NGSL, where subnetwork federation is managed, each subnet appears to be a (virtual) router itself (figure 5.3). Given the potential for physical routers to have active code execution capability, it seems a natural extension to grant this dynamic programmability to the virtual router. The purpose of the Network Operations Services Platform (NOSP) is to provide the middleware platform infrastructure to enable the construction of this virtual router programmability. The rationale behind the programmable virtual router model, and the requirements on the NOSP are distinct from those of physical active routers and are examined below:

- Independent from physical network routers/switches: there is no reason to locate the nodes providing the virtual router abstraction locally to the routing infrastructure. Indeed, routers have limited processing power and memory, and would not provide a successful basis for the virtual router node.

- Global resource abstraction: unlike active routers that are only aware of locally significant resources, the virtual router has a view of resources of all physical nodes in their domain. Since these physical nodes are likely to be heterogeneous, a generalised programmable abstraction is provided so that active code may be ported across topologies.

- Limited to control plane programmability: data-plane programmability is not supported by the virtual router, since data packets do not actually pass through the node. Active code on the virtual router configures and manages data-plane manipulation code running on the Active Intelligent Peripherals (AIP).

- Safety and security: although the risks involved with active code executing on the virtual router should be less than on a single active router, the potential damage of errant code is more likely to have global catastrophic consequences. Errant active code on a router may cause failure of that device, a locally significant problem, but is unlikely to cause the remainder of the network to fail. Since it is the role of the virtual router to expose programmable interfaces to third party service providers at the 'U' interface, the platform must assure greater safety in 'U' interface operation than applicable to 'L' interface which is purely intra-domain (both NGSL and VNDL roles are part of the network operator business).

There is a range of potentially sophisticated active applications to be deployed in the NGSL plane, including connection management, configuration management, fault monitoring, accounting and charging. These have in common the fact that they must analyse and manipulate large amounts of data. The Network Operations Services Platform (NOSP) supporting the virtual router abstraction is an extensible cluster of DPE nodes. These can be based on general-purpose workstations; additional nodes can easily be introduced to cope with increased computational demands. In common with other distributed platforms, on the event of a failure of an NOSP node, network management functions can be migrated between DPE nodes. Outage of a DPE node in the NGSL plane can cause total network failure, while failure of an active router (VNDL plane) has only local significance, and failure at the VASL affects only non-critical value-added services.

An Application Level Active Network solution is adopted for the design of virtual router nodes. Maximising performance of nodes is not a high priority as low cost workstations are used; active code can therefore be deployed in user-space. As data-plane processing on packets is not supported, there is nothing to be gained from siting active code in the kernel.

As general-purpose workstations may be used, there could be many combinations of hardware (processor type, available memory) and software (many operating systems, including UNIX, Microsoft Windows, and VMS). This therefore makes deployment of generic functions as active code in binary form inappropriate. The accepted AN solution is either to write active code in an interpreted language (e.g. TCL) or to distribute code in a machine independent low level form (e.g. Java bytecode). Both are equally applicable for programmable virtual routers, and multiple virtual machines can be deployed on the NOSP to provide a choice of active Execution Environments.

Just as Active Routers and Active Intelligent Peripherals (AIP) enable the dynamic upload and execution of active code, the same should be true for the virtual router. The capsule-based *integrated* approach is definitely not appropriate in this instance, since the virtual routers are, indeed virtual, and process no packet flows.

In common with the FunnelWeb (previously known as ALAN) (Fry & Ghosh, 1999) application level active platform, a control API is necessary to dynamically upload active code to active nodes supporting a Java Virtual Machine. Although the purpose of FunnelWeb active code (known as Proxylets) is for data-plane packet processing, the same fundamental control primitives are required, namely *load*, *run*, *modify*, and *stop*. The distinction between *load* and *run* is important for the *discrete* approach applied at the virtual router. Active applications within the NGSL plane are likely to be complex, and large in code size. It would be useful to have the facility to upload code onto the platform prior to execution thereby saving the high latency involved in uploading code when active processing is required immediately. Surprisingly, the inclusion of a *stop* operation as a basic control primitive is not universal amongst all AN platforms. The ANEP protocol used by many AN nodes, to send code to be executed, does not provide any active execution halting primitives; it is necessary to send an ANEP packet containing user-defined code to cease execution.

However, although the control primitives are similar to other application layer AN solutions, the security requirements are not. The FunnelWeb node from the University of Technology Sydney (UTS), for example, inherits a Java security manager for its proxylets similar to that used for Web applets. However, this is inappropriate for the programmable virtual router, as active code may often need to read and write from the disk and manipulate local state information in order to process configuration, topology and admission control databases, and keep usage/accounting logs.

## 5.4 The Programmable Network Element

At the element layer (i.e. VNDL and PE planes), active processing is undertaken that has a direct influence on data streams. Active programming of network elements within the public network is the responsibility of the network operator. Element level programming is generally likely to be inappropriate for end-users, due to safety concerns and lack of network knowledge (e.g. no knowledge of network topology). The exception being routers in local area networks belonging to small workgroups, where the end-user is also the network owner/operator, and outage of network connectivity due to errant active code is not critical and should not impact upon others.

Within the middleware architecture defined in section 5.2, there are two types of active network elements, the Active Router (AR), and the Active Intelligent Peripheral (AIP). The design of the AIP is not considered in detailed within this thesis; there are many different active processing requirements, and potentially many different AIPs, each specialised to meet these requirements. An AIP designed for active caching will be optimised for efficient storage I/O, while another aimed at transcoding applications will have optimised mathematics operations and may not support any local storage. Given the breadth of AIP uses, further specification would only serve to restrict the types of active processing within the Alpine framework.

The Active Router (AR) is a specialisation of the traditional IP router, with additional processing power. Early routers and switches were often computationally powerful, including data integrity checking on the forwarding function. Increased transmission reliability, coupled with greater end-terminal sophistication, started a trend to migrate these functions out of the network. This trend is best seen in the evolution of data networking from X.25 to ATM, and most recently to the connection-less IP protocol, where with the Transmission Control Protocol (TCP) (IETF, 1981b), error recovery, retransmission, and data rate control have been migrated entirely to the end-terminal.

Simplified network infrastructure has enabled the engineering of equipment capable of far greater speeds. Most approaches to active networking recognise this, and realised that it is not feasible to actively process all packets passing through modern high performance routers. This has led to the definition of the fast/slow data path model (Cohen & Rangarajan, 1999), where the traditional high performance packet forwarding function is not disturbed for the majority of packets. Only a small subset of packets matching distinct criteria (e.g. matching a specific flow) are withdrawn from this data forwarding function and passed over to the slow data path for active processing. This requires an efficient method to filter those packets for active processing, ideally a no copy architecture not impacting upon performance of non-actively processed packet forwarding. The Linux operating system that forms the basis of many AN platforms now includes kernel functions[1] to allow user-space applications to easily access packets in transit through the node.

The Alpine framework rejects this active node model. These research platforms are based on "off-the-shelf" PC components and commercial software; they have vast computational power and memory, but few input and output linecards. High performance commercial routers are highly optimised for packet forwarding, with limited computational complexity that might impact upon this core function. Therefore, as stated earlier, Alpine active routers are constrained to control-plane active processing and cannot access the IP packet itself. Rather than the kernel filter and slow path, those packets that require Active Data Plane processing are redirected by the router to an appropriate AIP.

[1]Netfilter available in Linux 2.3.x series kernel, details at `http://netfilter.kernelnotes.org/`

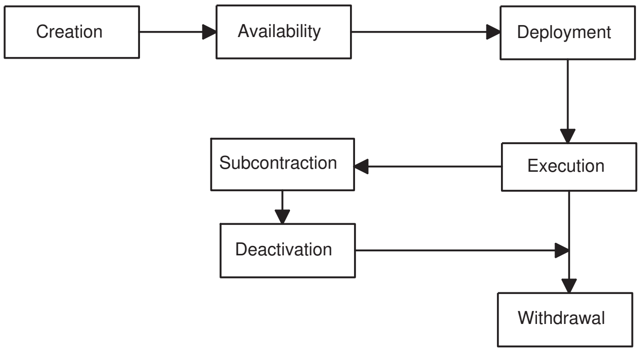


Figure 5.4: Alpine Active Lifecycle model

Active code is subject to a lifecycle model similar to the OMG object lifecycle (OMG, 2000b). The OMG object lifecycle service provides the mechanism for object creation, deletion, and migration (moving between nodes in the DPE). The Alpine lifecycle (figure 5.4) is derived from a similar background, but is tailored to active code rather than distributed object deployment. The model consists of seven states: creation, availability, deployment, execution, subcontraction, deactivation, and withdrawal.

- Creation: service requirements capture and implementation of code for active service.
- Availability: active service made available for deployment. Within the implementation of the Alpine framework, this stage is provided through making the active code available on a Web (HTTP) server.
- Deployment: upload of active code to an Active Router.
- Execution: execution of active code on an Active Router.
- Subcontraction: optional state, if the Active Router cannot meet active processing criteria, an active service may be subcontracted to another active node (e.g. an Active Intelligent Peripheral (AIP)).
- Deactivation: An active service that has been subcontracted may be deactivated by the parent contractor.
- Withdrawal: finally, functions are required to withdraw all instances of an active service on completion of the service lifetime.

It is a requirement of the Active Router that it must support this seven-state lifecycle. The Alpine Active Router is based upon the *discrete* approach to active code deployment,

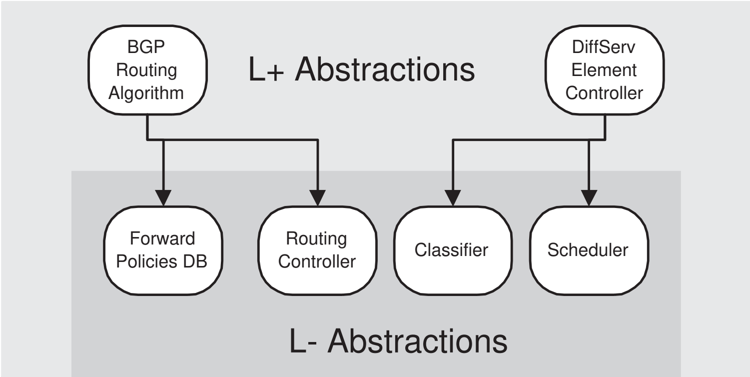


Figure 5.5: Two-tier model for abstracting resources

and uses the same *load*, *start*, *modify*, and *stop* primitives as for the virtual router. In the implementation of the architecture, further details of which are detailed in chapter 6, active code is made available by placing on an HTTP server and transit between states is controlled via CORBA interfaces exposing the four control primitives.

## 5.5 The 'L-' Interface

The original IEEE P.1520 reference model (Biswas & et al, 1998) consisted of four interfaces, 'V', 'U', 'L', and 'CCM'. These accorded a clear mapping onto ATM network technology, but at that time a definition of how these map onto IP was not clear (Biswas *et al.*, 1998). Since then, a separate subworking group has been created to consider the relationship between P.1520 and the Internet. This has culminated in the creation of the P.1520.3 project, to allow the definition of interfaces for IP technology, independent of the standardisation now taking place for ATM.

The P.1520 Reference Model for IP rejects the 'L' and 'CCM' interface definitions, since a clear mapping to IP has not been made. This was replaced with a two tier 'L' interface (Vicente *et al.*, 1999), of generic and service specific abstractions (figure 5.5).

The core abstraction, termed 'L-' interface, is a view of router resources that is neutral of service context. It is envisaged that an advanced network element programmer can use this abstraction to construct new router behaviours and functions without knowledge of the underlying device implementation.

A second abstraction, termed 'L+', provides a service specific abstraction of resources. The programmable interface thus has only meaning within a specific context, for example configuring per-hop-behaviours (PHB) on a router in a DiffServ cloud. Al-

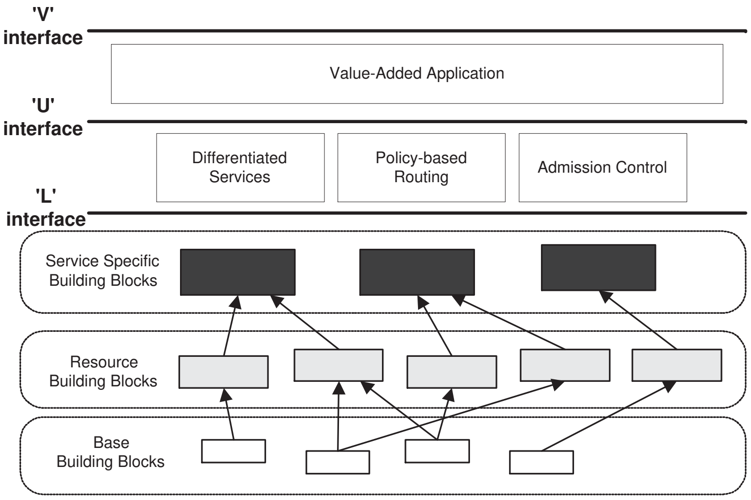


Figure 5.6: Three-tier model for abstracting resources

though this abstraction limits programmability, the abstraction provided is safer, more secure, and easier to program. Many equipment vendors would have concerns about granting the wholesale programmability of the 'L-' interface and would find this service configuration interface more appropriate.

This two-tier model is now obsolete, being replaced by a new three-tier model (Vicente & et al, 2000) (figure 5.6, taken from P1520/TS/IP013). The two 'L' interfaces, 'L+' and 'L-' are now redefined as "Service -specific building blocks" and "Resource building blocks" respectively. A third abstraction, "base building blocks" (the lowest layer of abstraction) has been added for abstractions that have no service or resource significance from a network element behavioural or packet processing perspective. These have been defined as component, action, condition, target and processing unit. At the present time a more detailed definition of the role of these abstractions has not yet been defined.

The Alpine framework is based on the former two-tier abstraction model. The three-tier model is not yet fixed and is in a constant state of evolution (IEEE, 2000), and therefore could not form a reliable basis for a programmable node abstraction. The P.1520 model is based on a static model of programmability and does not yet consider the advent of active network technology. Although not yet recognised by the whole P.1520 community, the active node application programming interface (API) is simply a realisation of the 'L-' interface abstraction. Furthermore, any configuration interface created by active code naturally resides as an 'L+' interface under the P.1520.3 model.

The challenge is to define a complete programmable abstraction to capture the complete functionality of a router. Although the task of generating an abstract model was relatively simple for ATM switches, capturing all functionality of an IP router has proved difficult. The simple fundamental function of any router is on a per packet basis to receive packets, make a decision on the appropriate output port, and forward the packet to that output port. The introduction of advanced services like Quality of Service and multicast has introduced heterogeneity into the router model.

One of the simplest representations of a fundamental router architecture can be found on the Pronto active network platform (Hjálmtýsson, 2000). Their model consists of three major parts, a forwarding engine, service generic facilities, and an active execution environment. The forwarding engine and generic services are fundamental to all router architectures, and have been further refined into six abstractions:

- Flow Classifier: maps packets into equivalence classes based on attributes of packets. This might be micro-flows (based on source and destination IP address and port) in an IntServ scenario, or could be coarse flow classes (based on TOS field) on DiffServ routers. On all routers, flows (equivalence classes) are associated with a forwarding state (i.e. an output queue).

- Queues: packets once classified are forwarded to an output queue, from which on a First-In-First-Out (FIFO) basis they are dispatched back into the network.

- Link Scheduler: responsible for the maintaining of predictable and fair bandwidth allocation between output queues.

- Signal processor: responsible for implementation of IP signalling protocols. These may include ICMP, IGMP, RSVP, BGP, etc.

- Resource Manager: schedules access to processor and memory, and ensures isolation of service specific features.

- Query and Response Engine: maintains a Management Information Database (MIB) containing statistics on router operation.

An alternative model is adopted by Darwin (Chandra & et al., 1998) as described in section 3.6.1. Here, five programmable abstractions are provided: flow manipulation, flow redirection, resource management, queue management, and active code (delegate) communication. Amongst the numerous programmable platforms there are many other router control abstractions, each not fully capturing all router functionality.

Clearly, the active node programmable abstraction or 'L-' interface needs to be extensible so new methods can be added as new technology emerges. However, this clearly makes the writing of code for Active Routers difficult because the author cannot guarantee the same set of programmable abstractions. This is the motivation behind the adoption of CCML (section 6.5), so the AR node can select an implementation appropriate to the abstraction supported by that particular node. Active networking platforms have up until now identified an Execution Environment (EE) by a unique ID. For example, the ANEP protocol (Braden *et al.*, 1997a) includes a 16 bit Type ID field, that is used to identify the EE with which the encapsulated active code is compatible. The authority for defining these unique identifiers rests with the Active Networks Assigned Numbers Authority (ANANA).

Given the multiplicity of possible runtime environments and 'L-' programmable abstractions, the number of unique IDs could be considerable. The integer type ID system is sequential, and cannot support versioning. There is no obvious way for an active node to know that code written for an ID of 67 is compatible with an EE of ID 127, because ID 67 is merely a subset of the functions supported by 127.

Instead, an Execution Environment is identified by two sets of values: supported run-time environments (e.g. Java 2 VM, TCL interpreter) and the set of supported 'L-' abstractions. A node can make the decision on whether the active code is admissible and which version is compatible, simply by validating if these two values are supported. New programmable abstractions can be added without affecting legacy active code. A component description language (CCML) and mechanism for resolving compatibility for Alpine is described later in chapter 6, section 6.5.

According to the P.1520 specification for 'L-' interfaces (Vicente *et al.*, 1999), when designing a programmable abstraction the interface should meet seven fundamental interface requirements:

1. Open programmability: separation between hardware and software, fostering third party service creation and open competition.

2. Operational support: management and administrative functions must be provided that over a slow time-scale provide network administrators and architects with intelligence gathering and control functions.

3. Service provisioning: through open APIs, third party service providers must have the ability to modify existing network services as well as the ability to introduce entirely new network services to the router or switch functionality.

4. Extensibility: must provide extensibility, and avoid creating proprietary interface formats.

5. Programmable abstraction granularity: interfaces must expose an appropriate granularity of abstraction of fundamental router components.

6. Time-scale flexibility: access and control of router resources must be achievable over alternative time-scales (control, management, and transport).

7. Resource partitioning: router resources should be provisioned so that several service providers may deploy and operate their network architectures on the same physical infrastructure confined to their allocated portion of the routers' resources.

Within IEEE P.1520.3, the task of defining the initial 'L-' programmable abstractions is still ongoing. The granularity of abstraction is a key question; many equipment vendors fear that exposed control and manipulation of network element resources may compromise proprietary features and prevent differentiation between competing vendors' equipment. Furthermore, any standardisation will inevitably be incomplete; the standardised interface must be extensible so as not to impede feature development in IP routers. It would certainly be unfavourable to replace the old standardisation procedures (e.g. ITU-T, ETSI, etc.) with something equally slow from IEEE P.1520. Modern software engineering principles such as distributed object orientation are used to encourage flexible abstraction modelling, interface inheritance, and extensibility.

According to the most recent specification (Raguparan & Vicente, 2000), there are 13 core abstractions (see Table 5.2).

## 5.6 Virtual Private Signalling

The Alpine framework, in common with other intelligent and programmable network architectures, has a high degree of utilisation of inter-component signalling independent of the multimedia transport. This identifies a separation between the transport of multimedia data and the signalling messages. The separation may be only logical, where signalling and multimedia data share a common physical network, or may be physical with the provision of an isolated network dedicated to transport of signalling messages.

Within the Alpine framework, there exist two types of signalling, active code upload, and fine grained 'L+' interface RPC (remote procedure call) operations. Each of these has distinct Quality of Service (QoS) and Quality of Protection (QoP) requirements. Active code upload being a single bulk data transfer, while 'L+' RPC communication more commonly generates regular smaller bursty traffic patterns.

| 'L-' abstraction | Notes |
|---|---|
| Packet Classifier | |
| Shaper | |
| Dropper | discards packets and related ICMP processing |
| Meter | measures packet level statistics |
| Queue | |
| Packet Scheduler | |
| Micro flow table (MF) | packet header information and traffic profile related to a micro flow |
| Behaviour Aggregate table (BA) | packet header information and traffic profile related to aggregates |
| Network Address Translator | |
| Computing Resource | this includes CPU, additional memory used (e.g. memory used for route computation) |
| Forwarding Table | |
| Routing table | various routing protocol databases |
| Header manipulation | |

Table 5.2: 'L-' interface abstractions

The kernel transport network (kTN) has traditionally been charged with the responsibility for transporting these signalling messages. The isolation of this distinct signalling role has recently become unfashionable with the belief in the trend towards integrated networks and the dominance of the Internet. However, there are benefits to be gained by considering this role in isolation (Banfield *et al.*, 2000b); signalling has very distinct characteristics to those of data transport. The segregation of data and signalling can allow mechanisms to develop for the provision of signalling QoS, without disturbing the efficient operation of the main data network.

