Department of Electronic and Electrical Engineering
University College London
University of London

THE EVOLUTION OF CONTROL ARCHITECTURES TOWARDS NEXT GENERATION NETWORKS

By
Christos Solomonides BEng(Hons), MRes(Distinction)

7 September 2001

A thesis submitted to the University of London
for the Ph.D. in Electronic and Electrical Engineering

ABSTRACT

This thesis describes the evolution of control architectures and network intelligence towards next generation telecommunications networks. Network intelligence is a term given to the group of architectures that provide enhanced control services. Network intelligence is provided through the control plane, which is responsible for the establishment, operation and termination of calls and connections.

The work focuses on examining the way in which network intelligence has been provided in the traditional telecommunications environment and in a converging environment. In the case of the traditional telecommunications environment, the thesis examines the Intelligent Network (IN) architecture as a case scenario. In the case of the converging telecommunications environment, the work focuses on examining the relation and impact of emerging architectures and protocols and the ways in which these can inter-work with the IN. The discussion is presented using a taxonomy reference model of network intelligence architectures and their relation to the IN. For example, a protocol based on existing IN capabilities is presented that allows end users to engage in electronic commerce without the need for credit cards.

The control plane architecture in the Public Switched Telephony Network (PSTN) is heavily based on state machines. The role of state models and the reliance of IP-based protocols on state models are also examined. For this, IP-based architectures are examined and the extent of state utilisation is presented. This enables a classification of IP-based architectures and protocols to be drawn with regard to state utilisation.

The role of existing network intelligence within the context of open programmable networks and application servers is also examined. The work identifies the need for a common communications framework between third-party service providers. This is the focus of the API server architecture, which draws from IN concepts and from approaches in the IP domain.

Dedicated to my parents

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation and gratitude to my supervisor Dr. M. D. Searle for his constant support, encouragement and guidance throughout this project. Without his help this work would never have been completed.

I would like to express my most sincere gratitude to my family for their unfailing support. To my Father who has provided me with the enthusiasm to complete this work. To my Mother for her continuous encouragement that has enabled me to successfully complete this work. Without the moral and financial help of my parents I would never have undertaken this work and for this I dedicate the work to them. I would like to extend this to my brother for always being there and supporting me at difficult times.

My deepest heartfelt appreciation is offered to