Many network control and management approaches have concentrated on identifying the main business roles and interfaces, and little thought has been given to the provision of signalling functionality or the relationship established between this and the communicating entities. TINA, for example, identified the requirement for a transport function dedicated to signalling early on. However, to date there has been no detailed definition

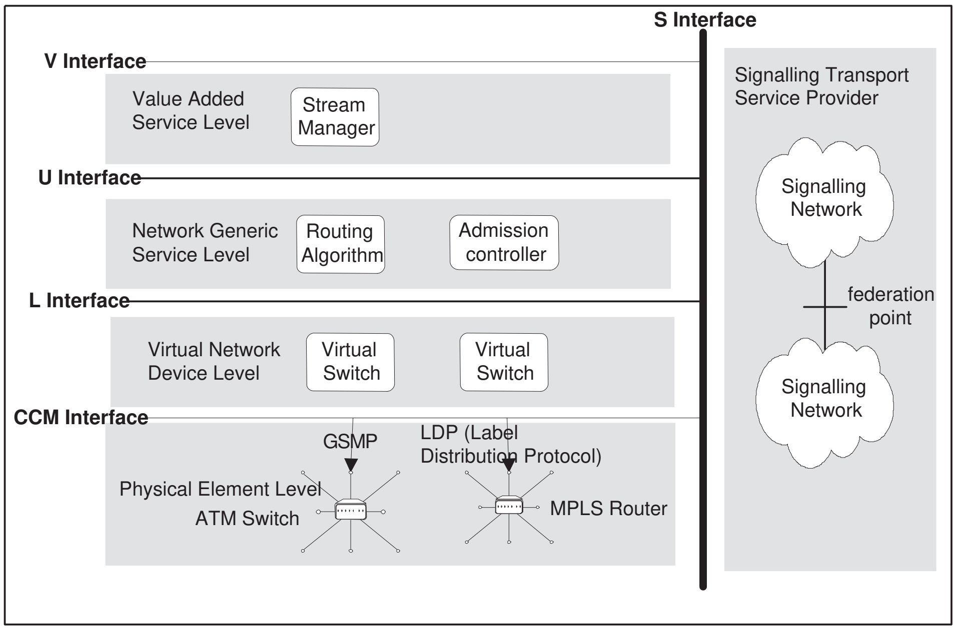


Figure 5.7: IEEE P.1520 RM extended with Signalling Transport Service Provider

of the TINA kTN (Kernel Transport Network).

An important contribution to the P.1520 reference model is the addition of a new business entity, the Signalling Transport Service Provider (STSP) whose function is to provide the transport of signalling messages (Figure 5.7). This applies equally to TINA, P.1520 and IN architectures but, in this thesis, P.1520 is used as the reference model for further exploration.

The isolation of this role extends the open market principles embraced in Open Signalling architectures. It provides for the outsourcing of signalling transport by the connectivity provider. This allows both the connectivity provider and the STSP to concentrate on building networks for efficient provision of their particular type of connectivity. This separation is, of course, a logical one. While in some circumstances the separation of STSP and multimedia transport network provider is made, in many cases it will be more appropriate in terms of cost for a single provider to combine both functionality sets. This detail is hidden from the control and management software, which should be unaware of how the signalling is transported, or of the ownership of the transport network.

Nevertheless, there are circumstances where this separation may be realised physically. The connectivity provider may not own a network appropriate for the transport of signalling (e.g. a wireless provider), or perhaps such high demands on network service quality are made that increased reliability through the use of an isolated signalling network provider may be appropriate. Another very important issue is that there can be no guarantee that value added service providers (located at the VASL (Value-Added Service Level)) are physically connected to the multimedia network on which they compose their "value-added services". For example, a US based value added service provider may retail a VPN service over a network owned by British Telecommunications plc and located in the United Kingdom. A third party STSP, perhaps an ISP, can act as a communication enabler between these geographically disconnected providers.

### 5.6.1 Requirements on the Signalling Transport Reference Point

The IEEE P.1520 Reference Model is based on a hierarchy of horizontal interfaces providing programmable abstractions through which functions located in the various horizontal planes may communicate. In addition to the introduction of a new vertical STSP plane, a corresponding vertical interface (the 'S' interface) is introduced, through which the VASL, NGSL, and VNDL establish a relationship with the signalling transport service provider (STSP). Two or more entities that wish to communicate must each form a binding with an STSP to enable them to do so. Since communication may be both intra-level (between entities located at the same service level) and inter-level (entities located at different service levels), the specification of the interface may be extended to service specific requirements appropriate for each signalling relationship.

The 'S' interface is used to negotiate the service requirements of the communicating entities much in the same way as a Service Level Agreement (SLA) is negotiated between an end customer and network operator over the UNI. It is not appropriate to define a complete set of SLA requirements that may be negotiated over the 'S' interface, since it will inevitably be incomplete. This is entirely consistent with the IEEE P.1520 reference model, where all of the interfaces beyond the 'L' interface are open to definition at the time of interface realisation. Flexible policy specification languages and XML metadata schemas can be defined similar to those for traditional SLA and policy exchange. However, for completeness an adopted mechanism should recognise the five essential areas of functionality that should be exposed over the 'S' interface:

1. Reservation of Service Quality - an agreed binding level of service quality must be negotiated between the parties. This could include high-level criteria such as "mean time to failure" and "percentage packet loss" or more detailed QoS characteristics such as bandwidth, delay, error rate and jitter.

2. Composition of Virtual Signalling Groups - although the signalling network provider may have many concurrent users of the signalling network, each user should be kept isolated from one another. Virtual signalling groups should be supported in a similar way to the provisioning of virtual private networks (VPN) to end-users. These groups may be used to constrain both intra and inter service level communication but should also enable federation between co-operating service providers.

3. Timescale and priority of interaction - a distinction is often drawn between the priority and timescale of connection control signalling compared to FCAPS functionality (ITU-T, 1997) (Fault, Configuration, Accounting, Performance, and Security Management).

4. Quality of Protection - signalling messages may have very high security requirements, since the interception or spoofing of signalling may provide a loophole to compromise the integrity of the multimedia data network.

5. Charging - given that the role of the Signalling Transport Service Provider may be out-sourced to an alternative operator, provision should be made for a transparent charging system. Even where both signalling and multimedia transport are kept in-house, it may be useful to know the associated cost of the signalling created by new intelligent and value-added services in order to charge correctly.

Alpine specifies the use of a distributed middleware platform that will take care of these details. The underlying primitives of the kTN are mapped onto the distributed object binding architecture offered by the middleware platform. Where the platform supports explicit object binding, the relationship between functions and the STSP can be created at run-time. Other platforms, including CORBA, which provide only an implicit binding model, must configure relationships with the STSP pre-runtime or must devolve to an autonomous 'S' interface binding agent within the client-side stub.

The Alpine framework is aimed at an IP based network infrastructure, and it is likely that both the data transport network and the kTN will be physically provisioned over the same physical infrastructure. However, this does not preclude a physically separate kTN, for example given the scenario of a data transport based on a corporate LAN but with out-sourced value-added services (located in the VASL plane) interconnected via the public Internet.

An unfortunate side effect of introduction of the Signalling Transport Service Provider (STSP) is a resurgence of the kTN-bootstrapping problem. It is necessary for active and DPE nodes to obtain an initial reference to the 'S' interface in order to commence the negotiation with functions located in other planes. Traditionally when using circuit-switched networks it is common to reserve permanent virtual circuits (PVC), established at network configuration time for bootstrapping network signalling. Unfortunately, this solution is not applicable to connection-less networks typified by IP. The simplest IP solution is simply to configure on each active and DPE node a bootstrap IP address during the configuration stage of the deployment cycle. This solution is less than ideal since it requires the manual (or some additional automation of) configuration of every active and DPE node to be deployed. Obviously, it would be ideal if nodes could be deployed and withdrawn without manual intervention.

Other solutions require the establishment of a virtual private network (VPN) similar to the role of the PVC for bootstrapping. However, this solution is no better than manual configuration since it requires an overhead of creating and maintaining the bootstrap VPN. Theoretically at least, the most promising approach would be the adoption of a URN (Uniform Resource Name) resolution scheme, whereby the URN of the 'S' interface resource can be resolved. However, at present an efficient URN resolution scheme is not available, and is an area for further exploration.

## 5.7 Summary

This chapter has defined the central role of distributed processing in the Alpine framework and has described the contribution of a novel active DPE architecture for realisation of telecommunication services.

The discussion started by considering the important role of signalling in all networks, due to the distributed nature of network functions. Section 5.1 presented a controversial division of telecommunications signalling functionality and showed this to be equally applicable to modern connection-less data networking.

Once the division of network functions and resulting importance of signalling was recognised, the Alpine Middleware Inter-networking Environment (AMIE) was introduced in section 5.2. AMIE provides the middleware support mechanisms to enable the provision of telecommunication services as envisioned in chapter 4. The design of AMIE divides the platform into four components: Active Router, Active Intelligent Peripheral, Network Operations Services Platform, and the Value Added Service Platform.

Together these elements permit active programming in two computation planes, both at the Network Management (NML) and at the Element Management (EML) Layers.

Programmability at the NML was discussed in detail in section 5.3. Active programmability at the NML is achieved through the construction of the virtual router, a property not found in other active platforms. Given the Internet's autonomous system (AS) peering model, with each AS being opaque and appearing as a virtual router, it is natural that the same level of programmability as at the element should be extended to the NML.

At the EML, discussed in section 5.4, programmability is divided between control and data-planes. Programmability of these planes is allocated to the Active Router and Active Intelligent Peripheral respectively. The core of this programmability is expressed through the features exposed by the active node API analogous to the 'L-' in the P.1520.3 reference model. Several important realisations were made, differentiating the design of this interface from those of conventional active nodes. For those nodes conforming to the DARPA Active Node architecture, the programmable API and run-time environment (e.g. JVM) are identical, and only one identifier is needed to recognise an execution environment (EE). Instead, 'L-' API and run-time environments should be expressed separately, allowing independent development of each. It was suggested, indeed, that unlike with ATM, it is infeasible to define a complete programmable abstraction for IP routers as new functionality is continually evolving. Instead, active code and EEs should be expressed as the set of 'L-' API abstractions supported/required. The need for a compatibility resolving system (i.e. Component Compatibility Mark-up Language) was defined; this will be described in detail in chapter 6, section 6.5.

Finally, the chapter closed by suggesting an extension for the IEEE P.1520 Reference Model resulting from the AMIE. A new entity, the Signalling Transport Service Provider (STSP), was introduced along with its corresponding vertical 'S' interface reference point.

# Chapter 6

# Realisation of the Alpine Programmable Network Platform

This chapter describes the engineering of a prototype realisation so that the validity of the proposed Alpine framework may be evaluated (Chapter 4).

## 6.1 Requirements of the Experimental platform

The Alpine framework covers a broad scope comprising service, network and element domains. It is obvious that a realisation of the complete framework is not feasible within realistic manpower restrictions and instead it was necessary to select only a subset of the framework for realisation. For this, the framework realisation chapter, and the successive evaluation chapter, it is the element layer, and in particular the Active Router (AR) that is considered.

The Active Router was chosen as it forms the fundamental programmable component of the Alpine framework (section 5.2). It provides the programmable control-plane environment that introduces active programmability into the network layer-3 functions. When additional active data-plane programmability is required, it is the responsibility of the active router to redirect packets to the appropriate Active Intelligent Peripheral (AIP).

Over the last couple of years there has been a considerable amount of effort on the building of efficient programmable nodes. Many of these schemes have taken advantage of features of particular Operating Systems and processor platforms. Others in the active network research community have concentrated on active network languages (e.g. OCaml, PLAN), and appropriate security mechanisms (e.g. Virtual Machines).

None of these issues is considered for the design of the prototype active node. There seems little purpose in designing a highly optimised platform based upon current commercially available hardware and general purpose operating systems. Should active and programmable nodes become mainstream, the major router vendors (Cisco, Nortel, etc.) have considerable engineering manpower at their disposal to optimise node programmability for their particular platform.

The emphasis in the design of the prototype Active Router has been to experiment with the appropriate programmable API abstraction and service control features to be exposed. This has been considered in three parts:

- Out-of-band active node control interface
- The Active Programmable 'L-' API
- 'L+' service oriented control interfaces

An important aim of the Active Router design has to been to maintain consistency with the IEEE P.1520 project on standardisation of programmable interfaces (section 3.4) (Biswas & et al, 1999), which has concentrated on defining programmable abstractions at the Virtual Network Device Level (VNDL). The prototype Active Router therefore serves as one of the first router designs compatible with the emerging P.1520.3 programmable IP element architecture. Validation of the appropriateness of the design for the prototype should serve as input into the ongoing standardisation effort with the P.1520.3 subworking group.

Since the aim of the prototype router was not to build an optimally efficient backbone router, several design choices were made on pragmatic grounds. Firstly, current commercial routers are generally built on proprietary hardware and software solutions, which it was not possible to obtain access for the purposes of this experimental work. Router and switch vendors have proved very protective of their internal switching architectures and proprietary APIs, leading to questions as to whether they are yet ready to accept programmable element technology. Instead, a router based upon a general purpose operating system (in this case Linux) such as used in small local area routers has been selected. While not offering the high through-put of a dedicated real-time network element operating system, by engineering the prototype on the Linux platform it is possible to take advantage of well established "off-the-shelf" compilers, interpreters, debuggers, and editors.

Secondly, although there has been a wealth of research into appropriate programming languages for active networks, and ensuring security and consistency, these issues are outside the scope of the prototype and Java has been selected as the active node programming language. The Java language was chosen because it is an easy language in which to program, thereby allowing fast prototype realisation, and provides a rich set of built-in classes for I/O, and data storage. The latest Java 2 (Sun Microsystems, n.d.) definition of the language has been selected for the prototype implementation because of its advanced configurable security mechanism. This enables the virtual machine security model to be defined to suit the restrictions imposed upon the active node execution environment.

The Active Router is an OSI layer-3 (network layer) device, concerned with the forwarding of IP packets, and is therefore not concerned with the underlying data-link layer. The inevitable choice for implementation was between building a router for the IPv4 or IPv6 protocols. The Active Router model is independent of this choice, but the selection impacts upon the type of programmable control plane features available. Although IPv6 would have provided a richer-featured experimental environment, and while not yet in wide-scale deployment, could conceivably be mainstream by the time active network technology is ready for roll-out, the IPv4 protocol was selected. The current development toolkits and available test applications are simply better supported for the current IPv4 protocol, and it was decided that, the design of an experimental active IPv6 router would pose at the current time too great a risk.

## 6.2 Programmable Node Architecture

Useful experience has been gained deploying and experimenting with a number of active networking platforms. Two of these platforms, ALAN and LARA, have provided a useful input into the design of the prototype Active Router.

### 6.2.1 The ALAN Active Node

The Application Level Active Network (ALAN) system (Fry & Ghosh, 1999; Fry & Ghosh, 1998) was developed by the University of Technology Sydney (UTS) for value added service provision. The system consists of standard IP applications such as WWW browsers and HTTP servers connected to regular IP network infrastructure. Active network service enhancement is made via Dynamic Proxy Servers (DPS) located in and around the network. Active Code, so called proxylets, act as communication proxies, providing active processing on the data stream. However, unlike network layer active networking schemes, packets from the client/server for active processing must be ex-

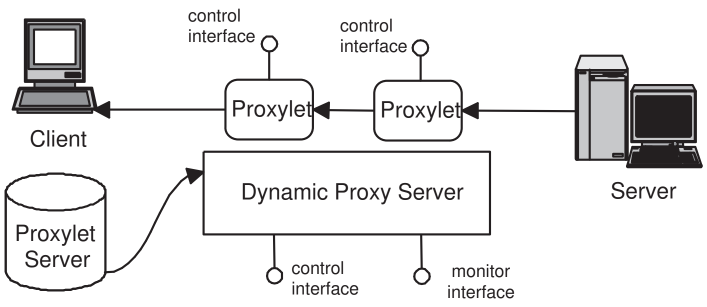


Figure 6.1: Application Layer Active Network (ALAN) Architecture

plicitly addressed to the proxy. This works well in a Web based scenario, where use of HTTP proxies is commonplace and supported by all major WWW browsers. For other types of application that do not explicitly support the use of proxies, this approach is less appropriate.

Typical active applications that may be usefully deployed in an ALAN system are transcoders, stream re-directors, and overlay network services (e.g. overlaying a multicast service over a uni-cast network infrastructure).

The current implementation (Ghosh, 2000), now know as FunnelWeb, is based on the Java 2 language and runs on the JDK 1.2.2 JVM. The architecture in figure 6.1, shows the basic components in the ALAN system. The Dynamic Proxy Server (DPS) forms the Active Node control and execution environment. It exposes two interfaces over Java RMI (a Java specific distributed RPC mechanism), one for control, the other monitoring. The control interface is a simple generic interface enabling a third party to initiate the downloading of proxylets into the DPS. In the current implementation, proxylets are made available for download by placing them on HTTP servers (proxylet servers) in and around the network. These proxylets are downloaded onto the DPS over HTTP by passing their URL (Uniform Resource Locator) over the RMI control interface. A monitoring interface is provided by the DPS to enable third parties to observe the state of proxylets currently running.

In the current implementation each proxylet runs in its own Java virtual machine. On the event of the DPS downloading a new proxylet, a new DPS virtual machine is created, each with a control interface to enable the commencement and halting of execution of the proxylet.

The ALAN environment in its current usage scenario is of no benefit to the design of the prototype Active Router. It is purely concerned with user-space application layer active processing. However, the FunnelWeb implementation contains a number of useful and innovative properties and characteristics that are applicable to the Active Router.

The selection of a general purpose programming language (in this case Java), enables fast efficient development of proxylets utilising off-the-shelf compilers, debuggers, and JVMs. While this may not offer optimal processing efficency, Java enables rapid prototyping.

The out-of-band control mechanism enabling third parties to control the downloading and execution of proxylets is another useful feature. This enables many different business roles to co-operate in the provision of active services. The downloading of active code pre-execution is an especially useful feature, enabling services to be installed appropriately ahead of use. Active Services can then be instantiated virtually instantly upon request rather than with the high latency normally associated with *discrete* active networking solutions.

Our experience with building proxylets to run on the FunnelWeb node has shown up deficiencies in the system. The choice of a user-space Java Virtual Machine environment is inefficient and ill-suited for heavy data-plane processing active applications (e.g. transcoding). The current prototype implementation further renders the system inefficient, because a completely new DPS in its own virtual machine is spawned for each proxylet.

The architecture lacks a node discovery scheme; third parties wishing to instantiate active services must explicitly address the DPS node upon which they require the active service to execute. Packets for active processing require similar explicit addressing to the proxylet on a particular active node, in many cases requiring the modification of application code running on the user terminal.

### 6.2.2 The LARA Active Kernel

There were a number of candidate modular network OS kernels suitable for use in the Active Data Plane role including, the Click router (Morris *et al.*, 1999; Kohler *et al.*, 2000) from MIT, and Router Plugins (Decasper *et al.*, 1998) from ETH and Washington University. However, most of these are in limited states of completeness, and therefore for pragmatic reasons, the Lancaster Active Router Architecture (LARA) was selected to form the ADP component of the prototype Active Router.

The Lancaster Active Router Architecture (Cardoe *et al.*, 1999) was developed during 1998-99 by a small team in the DMRG laboratory, to provide a very high performance active node platform. The LARA architecture considers both hardware and software requirments, and consists of four parts, namely the Cerberus hardware, the LARA Plat-

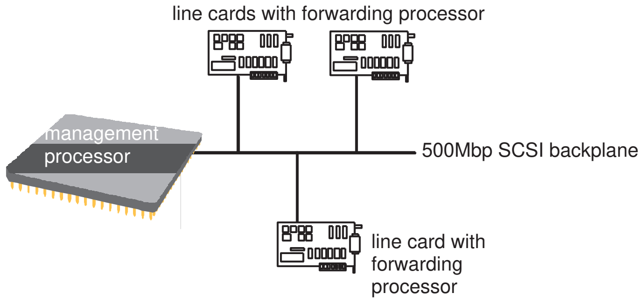


Figure 6.2: LARA Cerberus Hardware Architecture

form Abstraction Layer (LARA/PAL), the LARA Management and Policy Database (LARA/MAN) and the LARA runtime execution environment (LARA/RT).

Cerberus is the hardware architecture/platform that forms the basis of the LARA node. It aims to provide an inexpensive "edge router" built from off-the-shelf hardware components. The design (figure 6.2) is based around a dedicated active forwarding processor per port, connected together via a high speed backplane. A single management processor is responsible for active code compilation, configuration, authentication and policy management. Having a dedicated high performance forwarding processor per input/output port enables extensive computational processing to be performed. In the Cerberus implementation, each forwarding engine comprises a dedicated Intel Pentium II 266MHz processor equipped with 128MB RAM. These forwarding engines are interconnected via a high performance backplane (e.g. SCSI or ATM switch fabric), in the SCSI based demonstrator enabling backplane speeds of up to 0.5Gbps.

When designing LARA it was foreseen that no one Active Node execution environment would be able to provide for all active processing requirements. Instead, LARA/PAL was developed; a platform abstraction exporting a set of programming primitives (scheduling, memory management, network bandwidth management, policy enforcement). These abstractions are mapped onto the execution environment to create a virtual machine for each active process.

Installing active code in a shared network device introduces severe security implications. In the LARA architecture, these are addressed through the use of a policy infrastructure managed by the LARA/MAN component. Before active code is deployed on a forwarding processor it must undergo a two stage authentication process. First it is quickly evaluated against known security policies, for example, does the active code come from a known malicious source. If this stage is passed, then the code may go on to the second stage of EE specific authentication. Once these authentication tests have been passed, active code can only be instantiated if a matching policy can be found in the LARA/MAN.

The final component is the run-time execution environment. LARA is designed to support multiple concurrent run-time environments (SwitchWare, ANTS etc.), however currently only one, the LARA/RT is supported. Active code for the LARA/RT is composed of kernel loadable modules written in the C programming language. These may be dynamically inserted and withdrawn from the LARA node kernel, which is based on an extension of the Linux 2.2.$x$ kernel.

At present, only a small subset of the LARA architecture has been implemented. However, it has provided a useful input into prototyping an active router. The decision to place data-plane active processing into the kernel, enables an efficient zero-copy architecture which can support exceedingly high data-rates. However, the associated LARA/RT is difficult to program, consisting of little more than the semantics of the Linux kernel with added kernel threading and packet capture support.

## 6.3 Architecture of the Active Router

At this point it is appropriate to restate the goals of the prototype platform. The aim is to experiment with the development of programmable abstractions/interfaces, both the active node API ('L-' interface) and the control interfaces exposed by the active applications ('L+' interface). The primary goal of active applications is therefore active control-plane processing and not heavyweight data-plane processing typical of application specific active services (e.g. transcoders, active caches).

### 6.3.1 Control and Data Plane Programmability

The prototype divides router programmability into two environments, the Active Control Plane (ACP) and the Active Data Plane (ADP). The division is simple: functions residing in the ACP are those off the data path and concerned with the monitoring/influencing of state held within the router. A typical function residing at the ACP is a routing algorithm (see figure 6.3), responsible for calculating optimal routing strategies. Influence of the forwarding path is made by the ACP route calculation function. This is not by processing data packets themselves but by storing appropriate routing state in the routing table. ACP programmability is therefore isolated from the main function of the router.

The Active Data Plane (ADP) is where the core router functions reside. Functions

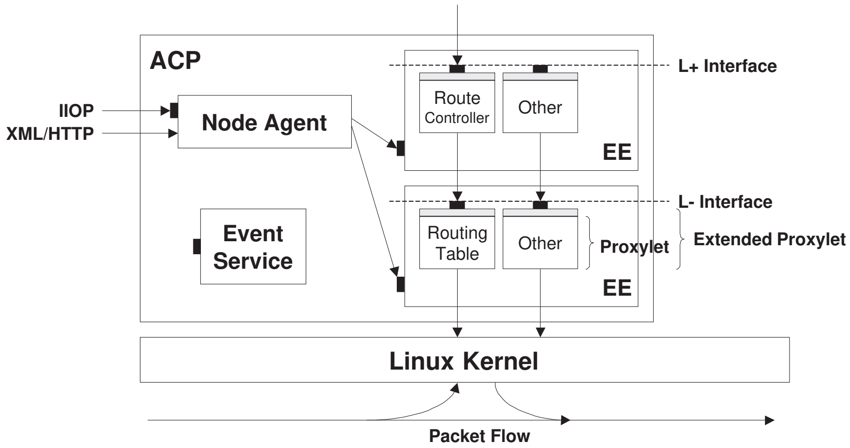


Figure 6.3: Active Router Architecture

residing in the ADP must be capable of processing packets at full line speed. Very little active processing is appropriate at this high-line rate; typically when data-plane processing is required, those flows requiring processing should be extracted and forwarded to a dedicated Active Intelligent Peripheral (AIP). Data-plane processing can then be undertaken by the AIP at much lower line rates.

Many core routers will have highly optimised ADP functions enabling them to process packets at many gigabits per second. It is unlikely any degree of third party programmability could be offered on such routers without severely impacting core network performance. However, with lower throughput routers such as edge and perhaps border routers and those found within local area domains, it may be appropriate to offer some ADP programmability. Whether or not third party ADP programming is supported, the purpose of functions residing at the ADP is to expose an interface to the Active Control Plane (ACP) at the 'L-' interface reference point.

### 6.3.2 Implementation

The implementation of the prototype Active Router explores both ACP and ADP programmability. The Linux (LARA) kernel is adopted as the ADP programming environment as there appears little to be gained from the considerable engineering effort required in the development of a new programmable node kernel.

The main ACP programming environment is based upon a user-space Java Virtual Machine. It possess many of the same characteristics as the ALAN (FunnelWeb) Dynamic Proxy Server but shares none of the same code. Although similar in design, the Active Router's ACP and the ALAN DPS are designed to serve very different types of active network programmability.

Figure 6.4 shows the main components in the prototype Router, detailed further as follows:

- **LANode Super Server**: is the entity responsible for the management of the Active Router platform. The main function is to receive requests for the instantiation of services from third parties, to obtain implementations of these services, and finally to spawn a Proxylet Environment responsible for the execution of that service. Unlike the FunnelWeb DPS, the Super Server role is separated distinctly from proxylet execution. The SuperServer exposes three external interfaces:
  - `UK.ac.lancs.sdyn.Sdyn` - a CORBA interface providing two methods. The first to request the upload of an active application, the second to obtain the object reference of the Super Server's registry interface.
  - `UK.ac.lancs.sdyn.SdynAdmin` - Administrative interface enabling a third party to perform general node monitoring. Facilities include a function to obtain lists of currently installed proxylets.
  - `UK.ac.lancs.CORBA.registry.Registry` - a CORBA interface similar to that used by RMI (Java Remote Method Invocation) registries (`java.rmi.registry.Registry`). Used to keep a record of 'L+' control interfaces exposed by current active applications.
- **Proxylet Environment**: on a given active router there will be many concurrent proxylet environments, each responsible for the management of a single active application. In the current implementation each runs in a separate Java Virtual Machine isolated from the LANode Super Server. Two interfaces, one public, one internal are exposed by the proxylet environment component:
  - `UK.ac.lancs.sdyn.Proxylet` - a CORBA interface providing methods to commence and terminate active code execution.
  - `UK.ac.lancs.CORBA.registry.Registry` - a local version of the interface exposed by the LANode SuperServer. Used to register the 'L+' control interfaces exposed by the proxylet.
- **Naming Service**: a database residing locally on an Active Router node responsible for storing 'L+' interfaces currently exposed by active applications executing on the node, and their respective object reference mapping. In the current implementation this is based on a CORBA Naming Service, which although has been

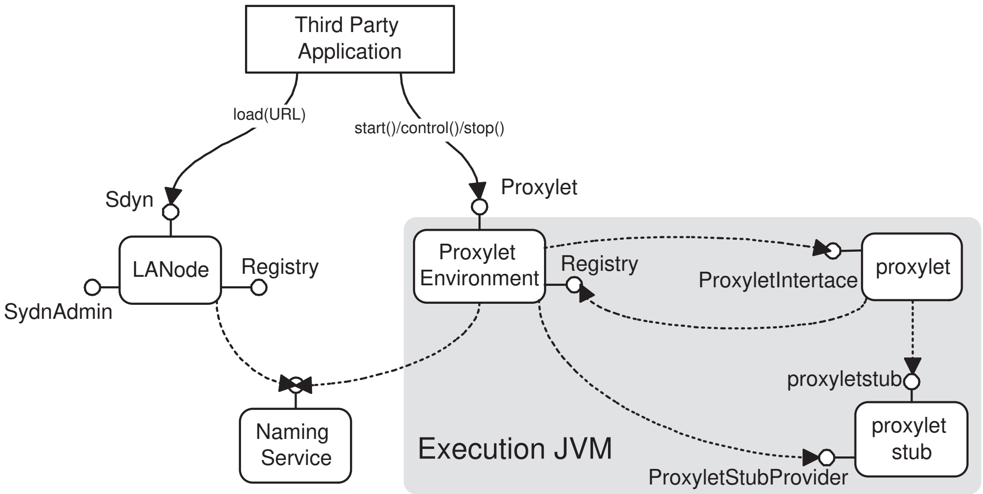


Figure 6.4: Active Router Component Architecture

shown to scale poorly as a global name resolution system, is considered appropriate for storage of small-scale locally significant (per-node) knowledge.

- **Proxylet**: is the active application code, which in the implementation is written in the Java programming language. A user proxylet is created by extending the class `UK.ac.lancs.proxylet.Proxylet` (or for compatibility with the FunnelWeb EEP extending `AU.edu.uts.proxylet`) with an active service implementation.

- **Proxylet Stub**: modelled on the notion of Applet Stub used by Java Applets, the proxylet stub enables the active application to obtain access to enhanced proxylet specific features. Typically the stub is used by the proxylet to obtain object references to the 'L-' interface objects (e.g. Java object encapsulating the routing table).

## 6.4 Out-of-band Control Interfaces

As defined in Chapter 3, approaches to active and programmable networks can be broken down into two broad categories, *integrated* and *discrete* active networking. The prototype Active Router adopts the *discrete* approach, with an out-of-band remote-procedure-call (RPC) mechanism for installing, managing and withdrawing active services. The alternative *integrated* approach would have been quite unsuitable given the underlying node technologies and the byte size of average active applications. *Integrated* approaches often assume regular active service code injection, but this would offer a very high service installation latency given the prototype's per-service virtual machine design.

Pragmatically, the *discrete* approach is easier to engineer on current "off-the-shelf" commercial devices. Virtually all modern Operating Systems include as standard an IP network protocol stack, which enables the ubiquitous TCP/IP protocol to be deployed for service control. TCP/IP provides its own mechanisms for node and transport port addressing and reliability through packet re-ordering, payload checksums and retransmission. The alternative *integrated* approach cannot adopt current freely available protocol mechanisms and instead requires the design, implementation and installation of new proprietary protocol stack modules into the kernel.

### 6.4.1 Middleware Signalling Infrastructure

It is a central part of the prototype Active Router design that control and management of the node and associated active applications can be made by dislocated third parties. Since these are physically remote from the Active Routers themselves, a remote management mechanism is required. As has been explored in earlier chapters in this thesis, there is a multitude of methods by which this may be done but the general consensus has gravitated towards the use of a distributed middleware platform. Potential platforms include the OMG's CORBA architecture, the Java language's own built-in RMI (Remote Method Invocation), or Microsoft's DCOM. The use of DCOM may be quickly rejected for the design of an Active Router, since the platforms for which stable implementations currently exist are aimed at the terminal market. Java RMI is an obvious alternative since it is built into all modern Java Virtual Machines (since JDK version 1.1) and is hence supported on virtually all major platforms (including Solaris, Linux, BSD, VMS, HP-UX, MS-Windows). Indeed it was selected by UTS for their FunnelWeb implementation of the ALAN platform, which has been a major influence on the Active Router ACP. However, Java RMI has not been selected as the middleware control platform for the Active Router, the reasons for this decision being twofold:

- Language and object model specific: RMI is native to Java and relies on many essential features of the programming language including object serialisation, interface definition and down-loadable object implementations. It is difficult to interwork with other languages, Java provides the Java Native Interface (JNI) to allow Java based code to call native methods. However, this may only be performed locally. To make remote calls a remote proxy object (written in Java) must be co-located with the foreign object; this acts as a proxy through JNI to enable remote inter-working.

- Stub generation: traditionally an interface is defined using an Interface Definition Language (IDL) which can be exported so that both client and server may be implemented independently according to the agreed definition. Client stubs and server skeleton can be automatically generated using an IDL compiler from the IDL file. However, using Java RMI, there is no IDL, and instead the client stubs are generated from the server implementation. This introduces the drawback that extra mechanisms must be introduced so the client can obtain this stub code.

Performance deficits of RMI against the major CORBA implementations could also cause concern in some domains. However, RPC performance is not a major criterion for the Active Router control interface since it is not conceived that there will be a large number of concurrent active application start/stop operations.

After some consideration CORBA/IIOP (Internet Inter-ORB Protocol) was selected as the middleware platform, and the Java binding of Orbacus v3.2 from Object Orientated Concepts (Object Oriented Concepts Inc, n.d.) was chosen as the default CORBA 2 compliant ORB for the Active Router platform. Orbacus is used to implemented the Active Router control interfaces (`UK.ac.lancs.sdyn.Sdyn` and `UK.ac.lancs.sdyn.Proxylet`) and may optionally be used by the proxylets to expose additional interfaces at the 'L+' P.1520 reference point.

An important tenet of the Alpine framework is the addition of the Signalling Transport Service Provider function (Banfield *et al.*, 2000b) into the P.1520 Reference Model (see section 5.6). The effect of this addition is to separate out the role of signalling, and provision it independently. Although in some network architectures this may be brought to fruition through the provision of a separate signalling network, in this Internet based scenario it is more appropriate to construct a virtual signalling network over the underlying public infrastructure. This detail may be hidden from the active service components through the use of the security abstraction provided by the middleware platform. The CORBA Security Service (OMG, 2000a) is one of the general (non-domain specific) CORBA Services, and the OMG has defined a range of security levels that an ORB may support. Known as Common Secure Interoperability (CSI), security is defined as ranging from CSI level 0 through to CSI level 2. According to the OMG definition "an ORB conforming to CSI level 2 must support all the security functionality in the CORBA security specification". At present only the lowest security level (CSI level 0) can be supported in the prototype implementation through the adoption of the SSL protocol plug-in for Orbacus.

The official OMG specification defines CSI level 0 security as:

> "At this level, only the identity (no other attributes) of the initiating principal is transmitted from the client to the target, and this cannot be delegated to further objects. If further objects are called, the identity will be that of the intermediate object, not the initiator of the chain of object calls"

Although lacking sophisticated features such as the ability to differentiate access control beyond the level of client identity (e.g. separate access and audit identities), the approach is flexible enough to construct virtual private signalling networks (VPSN). The additional delegation functionality available in CSI level 1 and 2 compatible ORBs, allowing the identity of the operation originator to be carried through the chain, would complicate the configuration of the VPSN and increase the likelihood of the VPSN integrity being broken.

### 6.4.2 Control Interface interaction model

Three agents are involved in the basic operation to establish and withdraw the execution of an active application on an Active Router. These are the third party user (client), the LANode Super Server, and the Proxylet Environment; to give a clearer understanding of the relation between these three elements, a UML Interaction diagram of the component intercourse across time is shown figure 6.5. For more details on the actual interface definitions please refer to figures 6.6 and 6.7, but the basic control interaction over an active components lifetime is:

1. Client passes URI (Uniform Resource Identifier) to Super Server by invoking (over CORBA/IIOP) the `load()` method on the `UK.ac.lancs.proxylet.Sdyn` interface.

2. The Super Server downloads over HTTP the document at the given URI. Under normal circumstances this should be a CCML document (see section 6.5), which will be parsed to obtain the URI of the active service implementation.

3. The active application is downloaded over the HTTP protocol.

4. The Super Server spawns a new virtual machine to run a new instance of the Proxylet Environment for this new application. Various configuration details are passed to the new Proxylet Environment over STDIN (*Standard-In*).

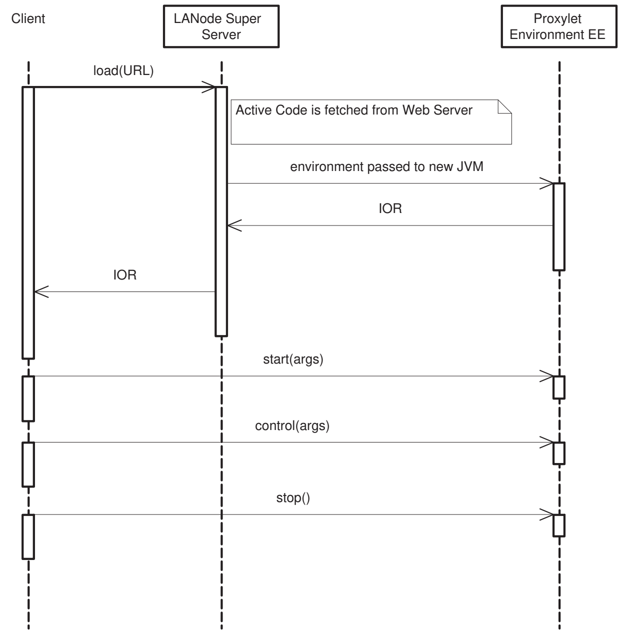


Figure 6.5: Interaction between control elements

5. Once the Proxylet Environment has been configured, then the IOR (Interoperable Object Reference) of the exposed proxylet control interface (`UK.ac.lancs.proxylet.Proxylet`) is passed back to the Super Server over STDOUT (*Standard-Out*), and returned to the initiating client over CORBA/IIOP as the return value to the `load(URI)` call.

6. Now the client can make requests directly to the Proxylet Environment running the active application.

The types of control operations exposed by an active application will be specific to each usage scenario. However, there are some generic features that are applicable to all active applications. These are found in the `UK.ac.lancs.proxylet.Proxylet` interface exposed by the Proxylet Environment.

In addition to the obvious `start()` operation, two extra methods, `control()` and `stop()`, are included. The `control()` operation is now deprecated. It is maintained to provide execution environment compatibility with proxylets written for the FunnelWeb EE, where it was used to pass configuration information. Within the Alpine framework, this configuration information is passed over interfaces exposed directly by the active application at the P.1520 'L+' reference point. This enables more sophisticated configuration information to be passed than using the flat data-structure of the string data-type (used by the `control()` method).

```
/**
   * A DPS has this CORBA interface.
   *
   * @see <A HREF="http://www.comp.lancs.ac.uk/computing/research/mpg/
                          projects/pan/apis/Sdyn.idl">Sdyn IDL</A>
   **/
  interface Sdyn {
    /**
     * Load a proxylet from a URI.
     *
     * @param code the URI of a proxylet
     *
     * @return a reference to the ready-to-run proxylet
     **/
    Proxylet load(in string code) raises (InstallationException);

    /**
     * Access the registry of proxylet-provided services.
     **/
    readonly attribute CORBA::registry::Registry services;
  };
```

Figure 6.6: `UK.ac.lancs.proxylet.Sdyn` interface exposed by LANode Super Server

It is interesting to note that the `stop()` operation is also exposed by the Proxylet Environment, when this could easily be delegated to the active application's 'L+' interfaces. However, this would be a potentially dangerous design, as it would put full responsibility for the withdrawal of the proxylet into the hand of the proxylet itself. Proxylets could easily be engineered that avoid being killed by third parties. By making the deactivation of a proxylet the responsibility of the Proxylet Environment, this issue is avoided.

## 6.5 The Component Compatibility Mark-up Language

When the initial prototype Active Router was designed, it was assumed that it would be possible to capture a single abstract view of an IP router. This would lead to a homogenous design for all Active Routers, and active applications could be easily written to run on all Active Router nodes.

Further study for this thesis into router designs has shown that this was a false assumption. There are many different abstraction details that vary between different data-link technologies and equipment manufacturers. Should open programmability

```
/**
   * A proxylet can be controlled through this interface.
   *
   * @see <A HREF="http://www.comp.lancs.ac.uk/computing/research/mpg/
                          projects/pan/apis/Sdyn.idl">Sdyn IDL</A>
   **/
  interface Proxylet {
    /**
     * Start running the proxylet. The method returns only after the
     * proxylet is initialised.
     **/
    void start(in Strings args) raises (ExecutionException);

    /**
     * Send an additional text message to a running proxylet.
     **/
    void control(in string con) raises (ExecutionException);

    /**
     * Stop a running proxylet.
     **/
    void stop() raises (ExecutionException);
  };
```

Figure 6.7: `UK.ac.lancs.proxylet.Proxylet` interface exposed by Proxylet Environment

reach commercial routers, there is no reason to expect, nor currently any forum to develop, a standardised programmable API for active nodes.

Given the assumption of heterogeneous Active Routers, a mechanism is required to place the compatible implementations of an active service on suitable Active Router nodes. The most obvious solution is a mechanism whereby the client (active service initiator) interrogates the chosen active nodes on their supported programmable abstraction (e.g. do they support a Cisco or Nortel environment). Although this would have been simpler to implement, for the Alpine prototype Active Router an alternative approach was chosen based upon the Component Compatibility Mark-up Language (CCML) (Simpson *et al.*, 2000).

CCML is an XML (eXtended Mark-up Language) derived language for agent-driven content negotiation. It was developed solely for the prototype Active Router, but has since been enhanced to apply to other domains. The use of an XML based language is not new to the active networking community: an XML based scheme has already been suggested for defining active code behavioural specifications (McKee & Marshall, 1999).

```
<!ELEMENT CCML (SELECT*)>
<!ELEMENT SELECT (CHOICE+)>
<!ELEMENT CHOICE (PROPERTY*)>
<!ATTLIST CHOICE URI CDATA #IMPLIED>
<!ELEMENT PROPERTY EMPTY>
<!ATTLIST PROPERTY NAME CDATA #REQUIRED
                   VALUE CDATA #REQUIRED>
```

Figure 6.8: CCML v0.1 DTD

With CCML, an active application is broken down into components, each component having multiple implementations each possibly with different resource requirements. Typical resource requirements would be a language interpreter (Java, TCL, Caml etc.), node architecture (Intel x86, SPARC etc.) and required programming API ('L-' interface).

The reason behind the adoption of the CCML-based approach is that it is the local node that has the best knowledge of its own properties. Only the local node can make the most appropriate decision as to which collection of components to instantiate to perform the active service.

### 6.5.1 CCML syntax

The syntax of CCML is defined by an XML DTD(Document Type Definition), the CCML v0.1[1] DTD being provided in figure 6.8. Since the initial DTD, CCML has evolved[2] towards a C preprocessor like structure providing a more powerful generic set of features that may be applied to general software installation, stylesheet selection, and layered multimedia delivery.

As an XML document, a CCML 0.1 document should begin with a DOCTYPE, to allow the parser to detect the appropriate DTD. CCML v0.1 documents begin with the following line:

```
<!DOCTYPE CCML SYSTEM "http://www.comp.lancs.ac.uk/computing/research/mpg/
projects/pan/ccml.dtd">
```

The top-level element of all CCML documents is the CCML tag, which has no attributes.

[1] version 0.1 co-developed with Steven Simpson

[2] later generic versions(Simpson, 2000) which are applicable beyond the Active Network scenario were developed by Steven Simpson

Each abstract component is described by a SELECT element. This has no attributes, but contains several CHOICE elements. In general for the purposes of the prototype Active Router, there will be three abstract components and therefore three select clauses. The first is for the Active Data Plane (ADP) component, the second for a veneer component to interface between the ACP and ADP, and the third for the choice of Active Control Plane (ACP) component. The order of these three abstract component is not defined.

Each SELECT component consists of a group of CHOICE elements, defining the choice of concrete components. Each CHOICE element may contain a list of (space separated) URIs from which the concrete component can be obtained. To enable the decision as to which CHOICE clause to select, each CHOICE clause may contain a set of PROPERTY elements. Each PROPERTY element holds a NAME and VALUE attribute which represents a single requirement. An example of a valid CCML v0.1 document is shown in figure 6.9.

### 6.5.2 CCML semantics

Given a CCML document, a node that knows its own characteristics (each of which is named) can choose a subset of all the URIs listed in the document, such that the resources at the selected URIs form a complete program that will run on that system without compatibility problems.

A node that is sent a CCML document should select exactly one CHOICE element (based on its contents) from each SELECT element, or report failure to the submitter. All the URIs from the selected choices form the resultant list. If a URI is relative, it should be resolved using the context of the CCML document (assuming that was obtained from a URI).

A CHOICE element is eliminated from the selection if any of its contained expression elements (only PROPERTY is allowed in 0.1) are found to be false with respect to the system's characteristics/properties.

After this elimination, if more than one choice remains, all but the last are eliminated. This implies that more specific/optimised choices within the same selection should appear later. If none remain, the software is deemed incompatible with the node.

### 6.5.3 CCML Parser implementation

A set of tools consisting of a collection of Java classes in the package `UK.ac.lancs.ccml` has been developed. This toolkit used by the prototype Active Router allows CCML

```
<!DOCTYPE CCML SYSTEM
    "http://www.comp.lancs.ac.uk/computing/research/mpg/
            projects/pan/ccml.dtd">
<!-- A proxylet requiring three components (one per SELECT):
       * the platform-specific ADP,
       * a veneer between the ADP and the ACP,
       * the platform-independent ACP. -->
<CCML>
  <SELECT>
    <!-- ADP component -->
    <!-- this component is allowed to be empty -->
    <CHOICE/>

    <CHOICE URI="library\ADP\intel.jar">
      <PROPERTY NAME="capsule" VALUE="linux-module" />
      <PROPERTY NAME="arch" VALUE="i386,i486,i586,i686" />
    </CHOICE>

    <CHOICE URI="library\ADP\sparc.jar">
      <PROPERTY NAME="capsule" VALUE="linux-module" />
      <PROPERTY NAME="arch" VALUE="sun4" />
    </CHOICE>

    <CHOICE URI="library\ADP\telefonicaNode.jar">
      <PROPERTY NAME="capsule" VALUE="telefonica" />
    </CHOICE>
  </SELECT>

  <SELECT>
    <!-- ACP/ADP veneer component -->
    <!-- this component is allowed to be empty -->
    <CHOICE/>

    <CHOICE URI="library\ACP\linux.jar">
      <PROPERTY NAME="profile" VALUE="UK.ac.lancs.linuxdevice.Kernel" />
    </CHOICE>

    <CHOICE URI="library\ACP\telefonicaNode.jar">
      <PROPERTY NAME="profile" VALUE="ES.telefonica.AOS" />
    </CHOICE>
  </SELECT>

  <SELECT>
    <!-- ACP component -->
    <!-- this component is platform-independent -->
    <CHOICE URI="library\ACP\my_activeApp.jar" />
  </SELECT>
</CCML>
```

Figure 6.9: Example CCML v0.1 document

documents to be obtained (over HTTP), the document parsed (against a list of supported properties), and a list of resources (URIs) returned for the chosen concrete components.

Obviously, the highest parsing performance could be achieved through the implementation of a dedicated CCML parser. However, this would have constrained the evolution of the language, each new version possibly requiring major parser re-engineering. Instead, a generic XML parser is used by the CCML toolkit based upon the standard SAX v1.0 (Megginson Technologies, 1998) parser API. This standard gives the flexibility to swap between XML parsers (such as high performance verses memory conservative parsers) without making changes to the application code.

The prototype Active Router makes use of the Sun Microsystems Java API for XML Parsing (JAXP) package based upon their high performance Java Project X parser.

## 6.6 The Active Node 'L-' API

Although not recognised by P.1520, it has already been stated in this thesis that the active node base API corresponds to the 'L-' interface reference point. It has also been ascertained that it is not possible to build a single 'L-' abstraction for all types of active routing devices. On many active routers, the supported 'L-' abstraction will be fixed, defined when the manufacturer designed its particular node architecture. There may be other nodes, of which the prototype Active Router is an example, which allow the 'L-' programmable abstraction to be reconfigured either at configuration or at runtime. The purpose of the prototype's Active Data Plane (ADP) based upon a modified Linux kernel is to allow new packet forwarding behaviour/functionality exposing alternative 'L-' abstractions to be dynamically inserted without requiring a reboot.

Since these components implementing new forwarding behaviour are inserted into the kernel they lack many of the basic features available to user-space applications. Of particular importance is the lack of readily available inter-process communication (IPC) mechanisms required to expose the 'L-' interface to the Active Control Plane (ACP).

The solution in the current implementation has been to introduce user to kernel space communication via a virtual device. Such a solution is applicable to nodes built on Unix derived operating systems such as Linux, where everything is mapped to a file, be it an output port or a process.

The ACP environment located in user-space can perform standard read and write file handling operations on the virtual device to communicate with the behaviour abstraction. Unfortunately, a virtual device and the corresponding file handling operations

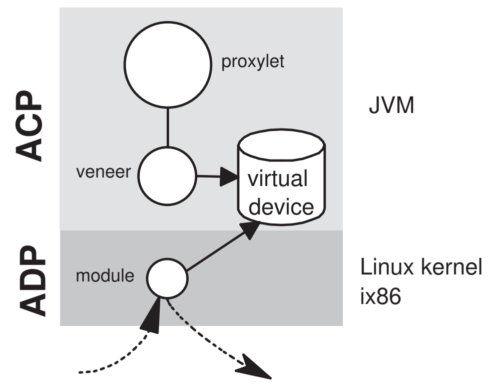


Figure 6.10: Veneer Component

do not offer a particularly user-friendly programmable abstraction. All data structures passed between user and kernel space must be flattened so they can be written/read from the virtual device. From the viewpoint of the active application programmer who is developing code to reside in the ACP, it would be preferable to have the programmable abstraction modelled in terms of objects, a solution more in line with the Java programming language.

To achieve this a veneer component (figure 6.10) is introduced, similar to a CORBA stub. This is a Java object, whose role is to provide a useable object-orientated 'L-' abstraction of the router functionality. Veneer objects flatten data-structures passed over the object oriented abstraction so that they may be transmitted over the virtual device. At present, these veneer objects must be hand-coded, but it is feasible that tools could be developed so they could be automatically generated, similar to the generation of stubs by an IDL compiler.

A proxylet that wishes to use a given programmable abstraction must obtain the object reference of the corresponding veneer object. The prototype Active Router makes this possible through the Proxylet Stub, a feature added to enable access by a proxylet to local resources. A method `getProfile(String)` is provided on the stub, allowing the proxylet to obtain the object reference of the veneer component simply by passing the URN of that particular programmable abstraction.

Abstractions exposed by the 'L-' API are classified into three types based on a classify, compute and schedule model. This simple model forms a generic skeleton upon which any router technology can be constructed. The role of each of these functions is detailed further as follows:

- Classify: the most fundamental feature of a router is to support a function to distinguish between packets in an input stream. Once individual packets and flows have been identified they can be treated correspondingly by the compute and schedule functions. In a basic best-effort IP router, the classifier is used to sort packets into flows, each packet in the flow sharing the common characteristic of being bound for the same next hop. There are several contending packet classifier APIs which are currently used to build packet analysis tools (e.g. tcpdump) and low performance firewalls. Amongst the most well known and expressively powerful is the Berkeley Packet Filter (BPF) (McCanne & Jacobson, 1993), supported in most modern UNIX derived operating systems. Packet filters are written in an assembler type language supporting basic load, store, arithmetic, and branch instructions, allowing the construction of control flow filter graphs (CFP) that run in a pseudo-machine.

- Compute: is the most generic of the three abstraction classifications, and captures a very wide range of router functions that are not simply associated with the receipt and transmission of packets. Typical compute functions are packet marking, tunnelling, and routing. For example, an active application deployed on an access router to a Differentiated Service domain can use the API to configure the stamping of priority classes into the TOS field of IPv4 packet headers. While many compute features will vary between routers, it is likely that most will at least support a routing table manipulation interface. Since the aim is to be compliant with IEEE P.1520, the draft "L- Interface for Routing Control" (Miki *et al.*, 1999) has been adopted, and is implemented in the package `org.ieee.p1520.ipswg.routing`. A veneer component has been implemented exposing the interface in figure 6.11.

- Schedule: advanced routers supporting some form of predictable Quality of Service behaviour are distinguished from traditional best-effort routers by their support for multiple priority output queues per output port. There are many possible scheduling algorithms that may be inserted into the ADP kernel on the prototype Active Router. The purpose of the scheduler programmable abstraction is to allow a proxylet to influence priority allocation of the given ADP scheduling function.

## 6.7 Summary

The realisation of the Alpine active programmable platform described in this chapter does not attempt to provide a complete implementation of the framework previously

```
interface RoutingTable {
  struct Route {
    long key;
    string DestID, NextHop, NetMask, InterfaceName;
    short Metric, Timer, Flag;
  };

  typedef sequence<Route> RouteTable;
  void getRoutingTable(out RouteTable rt);
  boolean getRoute(in long key, out Route route);

  /** Add an entry to the routing table. **/
  long addRoute(in long key, in string DestID, in string NextHop,
                in string NetMask, in string InterfaceName,
                in short Metric, in short Timer, in short Flag);

  /** Remove an entry from the routing table.**/
  void removeRoute(in long key);

  boolean getDestID(in long key, out string DestID);
  void setDestID(in long key, in string DestID);

  boolean getNextHop(in long key, out string NextHop);
  void setNextHop(in long key, in string NextHop);

  boolean getNetMask(in long key, out string NetMask);
  void setNetMask(in long key, in string NetMask);

  boolean getInterfaceName(in long key, out string InterfaceName);
  void setInterfaceName(in long key, in string InterfaceName);

  boolean getMetric(in long key, out short Metric);
  void setMetric(in long key, in short Metric);

  boolean getTimer(in long key, out short Timer);
  void setTimer(in long key, in short Timer);

  boolean getFlag(in long key, out short Flag);
  void setFlag(in long key, in short Flag);

  exception EntryNotFound { };
};
```

Figure 6.11: P.1520 Routing Table ‘L-’ abstraction

described in chapters 4 and 5. Service, Network, and Element layer programmability are considered by the framework, but it is only the programmable network element that has been selected for implementation. The network element forms the foundation for the introduction of active and programmable technology into the network, and is the natural place to build an experimental platform.

Given that the purpose of the platform was to experiment with element programmability, and the wealth of experience already obtained within the research group at Lancaster University, it was not necessary to build a new platform from scratch. Instead, two important state-of-the-art active nodes have been adopted as input into the design of the programmable node. In section 6.2.1, the ALAN (Application Layer Active Network) system from UTS was discussed, a platform which, although it has not contributed code to the Alpine programmable platform, has had a great influence on its design. The shortcomings of the ALAN platform were noted, and a quite different usage scenario is adopted for the application layer technology in the Alpine ACP (Active Control Plane) environment. The second influence introduced in section 6.2.2 is the LARA (Lancaster Active Router Architecture). LARA has been adopted as the ADP (Active Data Plane) environment, which is used to build programmable components on the Alpine node.

In section 6.3, the detailed design of the node was explained, justifying the decision to split control and data-plane programming. In section 6.3.2, the component architecture of the ACP environment was summarised.

The implementation of AMIE, the middleware infrastructure already introduced in chapter 5, was discussed in section 6.4. This includes the adoption of CORBA as the chosen RPC mechanism for the prototype platform. This provides an industrial strength codebase to build the implementation on, and on which the associated CORBA Security Service provides an excellent basis to construct a Virtual Private Signalling Network (VPSN).

Resulting from the implementation, it was discovered that it was not feasible to define a single programmable abstraction that may be supported by all nodes. Heterogeneity amongst layer-3 equipment functions (best-effort routers, IntServ/DiffServ routers, firewalls), hardware designs, and non co-operation amongst equipment vendors makes a single API unlikely. In section 6.5, CCML (Component Compatibility Mark-up Language), an XML derived language for content negotiation, was developed. This enables the Alpine node to select component implementations based on local knowledge, which are compatible with the abstractions supported by that particular node.

Finally, section 6.6 discussed the design of the 'L-' active network API, and the method by which the active node obtains access to the programmability embedded into the ADP environment. An example of a P.1520 compliant 'L-' interface (as defined in P1520/TS/IP010) for route control was presented in figure 6.11.

# Chapter 7

# Evaluation

This, the penultimate chapter, presents an evaluation of the Alpine framework and the corresponding middleware and programmable node implementations as described in chapters 5 and 6 of this thesis. While the goal of the framework is to enable rapid service creation, it has not been feasible to conduct practical experiments to introduce new services into an operator's network. It is obviously not practically possible to obtain access to install experimental elements in operational wide area networks. Instead, the approach has been to validate the flexibility of the architecture according to three criteria: ability to introduce heterogeneous services, scalable performance of the node architecture, and appropriateness of the programmable metaphor.

## 7.1 Methods of evaluation

There are in general two approaches taken to the evaluation of a work such as that presented in this thesis, namely quantitative and qualitative. Neither is exclusively suited to the evaluation of the Alpine programmable platform and middleware infrastructure, so a combination of the two approaches has been employed. Initially a qualitative approach is taken to evaluate the abstractions and architectural features of the platform. This is predominantly a functional evaluation, and to aid this a simple case study scenario is introduced.

It has been feasible to implement only a subset of the Alpine architecture, making the qualitative approach the most appropriate evaluation technique. However, it is important that an in-depth quantitative evaluation is also undertaken to evaluate the performance of programmable nodes. When considering this evaluation it must be remembered that the implementation is a prototype, and performance is unlikely to be as high as could be realised from a commercial product. Nevertheless, if the demonstrator

| Requirement No: | Requirement statement | Successfully met |
|---|---|---|
| A.1 | Open access to layer-3 functionality | Yes |
| A.2 | Rapid creation of network services by third party (network operator) | Yes |
| A.3 | Seamless compatibility with current IP infrastructure | Yes |
| A.4 | Requirement not to alter current IP applications in order to receive value of active services | Yes |
| A.5 | Build active services composed of composite active applications | Yes |
| A.6 | Interface between active services and legacy services | Yes |
| B.1 | P.1520.3 Interface Model compliant | Yes |
| B.2 | Compliance with DARPA Active Node architecture | No |
| C.1 | Supports the construction of "My Own Network" | Yes |
| C.2 | Built using off-the-shelf components and software | Yes |
| C.3 | Secure execution of active code | Partial |
| C.4 | Support for Alpine Lifecycle | Partial |
| C.5 | Hide complexity of network infrastructure | Yes |
| C.6 | Supports creation of VPSN | Yes |
| C.7 | Differentiates between different programmable abstractions | Yes |
| C.8 | *Discrete* approach to active networking | Yes |

Table 7.1: Alpine Requirements

prototype node can be shown to perform adequately then we can be reasonably assured that the design is legitimate.

## 7.2 Requirements fulfilment

This section identifies the central facets of the Alpine architecture and maps these onto requirements that must be met by the implementation of the prototype Active Router and middleware infrastructure. These are presented, along with the success or failure to comply with these requirements, in table 7.1. However, this list of requirements is not alone adequate and is complemented by the following discussion.

The requirement list has been broken down into three main sections to recognise the distinct aims of the prototype. Requirements A.1 to A.5 consider the features necessary in order to compose services within the Alpine framework. Firstly, requirement A.1 mandates that the central purpose of the programmable node is to provide a mechanism for layer-3 programmability. Along with this, requirement A.2 states that this programmability must be provided to third parties (i.e. not just the equipment vendor). Undeniably both these requirements are met by the prototype node through the adoption of active network technology, enabling the upload of code to an Execution Environment (EE) based on a Java Virtual Machine (JVM).

Unlike many experimental active nodes whose programmability resides fully in user-space, the design meets both requirements A.3 and A.4. For example, the ALAN architecture (Fry & Ghosh, 1999; Fry & Ghosh, 1998), and resulting FunnelWeb implementation that have been a major influence on this design, requires that the end-user applications are modified to address those IP packets requiring active processing directly to the node running the active service. Other platforms necessitate the use of purpose built active network protocols (e.g. ANEP(Braden *et al.*, 1997a)) to mark packets for active processing.

Finally in regard to service composition we have requirements A.5 and A.6, both of which are answered through the use of middleware technology. The choice of middleware platform is not mandated but in the case of the prototype a CORBA v2 compliant ORB was selected.

The next set of requirements relates to compliance with current standardisation. At present, there are no programmable and active network standards, so instead two quasi-standard reference models are selected. The design of the ACP (Active Control Plane) and the separation between Execution Environment and programmable abstraction have led to a node architecture in the spirit of the P.1520 reference model. At present the IP work-group (P.1520.3) have not yet standardised their interface descriptions, and have recently adopted a new three-layer abstraction model(Vicente & et al, 2000). Although it was not the aim to comply with this emerging model, it has recently transpired that both architectures have developed remarkably in parallel. The addition of the newly created P.1520.3 Base Building Block (BBB) component is remarkably similar to the super classes used to create veneer components on the prototype programmable router.

A mapping between the prototype router and the emerging P.1520.3 three-layer model is thus:

Active Application (providing 'L+' interface) = Service Specific Building Block
Veneer Component (provides 'L-' interface) = Resource Building Block
Veneer Component Superclasses = Base Building Block

To enable wide-scale experimentation, it was an objective that the prototype node be compliant with the DARPA active node architecture, and therefore active router nodes could have been incorporated into the international ABone (Berson, 2000) active network testbed. However, the DARPA active node architecture has proved to be too restrictive, and therefore requirement B.2 has not been met. The divisions between Active Application (AA), Execution Environment (EE), and Node OS are not compatible with the richer Alpine component model. The ABone approach of installing a new EE, to exhibit new security and functionality on an active node, is an extremely heavyweight and coarse-grained procedure for programmability differentiation. Instead, a clear distinction is made within Alpine and the prototype router, between an EE virtual machine type (Java, Caml, TCL), and the active API ('L-' interface) which is independent of EE type. A detailed discussion of this can be found in section 5.5.

The ANEP (Braden *et al.*, 1997a) protocol used for active packet encapsulation in the ABone is designed for *integrated approach* AN systems. The purpose of ANEP is to enable de-multiplexing of active packets to the relevant EE based upon a 16-bit type identifier. Not only is this inflexible (see section 5.5), restricting the evolution of EEs, but it lacks a mechanism to distinguish to which "My Own Network" a packet belongs (Requirement C.1); only supporting de-multiplexing up to the EE layer.

The final category of requirements, C.1 to C.8, considers those requirements directly imposed upon the Active Router. Requirement C.1, that the node must support the construction of a "My Own Network[1]" (MON) simply obliges the active router to distinguish packets for active processing, so that only those that have purchased the service may benefit from it. This requirement is met through the use of a Virtual Machine sandbox, and the provision of 'L-' APIs tailored to the security privileges of the MON.

The active router satisfies requirement C.2 in that it has been built using standard PC hardware, the freely available Linux operating system, and an off-the-shelf Java Virtual Machine. Full consideration of security measures as stipulated in requirement C.3 is only partially complete. Some of the security restrictions normally imposed on active JVMs

[1]requirement adopted from EURESCOM P926

have had to be relaxed to provide a mechanism for inter ACP-ADP communication (user-space to kernel). However, since the programmability of the active router is aimed at the network operator and trusted peers, implementing the full security measures required for end-user initiated code injection is not a high priority.

At present full support for the Alpine Lifecycle model as defined by requirement C.4 is incomplete. The implementation of the Active Router currently supports the creation, availability, deployment, execution, and withdrawal stages. However, since the implementation has considered the realisation of a prototype Active Router only, the subcontraction and associated deactivation stages (which relate to outsourcing service functionality to the AIP) are irrelevant.

The definition of the role of the Alpine Middleware Inter-networking Environment (AMIE) has enabled the active platform to meet requirements C.5 and C.6. Next, requirement C.7, that states that code may be written to a heterogeneous range of programmable abstractions, is supported through the separation of EE virtual machine type and 'L-' API. This is resolved at the active node by the CCML requirements meta-language. Finally, by supporting the control interaction model described in section 6.4.2, requirement C.8 is satisfied. This allows out-of-band active code upload and lifecycle control by third parties, most likely the network operator.

## 7.3 Qualitative Evaluation

This section presents a qualitative evaluation of the Alpine framework model. While the quantitative evaluation will consider results obtained from the prototype implementation, here design issues are considered that go beyond the limited scope of implementation.

### 7.3.1 Case Study Evaluation Scenario

To reflect on the applicability of the Alpine framework, an example case study is introduced to consider the mechanism for introducing new services. This case study considers the provision of enhanced quality of service for telephony (voice) over an IP infrastructure. Traditionally the voice communication service has been provided on a purpose built circuit-switched telephony network, providing guaranteed 64kbit/s channels. Recently, particularly in the US, newly established long-distance carriers have emerged, basing their voice communications service on low cost packet-switched IP technology. The guarantee of sufficient QoS for voice calls is made chiefly through the massive over-provisioning of link bandwidth on a private mono-service IP network. The case study presented here considers the use of active programmable technology to provide better network utilisation on a multi-service IP backbone.

Service enhancement features may be added into either of two domains, the service or network layer. In this scenario only extended network layer features are considered, while service layer protocols and mechanisms, e.g. the Session Initiation Protocol (SIP), are the responsibility of the value-added service provider and beyond the scope of the case study.

Standard implementations of the IP layer-3 functions (routing and packet scheduling) are generic, and may not offer optimum service for the IP telephony application. Routing algorithms, for example, are based upon a lowest metric count (normally number of hops) algorithm. There is no reason to presume that the route consisting of fewest hops is the most optimal for the given domain; there could be many other criteria (usage cost, services provided (e.g. multicast)). The adoption of active technology would allow new customised routing algorithms to be deployed; each could be tailored to a particular usage domain.

There is still ongoing debate over the granularity used to define Quality of Service requirements. Solutions range from fine-grained per-flow integrated services (Baker *et al.*, 1997), to the coarse Differentiated Service approach (Blake *et al.*, 1998). Active technology enables bespoke QoS reservation schemes to be deployed, specialised to meet the requirements posed by a particular application. Given our telephony case study, an active mechanism can be deployed to reserve quality approximating to that of a 64kbit/s ISDN circuit.

Active network technology enables rapid reaction to changes in network conditions, for example, in our given scenario (figure 7.1) the inability to sustain QoS guarantee. Typically, such an event would have to be detected at the point of connection termination (either source or sink), and action taken by the application there at the end terminal. Instead, monitoring of QoS guarantee and any necessary corrective action can be undertaken within the network. This may involve the decision to re-route traffic over a less congested path or perhaps forwarding traffic to a transcoder application, to lower bandwidth requirements. The interaction between and behaviour of the various components of the Alpine framework involved in the case study are detailed in Figure 7.2.

This case study illustrates an excellent example of the composite nature of likely active services. The service could not be provided by a single active application (AA)

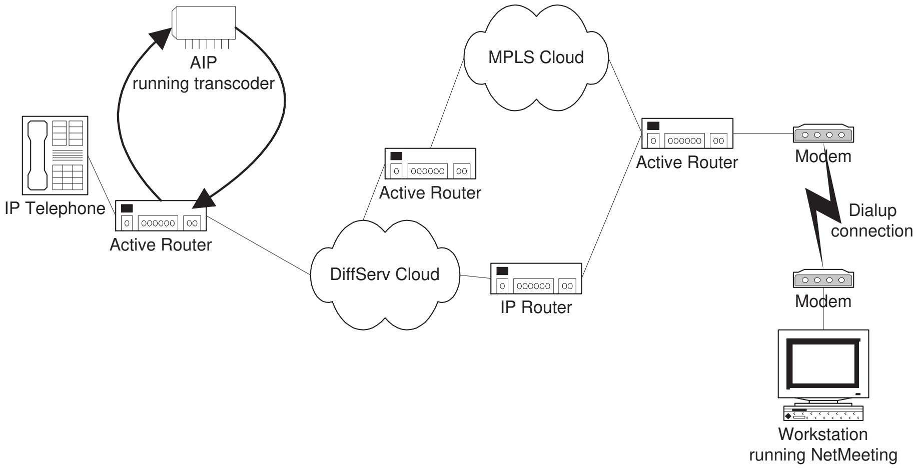


Figure 7.1: IP telephony evaluation scenario

alone, and requires the co-operation of the QoS scheduling AA on the Active Router and the transcoder AA on the active server. It highlights the importance of the Alpine Active Lifecycle, and the introduction in this thesis of the notion of service subcontraction.

### 7.3.2 The Active Service Model

Rather than simply consider the merits of active and programmable network technology, a subject well discussed within the research community, here instead the facets that distinguish the approach from others are considered. There are four main properties that characterise the Alpine platform:

- Composite Active Services
- Operator oriented programmability
- Active programmability in the control-plane
- Middleware support for active service composition

Commonly, Active Services are seen as self-contained entities implemented by an Active Application (AA). These AAs provide what is termed a "store-compute-forward" function.

Such a simplistic model has its advantages; each AA is an atomic component and is not reliant on others for service provision. The value-add algorithm implemented by the AA is self-contained, making it easier to develop, test, and deploy. Since the algorithm runs locally responding to local events/knowledge, the service is more robust, and faster to react (since no latency in receiving accurate information).

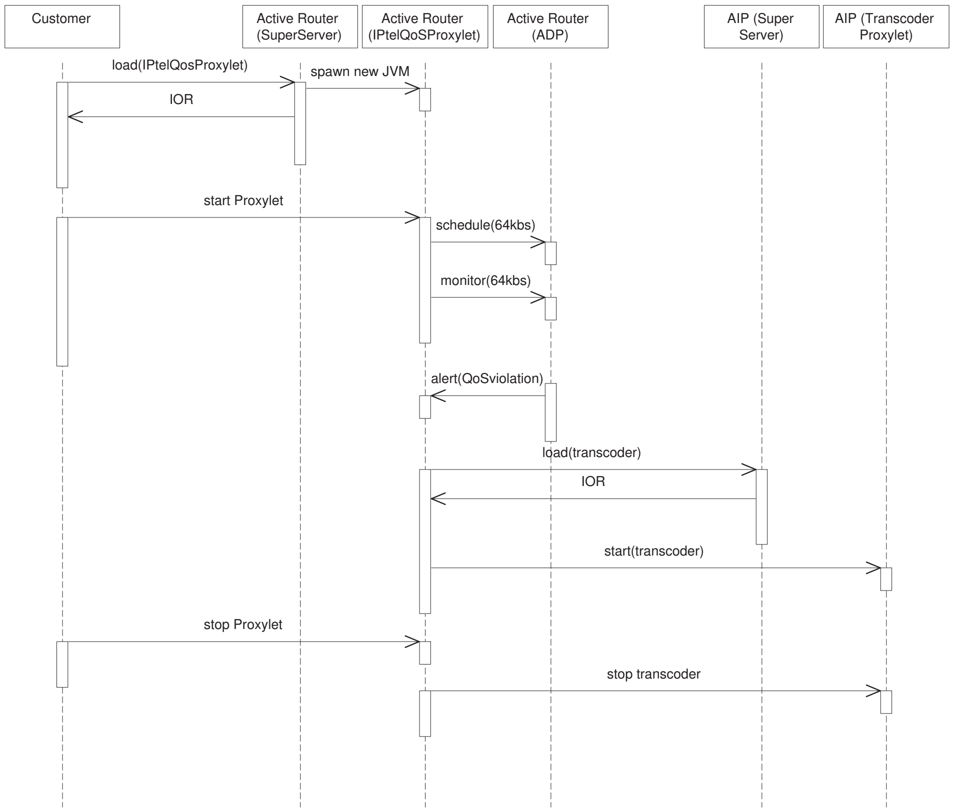


Figure 7.2: Component behavioural interaction in IP telephony evaluation scenario

Despite these attractive advantages, the Alpine framework recognises that more useful value can be created from the composition of many active functions, what is termed composite active services. An example of such as service is active caching (Vanet & Kiriha, 1999; Legedza & Guttag, 1998); isolated caches are now widely deployed within most corporate and campus networks, but a more beneficial caching service can be created through co-operation between these caching applications.

Composite active services are at the core of the Alpine lifecycle model. The introduction of the notion of subcontraction was new in the active networking research community. The ability to subcontract implies the transferring of a partial function of the service to another component (AA) possibly located on a remote node. An example of this activity is given in the case study scenario where the AA responsible for providing the IP telephony QoS service transfers part of this responsibility to a remotely located transcoder AA.

The massive gains in potential value-add by adopting a composite active service model inevitably introduces drawbacks in service resilience and ease of service construction. These are in part answered through the adoption of AMIE 5.2 (Alpine Middleware

Inter-networking Environment), a middleware platform for active networking.

While composite active services have opened the potential for active networking, the decision to tailor for operator oriented programmability could be criticised as a limiting factor. However, at least in the short-run, from a non-technical operations viewpoint, this is a false assumption. Network operators and administrators are very suspicious of the potential security and reliability outcome of introducing third party network programmability into their networks. Alpine is tailored to this audience, giving administrators the ability to flexibly customise, without risk to network integrity.

From a functionality viewpoint, making this programmability available only to trusted parties enables a greater degree of programmability to be exposed. There is inevitably a trade-off between functionality and safety, the ADP kernel employed in the Active Router opting for the former.

Given the peering topology of the public Internet, it is likely that additional end-to-end benefit for the customer may be achieved by distributing the value-add functionality beyond the operators' own networks. This will require operators to agree on active peering and trust relationships between competitors. If the direction towards mergers and alliances persists within the telecommunications industry then this issue could of course become irrelevant. However, this could lead to regulatory issues in an oligopolistic industry dominated by a handful of European and American operators, and thus representing a barrier to market entry. Consider the case of a small independent ISP competing against BT Internet, which is offering advanced services over the BT IP backbone and the Concert alliance member AT&T core IP network. The small independent ISP may be at a competitive disadvantage, being unable to offer advanced services with global reach. It is therefore possible that to ensure a competitive ISP market, operators could be forced to offer this programmability to all, an issue not considered by the Alpine framework.

Unusually for an active node, the prototype Active Router limits exposed programmable functionality to the control-plane. Like most other AN prototypes, the Alpine Active Router is built on general-purpose PC hardware and the Linux operating system. However, it is not presumed that this would be a likely platform for a commercial-grade Active Router. The internal architecture of commercial routers is a closely guarded secret, but the prototype design adheres to what is known within the public domain.

Traditional network devices such as routers use the CPU to perform two tasks simultaneously, the primary task being to forward packets and the other to control the

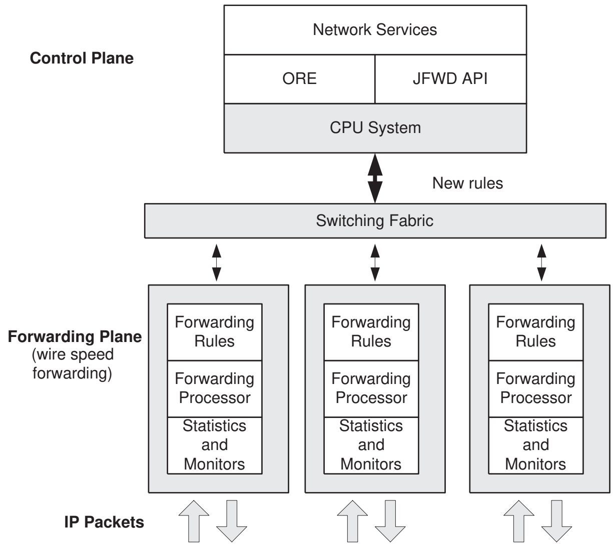


Figure 7.3: Nortel Passport/Accelar Architecture

forwarding policies. The two tasks compete for CPU power, memory and other resources and therefore they cannot obtain maximum performance.

Modern routers are based on dedicated network hardware to achieve gigabit and terabit line rates. New network processors including Intel's IXP (Intel Corporation, n.d.) and Juniper's Internet Processor ASIC (Semeria, n.d.) are about to change the functions performed on routers. To understand the design of modern high performance routers, and evaluate the correctness of the Alpine prototype design, a study has been made of a family of latest generation routers. Due to the openness of the vendor and mutual co-operation through P.1520.3, the Passport/Accelar series routers from Nortel Networks is selected as a design benchmark.

The Nortel Networks Passport/Accelar family of layer-3 routers offers gigabit data rate switching for deployment in access and border locations. They employ distributed ASIC (Application Specific Integrate Circuit) technology coupled with up to a 256 Gbit/s backplane. These are deployed in an architecture shown in figure 7.3, with each ASIC responsible for either four 100 Mbit/s Ethernet ports or one gigabit port. Currently the switches scale to support 384 100BaseT Ethernet or 96 Gigabit ports. A central CPU running control software built on the VxWorks real-time OS completes the architecture.

This design is entirely compatible with the design of the Alpine Active Router. The hardware architecture is similar to that of LARA, the ADP inherited by the prototype. Table 7.2 examines the two hardware architectures[2]. However, these ASICs do not offer

[2]figures for prototype Active Router based on ADP described in (Cardoe *et al.*, 1999)

the same degree of programmability as the general purpose Intel Pentium CPUs, being dedicated to packet scheduling and forwarding. Typically, each ASIC provides a switch specific API, to enable the central processor control of the eight forwarding output queues per port.

This approach is consistent with Alpine, which rejects the original LARA approach of injecting active code onto the forwarding processors. Instead, the LARA programmable kernel in Alpine is used to construct control elements exposed to the ACP (Active Control Plane) through the 'L-' API.

| | Passport/Accelar | Prototype Active Router |
|---|---|---|
| Switch type | Passport 1000 Series | LARA Cerberus H/W |
| Control-plane processor | Power PC | Intel Pentium II 400MHz |
| Control-plane OS | VxWorks | Linux/JVM |
| Control-plane memory | 32MB | 64MB |
| Data-plane processor | Proprietary ASIC | Intel Pentium II 400MHz |
| Forwarding Queues | 8 per port | Unconstrained - software configured |
| Ports | 10/100BaseT and Gigabit Ethernet | 10/100BaseT Ethernet |
| Processors per port | 4 @ 100MBit/s or 1 @ Gigabit line rate | 2 |
| Maximum Number of Ports | 96 100BaseT or 12 Gigabit Ethernet | 4 to 5 (@ 100BaseT) |
| Backplane speed | 5.6 to 256 Gbits/s | 0.5 Gbits/s |

Table 7.2: Nortel Passport/Accelar and Prototype Active Router Hardware comparison

### 7.3.3 Middleware for Active Service provision

A feature unique to Alpine as an active platform is the use of a middleware platform based on distributed remote procedure calls (RPC). It is arguable that active technology and distributed RPC platforms are mutually exclusive (orthogonal); a service can be engineered to use either approach. The notion of the Execution Environment (EE) eliminates the platform heterogeneity problem that most open-standard middleware technologies are designed to solve. Where there is a common runtime-EE, there is no need to mediate between heterogeneous hardware, operating systems, programming languages, or even heterogeneous management approaches.

This argument is indeed true where active services are stand-alone or for composite services assembled from components engineered for the same EE. This scenario is incompatible with the objectives and design of Alpine, where programmability is divided amongst the Active Router, Active Intelligent Peripheral (AIP), Network Operations Service (NOSP), and Value-Added SP (VASP) platforms.

It has already been established in this thesis that a single programmable abstraction is inappropriate. The example given is that an active node designed for active caching requires efficient I/O storage, while for a transcoder AA, high-performance mathematics functions are required from the EE.

Therefore, given the composition of composite services over a heterogeneous and distributed active platform, a middleware platform is required to provide *access* and *location* transparency.

These are defined thus (Blair & Stefani, 1998):

> *Access transparency*: masks differences in access methods when internetworking between objects. To solve this problem it is necessary to deal with potentially different data representations and different invocation mechanisms.
>
> *Location transparency*: hides the physical location of the object in the distributed environment. A logical name (or unique identifier) can be used for the object in place of a physical address.

The other distributed transparencies defined in RM-ODP (ITU-T, 1995b), namely failure, migration, relocation, replication, persistence, and transaction, are not mandatory for Alpine to function correctly, but may provide useful additional benefits. The CORBA v2 middleware architecture and OOC's Orbacus implementation used for the prototype has proved sufficient to provide the mandatory access and location transparency requirements.

The independent and logically separate provision of multimedia and signalling communications is another potentially controversial aspect of the Alpine framework, and requires some validatory remarks. Modern telecommunication networks employ common-channel signalling (CCS), an approach introduced during the 1970's whereby a separate channel is used to carry signalling data. The ReTINA Connection Manager (CM) 3.2 is an application of CCS signalling, where there is a separate signalling network (kTN) based on Ethernet; experimentation with this has taught much about the distinct but diverse requirements of signalling and multimedia data (Banfield *et al.*, 1999).

With the dominance of the Internet, a network that employs the former channel-associated signalling (CAS), this CCS model has become unfashionable. Nevertheless, the CCS approach has properties that lend it to adoption for the Alpine framework. Security is an important property of a signalling system, since if compromised, it has the potential to bring down or violate the entire network. A degree of security is naturally inherent in a CCS approach, whereby access to the signalling network is controlled independently from the data network. Secondly, new active network signalling introduced by the Alpine framework is without impact on the current Internet architecture. This allows signalling to be provisioned without impediment in terms of Quality of Servce (QoS), Quality of Protection (QoP), and name space by the legacy IP infrastructure.

The adoption of a CCS approach does not necessarily imply a physically separate signalling network; the most important properties (i.e. QoP) can be provisioned through virtual private network (VPN) technology. However, as was noted in section 5.6, this separation may also be physical. Signalling and data networks may require different topologies; the data network providing connections between end users, the signalling network providing connectivity terminating at the network operations (NOSP) and value-added services (VASP) platforms.

## 7.4 Quantitative Evaluation

The section presents a quantitative evaluation of the performance of the prototype active router component of the Alpine framework. The overhead imposed by the design and technologies used to engineer the prototype is first discussed. Next, the performance of the signalling mechanism and AMIE middleware architecture is considered.

All measurements presented in this section were taken on the prototype Active Router running on a hardware architecture based on the Intel Pentium II 400MHz CPU, equipped with 64MB of memory. The software platform is divided into two, the Active Data Plane (ADP) and Active Control Plane (ACP). The ADP was based on a general Operating System kernel, in this case a modified Linux 2.2.13 series kernel, although for the purpose of these experiments the control-plane was the main focus. The ACP was developed in Java, and was run on the Blackdown Release Candidate 4 port to Linux of Sun's JDK 1.2.2. For high-performance, native rather than green-thread (emulated in virtual machine) support was used in the JVM running the proxylets. However, due to the JVM being a pre-release version and a number of bugs discovered in the native-thread support, green-threads were used for the Super Server control environment.

The Alpine middleware platform, AMIE, was provided by the Java binding of the Orbacus v3.2 ORB from Object Oriented Concepts (OOC). This ORB was selected because of its low cost (it is free for non-commercial use), IIOP compatibility, and good security support (SSL (Secure Socket Layer)).

### 7.4.1 Active Router Performance

This section considers the performance of the Active Router prototype implementation according to three criteria: latency of service instantiation, control and data-plane division, and the internal-device loop.

Although not requiring code execution on a packet basis as inherent in the *integrated* approach to active networking, the latency of service instantiation still impacts upon the design of the prototype. High latency restrains the ability to react to service demands, and may result in expensive computation at the active router. The Alpine lifecycle divides service instantiation into two states: deployed and executing, the transformation triggered by the `UK.ac.lancs.proxylet.Sdyn.load()` and `UK.ac.lancs.proxylet.Proxylet.start()` methods. Service instantiation latency may therefore be introduced at the `load()` by the requirement for the node to download and parse a CCML document.

The CCML document is parsed by an XML parser in respect to the CCML DTD (Simpson, 2000), and decisions are made as to the correct version to acquire and execute based on knowledge (properties) at the local node. Parsing performance is an often criticised overhead of using an XML based approach. In several discussions over the design of the prototype, others have suggested that the use of a parsable meta-language (i.e. CCML) will impact too greatly on active application instantiation performance. However, no evidence has been presented as to the performance deficit of XML, nor has the overhead of the CCML meta-language been quantified. Therefore, to consider the performance of the adoption of an XML based meta-language to be parsed at service instantiation time, experiments were undertaken to ascertain CCML parsing performance.

The first stage of the experiment aimed to measure the performance of the initial implementation of the CCML parser based upon Sun Microsystems' Project-X SAX XML parser. Performance was measured according to two conditions; firstly the performance degradation from an increasing number of SELECT clauses, and secondly the overhead of the number of CHOICE clauses (per SELECT clause). The results obtained from the implementation of a CCML v0.1 parser utilising Sun Microsystems' SAX compliant XML parser are presented in Figure 7.4. Tests were conducted to measure the scalability of the parser firstly in respect to the number of sub-components declared (determined by the number of SELECT clauses). Secondly to determine the impact on performance of the number of component implementations (determined by the CHOICE clause).

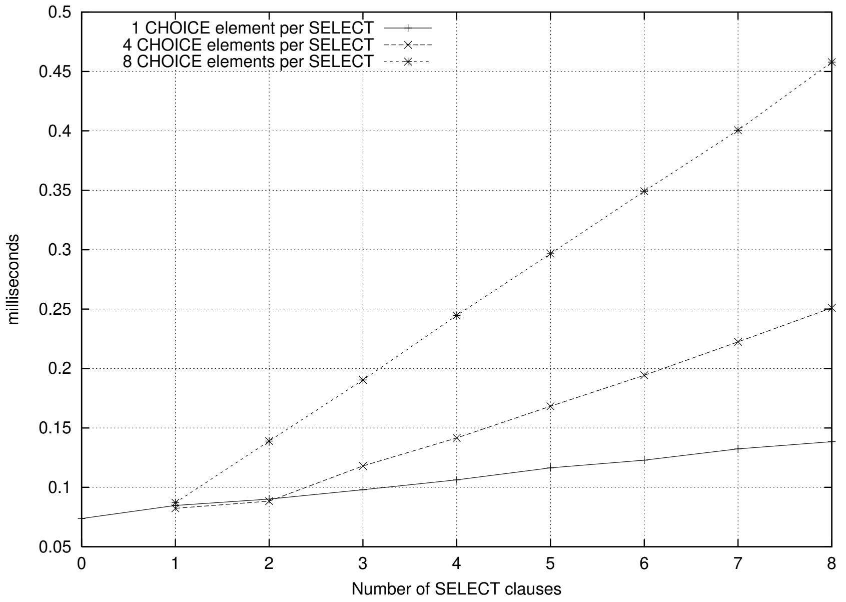


Figure 7.4: CCML parser performance (SAX parser)

The results clearly show that the parser scales very efficiently for resolving an increased number of sub-components. However, when the number of implementations (`CHOICE` clauses) to sub-component ratio increased, parsing performance degraded sharply. Whether this is an important flaw is dependent upon the number of component implementations. Indeed, if a conclusion can be drawn from current software engineering knowledge, this performance trade-off caused by coexisting sub-component implementations is desirable, rather than a performance degradation due to increased sub-components. Modern software developed using object-oriented design is composed of many sub-components, but it is indeed rare to find multiple implementations (other than perhaps a component port to two or three OS platforms). In any case the overall results for an estimated typical component (containing 8 sub-components) with a maximum of 8 implementations is still only 0.5 milliseconds, an extremely small portion of the total service instantiation time.

The second more significant stage at which latency is introduced is at the `UK.ac.lancs.proxylet.Proxylet.start()` method. Although the selection of component is made during the `load()` method, they are not instantiated for execution until invocation of the `start()` method. Since both the active applications (AA) and the ACP

(Active Control Plane) are coded in Java, the prototype adopts a URLClassLoader and Class.newInstance() mechanism to instantiate an executing instance of the active application. Although an AA's sub-components can be selected prior to instantiation, the constituent classes from which these are composed is not discovered and obtained until the Virtual Machine creates an instance of the AA.

This represents a potentially harsh overhead, meaning that extra class implementations may need to be downloaded at the moment the service is initiated. This latency has been measured and is presented in Figure 7.5. These results demonstrate the time to create an executing instance of an Active Application against the number of classes from which the active application is composed. There are two influencing factors, firstly the time to fetch the implementation of the class over HTTP. The second, the number of archives (JAR files) that must be searched to find the relevant implementation.

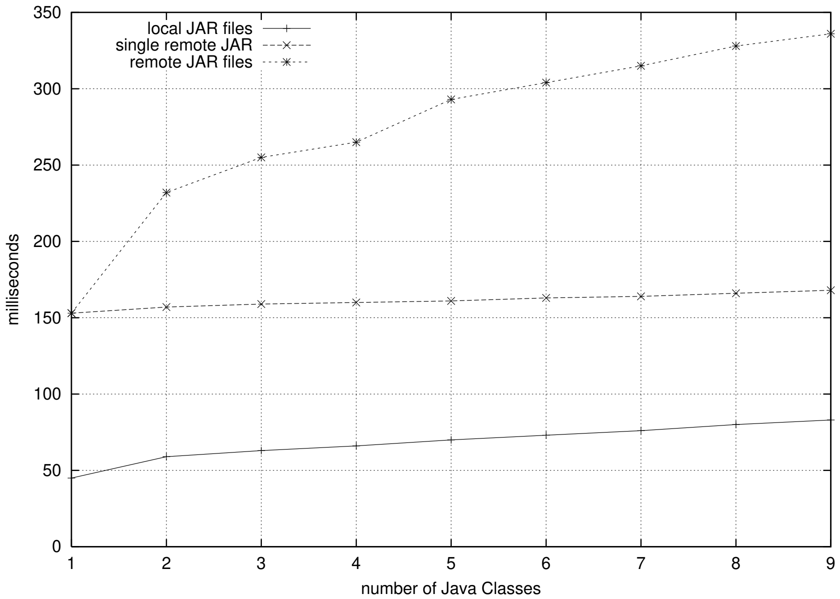


Figure 7.5: Active Application instantiation performance

The experiment comprised an AA composed of between one to nine classes. These classes were stored both in their own archive (JAR files) and in a single archive. These archives were stored both locally to the JVM, and remotely on an HTTP server at an HTTP request/response latency of 110 milliseconds.

The results indicate the ClassLoader mechanism to be very efficent, with an overhead of approximately 8 milliseconds per class. However, when the HTTP latency is factored into the results the remote class loading mechanism performs less satisfactorily. The results show a worse case scenario, where it is necessary to search through every archive to obtain the full set of required class implementations. The mechanism scales poorly, at the measured rate of incline a component consisting of 37 remotely archived classes would take over 1 second to fetch. However, given the efficiency of the ClassLoader mechanism itself, where these classes are archived together the latency is bound closely to the HTTP request/response latency.

The conclusion from these results is therefore that a single archive should be used for all of an AA's constituent classes if maximum service instantiation performance is a priority. Ideally a pre-fetch mechanism could be developed at service `load()` rather than `start()` time.

### 7.4.2 Signalling Performance

The importance of a middleware platform for composition of composite active services has already been established in this thesis. The efficient realisation of the active middleware platform is through the generation of a high-performance signalling system.

Signalling in the Alpine framework exists twofold; firstly the upload of new active code (proxylets) to the active router, and secondly the configuration of this active code through the 'L+' interface. These two signalling types can be considered as coarse and fine grain signalling respectively.

Although CORBA/IIOP has been adopted as the mechanism to provide the `UK.ac.lancs.proxylet.Sdyn` and `UK.ac.lancs.proxylet.Proxylet` interfaces used to support the Alpine lifecycle, the mechanism used to support the 'L+' interface is not stipulated. This is a matter for implementation of the active application, and should be tailored to interact with the management application residing at the Network Operations Services Platform (NOSP). The choice could range from *de-jure* standards including IIOP, RMI, DCOM, and SNMP, to proprietary protocols designed to inter-work with legacy systems.

As an example of a service deployed within the Alpine framework, a simple application has been developed to expose an 'L+' interface to enable influence of routing tables by algorithms and software situated off-router. An example of such a scenario is the establishment of Virtual Private Networks over which the customer has direct control. The 'L+' interface exposes a view of the router resources tailored to the permissions of the customer.

An excerpt of this interface (defined in CORBA IDL for simplicity) is provided (Figure 7.6) to depict a typical 'L+' interface. For comparison of 'L+' interface performance, active components were developed exposing this interface through the IIOP and SOAP

```
module routing {
  struct RouteId { string dest, mask; };
  struct RouteParams { string gateway; };
  struct Route { RouteId id; RouteParams params;};
  typedef sequence<Route> Routes;

  /* Accesses an L+ routing table. */
  interface RoutingTable {
    /* Get all routes. */
    Routes getRoutes();

    /* Get the parameters for one route. */
    RouteParams getParams(in RouteId id);

    /* Create a new route (or alter parameters of existing route). */
    void addRoute(in RouteId id, in RouteParams params);

    /* Delete a route.*/
    void delRoute(in RouteId id);
  };
};
```

Figure 7.6: 'L+' Route Control Interface

(Simple Object Access Protocol) (Box & et al, 2000) protocols. The IIOP instantiation was based on the Orbacus 3.2 (Java binding) ORB and the SOAP used the SOAP/Java (version 0.3) message parser from DevelopMentor [3]. This required the use of a SAX XML parser, for these tests the Xerces/Java v1.0.0 parser from Apache [4].

To gauge 'L+' interface performance and aid selection of interface instantiation mechanisms, measurements were made of SOAP and IIOP message sizes and round trip time for request/response. The "`getParams()`", "`addRoute()`" and "`delRoute()`" methods detailed in the IDL in figure 7.6 were taken as example 'L+' interface calls. It was necessary to factor out the network traversal time, and any work the interface implementation may make on receipt of the call, in order to achieve a true representation of the call processing time.

Table 7.3 shows the round trip request/response time. These figures show SOAP to be considerably slower and require more processing than equivalent IIOP invocations. The "on the wire" size of the invocation and response messages for SOAP was also found to be approximately eight times larger.

[3]SOAP/Java v0.3 parser available from DevelopMentor Inc. at `http://www.develop.com/soap/`

[4]Xerces XML parser available from Apache at `http://xml.apache.org/xerces-j/index.html`

| Method | IIOP | SOAP |
|---|---|---|
| `getParams()` | 1.95ms | 41.27ms |
| `addRoute()` | 2.03ms | 40.50ms |
| `delRoute()` | 1.72ms | 38.16ms |

Table 7.3: Round trip request/response times for IIOP and SOAP interface implementations

| RPC Protocol | IIOP | SOAP |
|---|---|---|
| Request | 100 bytes | 609 bytes |
| Response | 39 bytes | 520 bytes |
| Total | 139 bytes | 1129 bytes |

Table 7.4: SOAP and IIOP message sizes for `getParams()` method

Table 7.4 provides a comparison of the relative size of IIOP and SOAP request and response messages. Since SOAP is still relatively new and unknown, an example HTTP/SOAP message is illustrated in Fig 7.7. As can be observed, the SOAP message contains a standard HTTP header with the body in written in XML. Data to be transferred in the SOAP request is simply marshalled into textual representation, despite the inefficiency of this approach.

Clearly the figures obtained show that SOAP performs poorly when compared to the IIOP protocol, this despite SOAP lacking the rich set of RPC features found in IIOP, and without the overhead of the CORBA object model (which could indeed be built over SOAP) in the Orbacus experiments. In the most part the poor performance of the SOAP parser was due to the overhead of the XML. The Apache Xerces parser used in the experiments is a generic SAX parser, and should these become available, be replaced with a parser optimised for SOAP. The XML parser was developed using Java, and could obviously be re-engineered in C or C++ to improve performance, but to enable fair comparison the IIOP implementation was similarly Java based.

Nevertheless, within our chosen usage domain these issues are not the highest priority. The Java bytecode for these interface implementations are considerably larger than the SOAP/IIOP messages. Given the dynamic rapid interface instantiation feature that requires the upload of bytecode onto the Active Router, it is important that the implementation bytecode be as compact as possible.

A comparison of the total bytecode size of the interface implementations is made in

```
POST /L+interface/routing HTTP/1.0 Content-Type:
text/xml SOAPMethodName:  urn:schemas-lancs.ac.uk:caspian#delRoute
User-Agent:  Java1.2.2
Host:  wand:8888 Accept:  text/html, image/gif, image/jpeg, *;
q=.2, */*; q=.2
Content-length:  362
<?xml version='1.0' ?>
<s:Envelope xmlns:ns2='urn:schemas-lancs.ac.uk:caspian'
xmlns:xsi='http://www.w3.org/1999/XMLSchema/instance'
xmlns:s='urn:schemas-xmlsoap-org:soap.v1'>
       <s:Body>
                <ns2:delRoute>
                     <id s:href='#sid1' />
                </ns2:delRoute>
                <ns2:RouteId s:id='sid1'>
                     <dest>148.88.153.50</dest>
                     <mask>255.255.0.0</mask>
                </ns2:RouteId>
       </s:Body>
</s:Envelope>
```

Figure 7.7: Sample HTTP/SOAP message to invoke `delRoute()` method

Table 7.5. Class files that account for these totals have been selected based on their need to be present in code that exposes the interface of an object implementing `RoutingTable`. The ORB and the HTTP server have been excluded from these, since they would likely be provided locally on the system.

For both CORBA/IIOP and SOAP, this includes similarly sized parts such as classes representing types defined in the IDL, a skeletal interface implementation, and a main program to instantiate the implementation, and bind it to the relevant communication services.

In the IIOP version, a selection of other IDL-generated classes (stubs, helpers and holders) constitutes the bulk of the extra weight, since little of their function is needed for SOAP. What is needed includes a method dispatcher, SOAP body classes representing method requests and responses, and the registration of Java types to SOAP types, all of which have been hand-written.

The code-size advantage of SOAP has come from eliminating unnecessary code for simple distributed services. This shows that although SOAP has a considerably higher overhead in terms of message size and required processing, the actual implementation is approximately only 43% of the size of the equivalent IIOP implementation. Given the

| RPC protocol in implementation | Size of implementation |
|---|---|
| IIOP | 26129 bytes |
| SOAP | 11241 bytes |

Table 7.5: SOAP and IIOP implementation

different protocol features, it is obviously not possible to declare either protocol entirely suitable for all inter-component signalling.

Therefore, the choice as to which protocol to use to provide 'L+' interfaces is not specified within the Alpine framework; the most appropriate protocol should be chosen given the usage requirements posed. This should include consideration of interface lifetime, likely interface usage, type of clients, and security restrictions.

## 7.5 Summary

This chapter has presented the evaluation of the Alpine framework and associated component the prototype Active Router. An argument for the approach taken was described in section 7.1. This resulted in the presentation of the Alpine framework as a three stage process, first considering architectural requirements, secondly a qualitative evaluation of the framework, and thirdly a quantitative analysis of performance aspects of the Active Router prototype.

In section 7.2, the functional requirements that have been introduced throughout this thesis were collected. These requirements were arranged into three groups:

- Group A: requirements necessary to construct services within the Alpine Framework
- Group B: requirements relating to consistency with current standardisation
- Group C: design requirements imposed upon the Active Router

Each of these requirements was presented, along with a discussion as to whether the requirement has been met by the framework and prototype node described in the thesis. Only requirement B.2 was not met, and a justification was given as to why this requirement is inconsistent with the other requirements imposed upon the Alpine framework.

Section 7.3 presented the qualitative evaluation of the framework. This was achieved through the presentation of a usage case scenario, and a discussion on the merits of the composite active services model inherently supported by Alpine over the conventional model. This was complemented with a comparison of active router architecture against the latest generation Passport/Accelar family of routers from Nortel Networks.

Finally, the evaluation of the framework (section 7.4) was supplemented with some results on the performance of the prototype. This quantitative evaluation has tried to consider the factors that influence node performance that are unrelated to the non-production nature of the implementation code. This has concentrated on the two most important factors affecting the node, latency of service instantiation and 'L' interface performance. The results clearly show that the overhead of dynamic code instantiation is marginal, considering the *discrete* as opposed to the *integrated* approach to active networking followed.

The choice of distributed middleware mechanism for inter-domain communication (e.g. the 'L') is not mandated in the framework, but the selection inevitably impacts upon performance. Two techniques were considered, the established IIOP and the state-of-the-art SOAP protocols. The results clearly show that there is a trade-off between the interface performance, and size of active code implementation. It was impossible to draw a conclusion as to which technology is more appropriate; the selection is unique to each particular usage domain.

# Chapter 8

# Conclusion

This final chapter summarises the contributions and results presented in this thesis. The approach taken shall be to provide an overview of the structure of the thesis in section 8.1, followed by a synopsis in section 8.2 of major contributions. Section 8.3 discusses areas not considered within the remit of the thesis, and suggests further work that may be undertaken. Finally, in section 8.4 the chapter and thesis are drawn to a conclusion with some closing remarks.

## 8.1 Overview of thesis

Chapter 1 set the scene by introducing the notion of value-added services constructed over the underlying heterogeneous communication technologies. Once defined, the appropriate location of the service intelligence was discussed, both in the end terminal and within the network itself. The conclusion drawn, given the trend towards heterogeneous mobile terminals with low computational power and memory, was that the end-terminal is an inappropriate place to host extensible value-added communication services. The provision of these services as revenue generating network-hosted intelligence justifies the Network Operator/Internet Service Provider emphasis of the thesis. The seven main elements of the thesis were then introduced: customer oriented management, third party element programmability, fusion of programmable and active technologies, use of programmable interfaces, compatibility with trends and standardisation, utilisation of existing middleware technologies, and contribution of full service provisioning context to active network research.

Chapter 2 provided the background context from which the subject of this thesis, the Alpine framework, has emerged. This included an account of three factors, deregulation, broadband transport, and distributed object technology that have greatly influenced the developments within the telecommunications and information technology industries over the past decade. Three technologies have been of particular significance to the Alpine framework, the Intelligent Network (Berman & Brewster, 1992; Brendan *et al.*, 2000), Broadband-ISDN (ITU-T, 1991a; ITU-T, 1991b), and the Telecommunications Information Networking Architecture (TINA) (section 2.3) (Inuoe *et al.*, 1999). The chapter then explained why TINA and B-ISDN have not had the expected impact and introduces the emergence of the Internet and the challenges still faced.

Chapter 3 considered the state-of-the-art technologies in the field of distributed programmable mechanisms, that have provided the motivation for the Alpine framework. These stem from two categories of technologies, open signalling and active networking. The first school of thought examined emerged from the Open Signalling research community and is typified by the work done at Columbia University on the Extended Reference Model (XRM) (Lazar *et al.*, 1994; Lazar *et al.*, 1995), and Xbind (Chan *et al.*, 1996). Other fora that have adopted the technique are described, including the TINA NRA (and extensions in the ReTINA project, section 3.2), and the IEEE standardisation project P.1520 (section 3.4).

In the second category of distributed programmable mechanisms described by the chapter, that of the active network, code to control the network element is located directly on the controlled device rather than at the OSS (Operations Support System) as typical in open signalling solutions. Active networking research covers a broad spectrum of solutions and has been plagued with many common misunderstandings. To clarify the type of active network technology applicable to this thesis, two categories were defined thus: *integrated* and *discrete*. The most commonly (mis)understood *integrated* approach follows the in-band capsule mechanism, whereby every packet may contain both data and active code. Programmability is projected towards end-user service creation but there are still many issues relating to the practicality of this scheme including latency of packet authentication, charging, and efficient code execution.

Recent active network research (Marshall *et al.*, 1999; Juhola *et al.*, 1999; Fry & Ghosh, 1999; Banfield, 2000; Marshall & Banfield, 2000), including the Alpine framework discussed in this thesis has followed the *discrete* approach. Several schemes that have influenced the design of the Alpine framework are discussed.

Chapters 4 and 5, presented the bulk of the contributions made in the thesis. This was given in two stages, first chapter 4 provides the requirements and motivations behind the framework. To aid presentation of the framework, a number of established technologies and toolsets were adopted, including RM-ODP, OMG IDL, UML, and the

IEEE P.1520 Reference Model. The importance of the business model (RM-ODP enterprise viewpoint) was acknowledged, and two current examples, the OMG's "Open Service Marketplace" (Schenk *et al.*, 1999) and the EURESCOM P926 "My Own Network" Model (van Tilborg & Schuring, 2000) were given. The Alpine business model, a fusion of the Internet marketplace and the IEEE P.1520 Reference Model, was defined to aid discussion as to which stakeholders the service provisioning through active network technology is applicable. Consideration was then given to the three locations where active programmability may be introduced into the Internet, on the access network, at the Autonomous System border nodes, or most controversially within the core network.

The importance of active network technology within a complete distributed process environment was described in Chapter 5. Important issues discussed include the application of a full active service lifecycle, and the role of signalling for the provision of active services. The chapter then described the Alpine Middleware Inter-networking Environment (AMIE), a complete architecture for realisation of the Alpine framework within the spirit of IEEE P.1520. Finally, a novel extension of the P.1520 reference model was suggested, incorporating a new vertical-plane, namely the Signalling Transport Service Provider (STSP) and corresponding 'S' interface reference point.

Chapter 6 described several of the components of the Alpine framework in more detail, focusing on the implementation issues. Since the Alpine framework covers a broad spectrum, from element to service layer, it was obviously infeasible to provide a complete implementation of all components specified in AMIE. Instead, this chapter described the implementation of the Active Router, the fundamental base component. The motivation behind the implementation was as an experimental platform, and the chapter described two current programmable node designs, ALAN and LARA, which have served to contribute to the realisation of the programmable Active Router (AR). Following this, the overall architecture of the node was described, dividing programmability between the Active Control Plane (ACP) and Active Data Plane (ADP). Secondly, the mechanism for realisation of the middleware signalling infrastructure was defined, adopting a CORBA implementation to realise the access and location transparency properties defined within AMIE.

An essential facet of the Alpine framework is that there can be no single extensible programmable abstraction, with heterogeneous nodes from multiple vendors, located throughout the network in different operator domains.

Heterogeneity of programmable abstraction amongst nodes complicates the task of installing the correct active service implementation in the correct locations. A novel agent-driven content negotiation mechanism was developed and adopted, the Component Compatibility Mark-up Language (CCML), an XML-based language enabling the node, based on local knowledge to select the appropriate service implementation.

Chapter 7, the penultimate chapter, presented an evaluation of the proposed architecture. First, the evaluation took the form of a simple statement of the fundamental requirements posed by the Alpine framework, followed by an explanation of success or failure to comply in the resulting implementation. Following this, a more detailed evaluation was made both of the framework and the Active Router implementation. Since it was infeasible to realise a complete implementation of the Alpine framework, qualitative techniques were adopted to evaluate the appropriateness of the framework. This was partially based on two case studies; a usage evaluation scenario, and a comparison of the prototype Active Router design against a state-of-the-art Gigabit router.

A quantitative approach was followed for the evaluation of the Active Router implementation. In particular two factors were considered, service instantiation latency, and interface (signalling) performance. Service instantiation performance was considered according to two factors, overhead of Component Compatibility Mark-up Language (CCML), and latency of the active component fetch inherent in *discrete* active network approach. The results showed that although there were some minor scaling issues with regard to CCML when increased choice amongst component implementations was introduced, these were insignificant compared to total service instantiation time. Active component fetch latency amounted to a greater proportion of instantiation time, the *discrete* approach requiring out-of-band remote retrieval of active component implementations. Given the measurements obtained, it was calculated that it would take over one second to fetch 37 remotely archived component implementations. Further study showed this latency to be introduced not by the ClassLoader mechanism, but by the HTTP request/response latency. Performance could be improved by the same caching mechanisms used for Web (HTML) traffic.

## 8.2 Thesis Contributions

The following sections consider the main contributions made and successful achievement of the aims of this thesis.

### 8.2.1 Customer-oriented control and management

An important aim of active network solutions is to be able to tailor services to meet customer demands. The Alpine framework provides this by enabling code to be developed in general purpose programming languages (e.g. in the prototype node, code is written using the Java language), and dynamically installed on network elements. Unlike traditional service provision models there is no divide between control and management functionality, and applications can easily be constructed for services spanning both sets of functions.

### 8.2.2 Third-party element programmability

The Active Router component of the Alpine framework strongly supports the opening of router architecture to allow the creation of services by third parties. However, unlike other solutions which try to define a homogeneous "catch-all" API, instead, this thesis accepts the inevitable heterogeneity amongst network elements and the naturally resulting heterogeneous programmable abstractions. The Alpine framework defines a component model in which multiple abstractions can be composed and dynamically introduced. The Component Compatibility Mark-up Language (CCML) is provided as a mechanism for resolving appropriate service implementations against programmable abstractions.

### 8.2.3 Fusion of programmable and active network technologies

A significant contribution made by this thesis is to recognise the mutual goals shared between the programmable (open signalling) and active network research communities. Both share a principal objective to move beyond the current closed proprietary programmed network elements to enable open third party element programmability. Surprisingly there has been little awareness and fusion of the open signalling broadband kernel and BIB (Binding Interface Base) approach typified by Xbind (see section 3.4.3), and the active node technology epitomised by ANTS (Wetherall *et al.*, 1998) and Switchware (Gunter *et al.*, 1997).

The Alpine framework described within draws upon the strength of both approaches, incorporating the potential to reprogram node behaviour locally at the network element (e.g. redefine a scheduling algorithm), with a finer-grained configuration from the network-wide perspective at the OSS (Operations Support System). Indeed this is a logical division of programmability, allowing generic active components to be engineered that can be tailored to specific run-time needs. A parallel can be drawn with service components at the terminal device, where conventionally we use a generic TCP/IP stack and configure its IP address at boot-time through static allocation or DHCP (Dynamic Host Configuration Protocol).

### 8.2.4 Utilisation of programmable interfaces

The Alpine framework offers a pragmatic approach to network programmability using heterogeneous programmable abstractions. Many approaches to active networking are unsuitable because they fail to recognise the differing programmability requirements posed by different domains. Rather than specify a fully featured node OS, and constrain this programmability through the Execution Environment (EE) abstraction, Alpine recognises there cannot be a single full featured Node OS specification. Instead, active nodes may be highly specialised devices, some containing large amounts of physical storage, others high-speed computation.

Different programmable abstractions can be constructed to provide a view of programmability appropriate to the user's viewpoint. The range of viewpoints and underlying resources makes it difficult to provide a range of suitable Execution Environment generalisations. Instead, the programming language, user viewpoint, and abstraction of underlying resources are de-coupled so that each can be configured independently. Thereby enabling programmability to be tailored to a finer degree than possible using the common EE notion.

### 8.2.5 Compatibility with emerging IEEE P.1520 standardisation

A potentially important contribution has been through participation in the IEEE P.1520 IP subworking-group, P.1520.3. The Active Router component of the Alpine framework and working-groups effort on 'L' interface standardisation have independently developed to a similar conclusion.

Although IEEE P.1520 initially favoured a static open-signalling inspired approach to element programmability, no approach is mandated. The Alpine framework should be considered as an interpretation of IEEE P.1520 in an active networking context. Whilst originally controversial, an active approach is now an accepted possible interpretation, and two such solutions, the Alpine framework, and the EC ISTS funded FAIN project (Galis *et al.*, 2000) were presented in July 2000 at the IEEE P.1520 meeting in Santa Clara, USA (Banfield *et al.*, 2000a; IEEE, 2000).

The Alpine prototype Active Router remains one of the first routers inspired by the spirit of the P.1520 Reference Model. Although standardisation is far from complete, the element programmable abstraction (Service Specific Building Block, Resource Building Block, and Base Building Block component abstractions) has been accepted. These components have direct interpretation within the Active Router design, the Service Specific Building Block represented by the Active Application (AA), and the Resource Building Block (RBB) interpreted as the veneer component stub.

### 8.2.6 Middleware service support for active networks

A unique feature to Alpine as an active platform is the adoption of an underlying middleware platform based on distributed remote procedure calls (RPC). Following the *integrated* active networking approach, an active application (AA) is normally self contained, passing between active nodes with little need to inter-operate with other active services. However, within the Alpine framework more advanced composite services can be constructed, composed of multiple active applications distributed throughout the network.

Most active network approaches avoid heterogeneity issues (platform, operating system, language) by adopting an Execution Environment (EE) abstraction. The Alpine framework instead embraces heterogeneity with an extensible abstraction at the Active Router and Active Intelligent Peripheral.

Therefore, given the distributed and heterogeneous nature of the active platform, a middleware platform is required to provide access and location transparency. The CORBA v2 middleware architecture and OOC's Orbacus implementation used in the prototype are sufficient to meet these requirements.

### 8.2.7 Full service provisioning context for active network technology

Other distributed system technologies (e.g. OMG CORBA, TINA, agent technologies) have defined a full model for service provisioning but little support has been provided for the lifecycle of an active service. The Alpine framework recognises the importance of the definition of service lifecycle, and has defined a simple seven stage model (see figure 5.4). This model is integral to the platform architecture, the Alpine Middleware Internetworking Environment (AMIE), defining the mechanism by which code is downloaded to the Active Router (via HTTP), and the process by which active services may be "subcontracted" to application layer active components installed at the Active Intelligent Peripheral (AIP).

The framework is based upon the *discrete* approach to active code deployment, and defines a series of interfaces (figure 6.6 & 6.7) describing four basic control primitives (*load*, *start*, *modify*(in the interface defined as `control()`), and *stop*) that must be offered by all active nodes. The implementation of the framework uses an out-of-band mechanism exposing CORBA interfaces and IIOP encoded RPC to envoke operations to shift between states.

### 8.2.8 Isolate role of signalling in service provision

Fundamental to the provision of network services is a mechanism to provide inter-component communication (signalling). However, consideration of the role of signalling and allocation of adequate resources is often overlooked. An important contribution made in this thesis is the addition of a new business entity, the Signalling Transport Service Provider (STSP), whose function is to provide the transport of signalling messages.

The isolation of this role is a natural extension of the open market principles embraced by the Open Signalling Community. The role of signalling provision applies equally to IEEE P.1520, IN, Parlay, or the Internet, but in this thesis the P.1520 reference model was used for further exploration.

In addition to the introduction of the new STSP role and associated vertical plane in the P.1520 reference model, a new corresponding vertical interface (the 'S' interface) was introduced. The 'S' reference point serves as a control and management interface between the VASL, NGSL, and VNDL horizontal planes and the signalling provision (STSP). Two or more functions wishing to communicate must both form a binding with an STSP to enable them to do so. The resulting contract between the signalled entities and the STSP is similar to the notion of a Service Level Agreement (SLA) between end customer and network operator over the UNI (User-Network Interface).

At present, this additional reference point has not been adopted in the IEEE P.1520 reference model. Definition of this interface is still at an early stage and more work is required before it can be put before the standardisation forum.

### 8.2.9 Pragmatic approach to network element programmability

The Alpine framework divides programmable elements into two categories of device, namely Active Router and Active Intelligent Peripheral. The Active Router extends the functionality of a traditional IP router by allowing the upload of programs to influence packet forwarding. These devices have limited computational power, and are concerned solely with OSI layer-3 programmability.

A greater range of programmability is available at the Active Intelligent Peripheral

(AIP); these are clusters of computation in the network, possibly optimised to specific tasks. Although ideally located in the vicinity of an Active Router, the AIP is not required to function at full line-rates.

The design of the prototype Active Router has been inspired by the architecture of the current generation of non-programmable IP routers. These achieve scalability to a large number of ports by deploying a distributed data-plane, usually based on dedicated per port hardware interconnected by a high-speed backplane. Control-plane functions (routing table calculation, SNMP daemon etc.) usually reside on a separate general-purpose control processor. A similar division is made in the prototype Active Router with the notion of the Active Control Plane (ACP) and Active Data Plane (ADP).

Locating these active nodes within the IP network introduces additional requirements upon their design. The thesis identified four locations where service programmability may be introduced into the Internet: the end terminal, the access network, the Autonomous System border routers, and the core network devices. These in turn are required to operate at higher line rates, with greater security and resilience. The conclusion drawn was that given the trend towards terabit line speeds in the core network, combined with the future use of cut through layer-2 routing and optical switching, programmability in the core network is unlikely. However, in the short-term programmable nodes could be introduced into the access network since it is here that many of the advanced network services (QoS and caching) are required. Given the globalisation of telecommunication services, over the medium term it is natural that programmability should be extended to the border routers so as to provide inter-network end-to-end programmable services.

### 8.2.10 Other Contributions

The Alpine framework has not been developed in isolation from other research activity in the field of active and programmable network technology. The contributions made in this thesis have been presented, discussed and in some cases incorporated in several other projects.

The first and most important of these has been EURESCOM P926, a project considering the construction of a dynamic service environment using policies integrated with active networks. Although the architecture differs from the Alpine framework, adopting a more user-oriented approach to active service provision, a similar division is made between programmable nodes, in P926 terminology the active router and active server. The P926 active router is based on an abstraction model similar to that defined within this thesis.

With funding from the British Telecom University Research Initiative, a collection of Universities (University College London, Imperial, Sussex, University of Technology Sydney, and Lancaster University) are co-operating on active networking research activity. Many of the ideas discussed within this thesis have been presented to the group, culminating with a joint presentation (Marshall & Banfield, 2000)(Simpson *et al.*, 2000) with BT at the IEE Application Layer Active Networking workshop in London, November 2000.

Finally, the implementation of the prototype Active Router, has taken inspiration from the FunnelWeb active node platform developed at the University of Technology Sydney (UTS). The authors of this platform have visited the laboratory at Lancaster University, and viewed/discussed the prototype Active Router implementation. It is expected that UTS will incorporate a content negotiation mechanism similar to CCML into future versions of their FunnelWeb platform.

## 8.3 Further Work

### 8.3.1 Complete specification of Building Block abstractions

Clearly the IEEE P.1520 'L+' and 'L-' interface reference points and the emerging three tier building block abstraction have a major impact on the design of the prototype Active Router. Although the prototype was designed with regard to the preceding two-tier 'L' interfaces, the prototype and the P.1520.3 subworking group models have been developed towards a common conclusion.

While the Active Router can be claimed to be one of the first IP router implementations in the spirit of the P.1520 reference model, there are at present within the IP subworking group no agreed interface definitions to standardise upon. An initial set of Base Building Block (BBB) abstractions has been suggested in the three-tier architecture definition document (Vicente & et al, 2000). However these five abstractions (action, component, condition, target, and processing unit) are serving as an initial discussion point, and should not be taken as an agreed definition. The first priority is to standardise the Resource Building Block (RBB) abstraction, and from these, common features can be extracted to form the BBB abstraction.

An initial set of RBB abstractions to act as a discussion point has been presented by KRDL and Intel to the P.1520.3 subworking group (Raguparan & Vicente, 2000). However, these have been rejected as not capturing a generic router abstraction. Indeed this rejection as "being just another set of proprietary interfaces" influenced the decision not to attempt to define a complete set of abstractions for the prototype Active Router.

The P.1520.3 IP subworking group is currently charged with standardisation of a complete set of RBB abstractions. Originally, it was intended that a draft set should be available during October 2000, although this deadline has not been met. Once a draft RBB specification is published, the Active Router can be used as a platform to experiment and provide feedback on the proposed abstractions.

### 8.3.2 Active feature interaction

An important contribution in this thesis is the assertion that active services are not stand-alone components but composed of multiple distributed Active Applications. A useful service is composed through the interaction and co-ordination of the services provided by these Active Applications. However, this thesis has not considered the effect of the unforeseen interactions between heterogeneous Active Applications.

This problem of so called "feature interaction" is well known to the IN (Intelligent Network) community. A simple example is a user who has both a GSM and PSTN telephone and uses the call forwarding service. The user starts by deciding to set the PSTN to forward all calls to his GSM handset. However, later he decides not to receive calls on his mobile, and redirects this to the PSTN telephone. The result is a continuous call forward loop because the two network services failed to interact intelligently.

Naturally, the potential for unfavourable feature interaction between active services is greater than for IN because of the greater range of potential services. No solution is offered in the Alpine framework, this being a matter for further work.

A possible solution is the use of composition filters (Bergmans & Aksit, 2000), additional active components that act as the glue between the Active Applications that form composite active services. Such as solution is being developed by KPN Research as part of EURESCOM P926, and it is intended that these results will be combined with the active node architecture (of which the Active Router is a component) in a later stage of the project.

### 8.3.3 Extend prototype to next-generation programmable platforms

An important facet of the design of the Active Router is the recognition of heterogeneity amongst router platforms. The prototype has been built using a Linux (LARA) kernel as an example Active Data Plane (ADP), but the design is claimed to be applicable to other programmable environments.

Several interesting programmable platforms are under-development that could serve as alternative ADP to prove the flexibility of the platform abstraction. First is the LARA++ active environment (Schmid *et al.*, 2000a)(Schmid *et al.*, 2000b), the successor to the LARA programmable kernel. LARA++ is a completely new programmable environment based on composing active services through constructing a chain of programmable components. Future work could include investigating the relationship between the component chain and the Resource Building Block (RBB) abstraction.

A second interesting candidate platform is the Oplet Runtime Environment (ORE) from Nortel Networks Openet Technology Laboratory in Santa Clara, USA. This platform is very significant to the programmable network community, since this is the first occasion that a major vendor has opened a commercial router to third party programmability. Based on the new Passport/Accelar series of gigabit routers, the ORE provides a Java VM abstraction to directly control the underlying packet forwarding hardware. The platform supports secure downloading, installation, and execution of active code written in Java and wrapped as an "oplet" using the ORE interfaces. The ORE supports a number of APIs including JMIB (for Management Information Base access), JCapture (for packet capture), and JFWD (to control packet scheduling and forwarding). The ANTS (Active Node Transfer System) from MIT has been ported to the ORE platform, and it would be interesting to explore the possibility to port the prototype Active Control Plane (ACP) to this commercial grade router.

### 8.3.4 Interpretation for other programmable architectures

The Alpine framework is an interpretation of the application of active network technology in the realisation of the IEEE P.1520 reference model. Its applicability can be estimated by the fact that an active interpretation is increasingly accepted within the P.1520 and Open Signalling communities.

Nevertheless, the relevance of IEEE P.1520 is questionable. Although formed in 1997, its efforts towards standardisation have made slow progress, especially within the IP subworking group. However, more positively, both Cisco and Nortel Networks, the largest IP router vendors, have joined and are actively participating in the standardisation effort. The inclusion of Intel, with their new IXP "network processor" adds an interesting new dimension to the group.

Other similar consortia considering programmable architecture could be realised through the application of active networking technology. Of particular worth is the Parlay Consortium (Parlay Group, 1998) (section 3.5), an eleven company consortium to design and promote an Application Programming Interface (API), providing safe access to network capabilities that enable the creation of novel services. Importantly, the Parlay Group is not intending to define how these features provided through their API are mapped onto the underlying infrastructure. Solutions to map the Parlay API onto the physical resources are made on a per network basis, and it is obvious that active networking is a good candidate for control and management of resources within an IP network.

## 8.4 Concluding Remarks

The convergence of the Telecommunications, Information Technology, and Broadcasting industries is resulting in many new demands on communications services. Clearly the current Internet, although hailed as the future superhighway, cannot yet meet these new demands. There are many working groups within the Internet Engineering Task Force (IETF) developing ad-hoc solutions to Quality of Service, multicast, mobility, and security.

Network equipment items (routers, switches, firewalls, etc.) are closed devices, and the provision of a new service must be delayed until it is expedient for the equipment vendor to make available updated software releases. This is in stark contrast to the Personal Computer industry, where third party access to open APIs (Application Programming Interfaces) has created a vibrant market for specialised software.

The Internet enjoys a similar open environment, with a host of client-side Java applets, Javascript and Active-X controls combined with server-side servlets and CGI (Common Gateway Interface) scripts providing novel Web-based services. The use of active network technology aims to open the network element to provide similar programmability within the network.

This thesis provides a basis for the opening of network elements through active technology to provide these emerging network services (QoS, multicast, mobility, and security). Early active network research concentrated upon the building of active nodes to prove the concept of the dynamically reconfigurable network element. The Alpine framework described within this thesis built upon these node technologies (in particular LARA and FunnelWeb), concentrating instead on providing a complete framework resolving the heterogeneity and distribution issues faced in the creation of composite active services.

A crucial outcome of this work is the co-operation with international industrial projects and standardisation groups. Of particular relevance are the EURESCOM P926 and IEEE P.1520 projects. Regular feedback from industrial colleagues working on the P926 project (Caspian), has provided invaluable requirements that have helped shape the thesis, and in return fed results back to the project. Participation in several IEEE P.1520 standardisation meetings has provided an in-depth understanding of the emerging architecture. In turn, the resulting Alpine framework provides an early interpretation of the architecture and the prototype node has served as an appropriate test-bed.

# References


Angin, O., Campbell, A., Kounavis, M., & Liao, R. 1998. The Mobiware Toolkit: Programmable Support for Adaptive Mobile Networking. *IEEE Personal Communications, Special Issue on Adapting to Network and Client Variability*, **5**(4), Pages 32–44.

Aranda Gutiérrez, P., Salgado, J., & Calvo, T. 2000. Active e-mail Infrastructure. *In: Telecom I+D'2000*. Madrid and Barcelona: Spain.

ATM Forum. 1995 (December). *B-ISDN Inter Carrier Interface (B-ICI) Specification, Version 2.0 (Integrated)*. af-bici-0013.003 edn. ATM Forum Technical Committee.

ATM Forum. 1996a (July). *ATM User-Network Interface (UNI) Signalling Specification Version 4.0*. af-sig-0061.000 edn. ATM Forum Signalling Working Group.

ATM Forum. 1996b (April). *Traffic Management Specification Version 4.0*. af-tm-0056.000 edn. ATM Forum Traffic Management Working Group.

Baker, F., Braden, B., Bradner, S., O'Dell, M., Romanow, A., Weinrib, A., & Zhang, L. 1997 (September). *Resource ReSerVation Protocol (RSVP) Version 1 Applicability Statement*. Request for Comment 2208. IETF.

Banfield, M. 2000 (October). CASPIAN: Constructing A dynamic Service environment using Policies Integrated with Active Networtks. *In: Applied IP and Multimedia Services Workshop*. EURESCOM, Heidelberg, Germany.

Banfield, M., & Hutchison, D. 1997 (July). Provision of QoS-based routing under TINA. *In: 4th Communication Networks Symposium*. Manchester, UK.

Banfield, M., & Oliver, H. 1997 (March). *A Generic Quality of Service Specification*. Design Specification RT-TR-LAN-2-3/97. ACTS Project ReTINA.

BANFIELD, M., EDWARDS, C., CHARTON, N., & HUTCHISON, D. 1999. Providing Scaleable QoS-based Connectivity Services. *Pages 216–225 of: TINA'99*. Oahu, Hawaii: USA.

BANFIELD, M., BRAUN, T., & PRIGENT, E. 2000a. Caspian: An Integration of P.1520 and active network technology. *In: Presented at IEEE P.1520.3 Sub-working group meeting*. Santa Clara, California: USA.

BANFIELD, M., SIMPSON, S., & HUTCHISON, D. 2000b. Provision of Signalling for Programmable Services. *Pages 307–330 of: Telecommunication Network Intelligence*. Sixth International Conference on Intelligence in Networks (SmartNet 2000). Vienna: Austria.

BERGMANS, L., & AKSIT, M. 2000. Constructing Reusable Components with Multiple Concerns Using Composition Filters. *Software Architectures and Component Technology*.

BERMAN, R., & BREWSTER, J. 1992. Perspective on the AIN Architecture. *IEEE Communications*, **Vol 1**(Issue 2), Pages 27–32.

BERSON, S. 2000 (October). A Gentle Introduction to the ABone. *In: OPENSIG 2000*. OPENSIG, Napa Valley, California, USA.

BISWAS, J., & ET AL. 1998. The IEEE P.1520 Standards Initiative for Programmable Network Interfaces. *IEEE Communications Magazine*, **Vol 36**(Issue 10), Pages 64–70.

BISWAS, J., & ET AL. 1999. *Application Programming Interfaces for Networks*. Whitepaper. IEEE P.1520.

BISWAS, J., WANG, W., & ET AL. 1998 (November). *Programming Interfaces for IP Networks*. Whitepaper Version 4. IEEE P.1520. P1520/TS/IP001.

BLAIR, G., & STEFANI, J-B. 1998. *Open Distributed Processing and Multimedia*. Addison-Wesley. ISBN 0-201-17794-3.

BLAKE, S., BLACK, D., CARLSON, M., DAVIES, E., WANG, Z., & WEISS, W. 1998 (December). *An Architecture for Differentiated Services*. Request for Comment 2475. Differentiated Services Working Group, IETF.

BOX, D., & ET AL. 2000 (May). *Simple Object Access Protocol (SOAP) 1.1*. Tech. rept. World Wide Web Consortium.

Braden, B., Gunter, C., Jackson, A., Keromytis, A., Minden, G., & Wetherall, D. 1997a (July). *Active Network Encapsulation Protocol (ANEP)*. RFC Draft edn. `http://www.cis.upenn.edu/ switchware/ANEP/docs/ANEP.txt`.

Braden, R., Zhang, L., Berson, S., Herzog, S., & Jamin, S. 1997b (September). *Resource ReSerVation Protocol (RSVP) -Version 1 Functional Specification*. Request for Comment 2205. RSVP Working Group IETF.

Brendan, R., Jennings, B., McArdle, C., & Curran, T. 2000. Evolutionary Trends in Intelligent Networks. *IEEE Communications*, **Vol 38**(Issue 6), Pages 86–93.

Breugst, M., Busse, I., Covaci, S., & Magedanz, T. 1998. Grasshopper - A Mobile Agent Platform for IN Based Service Environments. *Pages 279–290 of: Proceedings of IEEE IN Workshop*. Bordeaux: France.

Byström, Leif. 1999 (October). *A Framework for the Connectivity Service Delivery Process*. Request for Proposal (RfP). TINA-C IPCM Working Group.

Cardoe, R., Finney, J., Scott, A. C., & Shepherd, W. D. 1999. LARA: A Prototype System for Supporting High Performance Active Networking. *Pages 117–131 of: IWAN*, vol. Springer Lecture Notes in Computer Science 1653. Berlin: Germany.

Case, J., Fedor, M., Schoffstall, M., & Davin, J. 1990 (May). *Simple Network Management Protocol (SNMP)*. Request for Comment 1157. IETF. Obsoletes RFC1098.

Case, J., McCloghrie, K., Rose, M., & Waldbusser, S. 1996 (January). *Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)*. Request for Comment 1907. SNMPv2 Working Group, IETF. Obsoletes RFC1450.

Chan, M., Huard, A., & Lim, K. 1996 (November). On Realising a Broadband Kernel for Multimedia Networks. *Pages 56–74 of: Proceedings of the Third International COST 237 Workshop*. COST 237, Barcelona, Spain.

Chandra, P., & et al. 1998. Darwin: Customisable Resource Management for Value-added Network Services. *In: Sixth IEEE International Conference on Network Protocols (ICNP'98)*. Austin, Texas: USA.

CHANDRA, P., FISHER, A., & STEENKISTE, P. 1998a (August). *Beagle: A Resource Allocation Protocol for an Advanced Services Internet.* Technical Report CMU-CS-98-150. CMU.

CHANDRA, P., FISHER, A., KOSAK, C., & STEENKISTE, P. 1998b. Network support for Application-Oriented QoS. *Pages 187–195 of: Proceedings of Sixth International Workshop on Quality of Service.* Napa, California: USA.

CHARLES UNIVERSITY. 1998 (June). *CORBA Comparison Project - Final Project Report.* Technical report edn. Prague: Czech Republic.

COHEN, A., & RANGARAJAN, S. 1999. A Programming Interface for Supporting IP Traffic Processing. *Pages Pages 132–143 of: IWAN*, vol. Springer Lecture Notes in Computer Science 1653. Germany.

DECASPER, D., DITTIA, Z., PARULKAR, G., & PLATTNER, B. 1998 (September). Router Plug-ins a Software Architecture for Next Generation Routers. *Pages 229–240 of: Proceedings of ACM SIGCOMM'98.*

DEERING, S., & HINDEN, R. 1995 (December). *Internet Protocol, Version 6 (IPv6) Specification.* Request for Comment 1883. Network Working Group IETF.

DORIA, A., HELLSTRAND, F., SUNDELL, K., & WORSTER, T. 2000 (July). *General Switch Management Protocol Version 3.* Tech. rept. GSMP Working Group, Internet Engineering Task Force (IETF). work in progress, expires January 2001.

DTI. 1982. *The Future of Telecommunications in Britain.* Department of Trade and Industry. White Paper.

FLORES, A.AND LECLUSE, C., & VAN LANDEGEM, T. 1996. Prototyping TINA based Services - The ALCIN Project. *Alcatel Telecommunications Review*, First Quarter.

FRY, M., & GHOSH, A. 1998 (June). Application Level Active Networking. *In: High Performance Protocol Architectures (HIPPARCH'98).*

FRY, M., & GHOSH, A. 1999. Application Level Active Networking. *Computer Networks and ISDN Systems*, Pages 655–667.

GALIS, A., PLATTNER, B., SMITH, J., DENAZIS, S., MOELLER, E., GUO, H., KLEIN, C., SERRAT, J., LAARHUIS, J., KARETSOS, G., & TODD, C. 2000 (October). A Flexible IP Active Networks Architecture. *Pages Pages 1–15 of: IWAN'2000*, vol. Springer Lecture Notes in Computer Science 1942.

GHOSH, A. 2000. *FunnelWeb v2.0.1.* `http://dmir.socs.uts.edu.au/projects/alan/`.

GUNTER, C., NETTLES, S., & SMITH, J. 1997. The SwitchWare Active Network Architecture. *IEEE Network Special Issue on Active and Controllable Networks*, **Vol 12**(Issue 3), Pages 29–36.

HANDLEY, M., SCHULZRINNE, H., SCHOOLER, E., & ROSENBERG, J. 2000 (August). *SIP: Session Initiation Protocol.* Internet Draft. SIP Working Group, Internet Engineering Task Force (IETF). work in progress, expires April 2001.

HICKS, M., & KEROMYTIS, A. 1999. A Secure Plan. *Pages Pages 307–314 of: IWAN*, vol. Springer Lecture Notes in Computer Science 1653. Berlin: Germany.

HJÁLMTÝSSON, G. 2000. The Pronto Platform: A Flexible Toolkit for Programming Networks using a Commodity Operating System. *In: Third IEEE Conference on Open Architectures and Network Programming (OPENARCH).* Tel Aviv: Israel.

IEEE. 2000 (18th-19th July). *Minutes of the P.1520.3 IP Subwork group meeting.* Santa Clara, California: USA.

IETF. 1981a (September). *Internet Protocol.* Request for Comment 791 edn. DARPA Internet Program Protocol Specification.

IETF. 1981b (September). *Transmission Control Protocol.* Request for Comment 793 edn. DARPA Internet Program Protocol Specification.

INTEL CORPORATION. *Intel Internet Exchange Architecture (IX-API).* Available by ftp from download.intel.com/design/IXA/ixawp-rev4.pdf.

INUOE, Y., MARTINE, L., & MOSSOTTO, C. 1999. *The TINA Book.* Prentice Hall Europe. ISBN: 0-13-095400-4.

ITU-T. 1991a. *Recommendation I.113, Vocabulary of Terms for Broadband Aspects of ISDN Rev. 1.* International Telecommunications Union.

ITU-T. 1991b. *Recommendation I.121, Broadband Aspects of ISDN, Rev. 1.* International Telecommunications Union.

ITU-T. 1992a (June). *Recommendation G.803, Architectures of Transport Networks Based on the Synchronous Digital Hierarchy (SDH.* International Telecommunications Union.

ITU-T. 1992b. *Recommendation M.3010, Principles of a Telecommunications Management Network*. International Telecommunications Union.

ITU-T. 1993. *Recommendation I.211, B-ISDN Service Aspects*. International Telecommunications Union.

ITU-T. 1994 (July). *B-ISDN ATM Adaptation Layer - Service Specific Connection Oriented Protocol (SSCOP)*. Recommendation q.2110 edn. International Telecommunications Union.

ITU-T. 1995a (June). *Recommendation G.805, Architectures of Transport Networks*. International Telecommunications Union.

ITU-T. 1995b. *Recommendation X.902, Open Distributed Processing, Reference Model Part 1:Overview*. International Telecommunications Union, ISO/IEC.

ITU-T. 1996. *Recommendation F.700, Framework for audiovisual/multimedia services*. International Telecommunications Union.

ITU-T. 1997. *Recommendation M.3040, TMN Management Functions*. International Telecommunications Union.

ITU-T. 1998 (November). *Recommendation G.164, Echo Suppressors*. International Telecommunications Union.

Juhola, A., Marshall, I., Covaci, S., Velte, T., Donohoe, M., & Parkkila, S. 1999. The Impact of Active Networks on Established Network Operators. *Pages 188–198 of: IWAN*, vol. Springer Lecture Notes in Computer Science 1653. Berlin: Germany.

Kohler, E., Morris, R., Chen, B., Jannotti, J., & Kaashoek, F. 2000. The Click modular router. *ACM Transactions on Computer Systems*, **18**(3), 263–297.

Lazar, A. 1997. Programming Telecommunications Networks. *IEEE Network*, **11**(5), Pages 8–18.

Lazar, A., Bhonsle, S., & Lim, K. 1994 (November). A Binding Architecture for Multimedia Networks. *Pages 103–123 of: Proceedings of Multimedia Transport and Teleservices International COST 237 Workshop*. Lecture Notes in Computer Science, no. 882.

LAZAR, A., BHONSLE, S., & LIM, K. 1995. A Binding Architecture for Multimedia Networks. *Journal of Parallel and Distributed Systems*, **30**(2).

LEGEDZA, U., & GUTTAG, J. 1998. Using Network-level Support to Improve Cache Routing. *In: 3rd International Caching Workshop*. Manchester: UK.

LINDELL, B. 2000 (October). Active Signalling Protocol (ASP) Execution Environment. *In: OPENSIG 2000*. OPENSIG, Napa Valley, California, USA.

MARSHALL, I., & BANFIELD, M. 2000. An architecture for application level active networks. *In: IEE Workshop on Application Layer Active Networks: techniques and deployment*. London: UK.

MARSHALL, I. W., CROWCROFT, J., FRY, M., GHOSH, A., HUTCHISON, D., PARISH, D. J., PHILLIPS, I. W., PRYCE, N. G., SLOMAN, M. S., & WADDINGTON, D. 1999. Application-level programmable internetwork environment. *BT Technology Journal*, **Vol 17**(Issue 2), Pages 82–94.

MCCANNE, S., & JACOBSON, V. 1993. The BSD Packet Filter: a New Architecture for User-level Packet Capture. *In: USENIX*.

MCKEE, P., & MARSHALL, I. 1999. Behavioural Specification Using XML. *In: FTDCS*. Cape Town: South Africa.

MEGGINSON TECHNOLOGIES. 1998 (May). *SAX: The Simple API for XML*. `http://www.megginson.com/SAX/SAX1/index.html`.

MICROSOFT CORPORATION. *Distributed Component Object Model Protocol - DCOM/1.0 Specification*.

MIKI, K., CAMPBELL, A., & KOUNAVIS, M. 1999 (July). *L-Interface for Routing Control*. Tech. rept. IEEE P.1520. P1520/TS/IP010.

MORRIS, R., KOHLER, E., JANNOTTI, J., & KAASHOEK, F. 1999 (December). The Click modular router. *Pages 217–231 of: Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP'99)*.

NATARAJAN, N., & ET AL. 1997 (May). *Network Resource Information Model (NRIM) version 3.0*. TINA-C.

NAVAL RESEARCH LABORATORY. 1995 (January). *VINCE 1.0 : Application Programmers Interface (API)*. Advanced Research Projects Agency (ARPA/CSTO) and the Naval Research Laboratory (NRL/CCS).

Object Oriented Concepts Inc. *ORBacus for C++ and Java, Version 4.0.* avaiable at `http://www.ooc.com`.

Oliver, H. 1997. TINA Connectivity Services: Quality and Quantity through QoS Based Routing and DPE Performance. *Pages 260–270 of: TINA'99: Global Convergence of Telecommunications and Distributed Object Computing.* Santiago: Chile, for TINA-C.

Oliver, H., Banfield, M., & Hutchison, D. 1997 (December). Distributed Connection Management for Real-Time Multimedia Services. *In: COST 237.*

OMG. 1997a (May). *OMG IDL Syntax and Semantics.* Object Management Group (OMG).

OMG. 1997b (September). *Unified Modelling Language: UML Summary Version 1.1.* Object Management Group (OMG). ad//97-09-03.

OMG. 1998 (June). *Control and Management of Audio/Visual Streams.* Object Management Group (OMG).

OMG. 2000a (June). *CORBA Security Services Specification Version 1.5.* Object Management Group (OMG). formal/2000-06-25.

OMG. 2000b (April). *Life Cycle Services Specification Version 1.1.*

OMG. 2000c. *The Common Object Request Broker: Architecture and Specification Version 2.4.* Object Management Group (OMG). Chap. 3, pages 3–1, 3–56.

Open Group. 1990. *DCE Overview, Report Number: OSF-DCE-PD-1090-5.*

Opensig. 2000 (October). *OPENSIG: Workshop on Programming the Internet.*

Parlay Group. 1998 (November). *The Parlay Group Intorduction and Objectives.* Parlay Technical Specification Issue 2.2. Parlay Consortium. `http://www.parlay.org`.

Parlay Group. 2000 (June). *Parlay API Specification 2.1.* `http://www.parlay.org`.

Partridge, C. 1992 (September). *A Proposed Flow Specification.* Request for comment 1363 edn. IETF.

Postel, J. 1981 (September). *Internet Control Message Protocol.* Request for Comment 792. DARPA Internet Program Protocol Specification.

Raguparan, M., & Vicente, J. 2000. *L- Interface Specification for IP devices.* Tech. rept. IEEE P.1520. P1520/TS/IP014.

Rimy, D., & Vouillon, J. 1998. Objective ML: An effective object-oriented extension to ML. *Theory And Practice of Objects Systems*, **4**(1), Pages 27–50.

Rooney, S. 1997. The Hollowman: An Innovative ATM Control Architecture. *Pages 369–380 of: IFIP/IEEE International Symposium on Integrated Network Management.* San Diego, California: USA.

Schenk, M., Jarboe, G., & Moerdijk, A-J. 1999 (December). *Open Service Marketplace Model.* Technical White Paper. OMG Telecommunications Domain Taskforce. telecom/99-12-03.

Schmid, S. 2000. *LARA++ Design Specification.* Internal Report MPG-00-03. Lancaster University. Available by ftp from `ftp.comp.lancs.ac.uk`.

Schmid, S., Finney, J., Scott, A., & Shepherd, D. 2000a. Active Component Driven Network Handoff for Mobile Multimedia Systems. *Pages 266–278 of: Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS)*, vol. Springer Lecture Notes in Computer Science 1905. Enschede: The Netherlands.

Schmid, S., Finney, J., Scott, A., & Shepherd, D. 2000b. Component-based Active Networks for Mobile Multimedia Systems. *In: Proceeding of NOSSDAV 2000.* USA.

Semeria, C. *Internet Backbone Routers and Evolving Internet Design.* Juniper Networks Inc. `http://www.juniper.net/techcenter/techpapers/200002.html`.

Shenker, S., Partridge, C., & Guerin, R. 1997 (September). *Specification of Guaranteed Quality of Service.* Request for Comment 2212. IETF.

Simpson, S. 2000. *Component Compatibility Markup Language.* Website at `www.comp.lancs.ac.uk/computing/research/mpg/projects/pan/ccml/`.

Simpson, S., Smith, P., Banfield, M., & Hutchison, D. 2000. Component Compatibility for Heterogeneous Active Networking. *In: IEE Workshop on Application Layer Active Networks: techniques and deployment.* London: UK.

Steegmans, F., & et al. 1997 (February). *Network Resource Architecture version 3.0.* TINA-C.

Steegmans, F., Kin, H., Natarajan, N., & Rajahalme, J. 1997 (November). Managing TINA Streams that use Internet Transport. *Pages Pages 111–118 of: TINA'97: Global Convergence of Telecommunications and Distributed Object Computing*. TINA-C, Santiago, Chile.

Sun Microsystems. *Java 2 SDK, Standard Edition Documentation v1.3.* `http://java.sun.com/j2se/1.3/docs/index.html`.

Tateson, J. 2000. Scatter Routing. *In: Multi-Service Networks 2000.*

Tennenhouse, D.L., Smith, J.M., & et al. 1997. A Survey of Active Network Research. *IEEE Communications Magazine*, **Vol 35**(Issue 1), Pages 80–86.

TINA-C. 1997a (February). *Business Model and Reference Points Version 4.0.*

TINA-C. 1997b. *Computational Modelling Concepts.*

TINA-C. 1997c (February). *Connectivity Service (ConS) Reference Point Version 1.0.*

Van der Merwe, J., & Leslie, I. 1997. Switchlets and Dynamic Virtual ATM Networks. *Pages Published in Integrated Network Management V, Chapter 57, Pages 355–368 of: IFIP/IEEE International Symposium on Integrated Network Management*. San Diego, California: USA.

Van der Merwe, J., Rooney, S., Leslie, I., & S., Crosby. 1998. The Tempest - A Practicle Framework for Network Programmability. *IEEE Communications.*

van der Merwe, J.E., Sreenan, C.J., Donnelly, A.N., Basso, A., & Kalmanek, C.R. 2000. A Network based replay portal: service scenario, architecture and implementation. *Pages Pages 521–540 of: Telecommunication Network Intelligence.* Sixth International Conference on Intelligence in Networks (SmartNet 2000). Vienna: Austria.

van Tilborg, E., & Schuring, O. 2000 (June). *Caspian Business Model.* Internal Report. EURESCOM P926.

Vanet, G, & Kiriha, Y. 1999. A Self-Configuring Data Caching Architecture Based on Active Networking Techniques. *Pages Pages 85–96 of: IWAN*, vol. Springer Lecture Notes in Computer Science 1653. Berlin: Germany.

Vicente, J., & et al. 2000 (January). *Proposal for IP L-Interface Architecture.* Tech. rept. IEEE P.1520.3. P1520/TS/IP013.

VICENTE, J., CAMPBELL, A., DENAZIS, S., & MIKI, K. 1999 (March). *Designing IP Router L-Interfaces.* IEEE P.1520 Deliverable Version 3. P1520/TS/IP005.

WETHERALL, D., GUTTAG, J., & TENNENHOUSE, D. 1998. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. *In: IEEE OPENARCH.* San Francisco, California: USA.

WROCLAWSKI, J. 1997 (September). *Specifciatiom of the Controlled-Load Network Element Service.* Request for Comment 2211. IETF.

YEMINI, Y., & DA SILVA, S. 1996 (October). Towards Programmable Networks. *In: IFIP/IEEE International Workshop on Distributed Systems.*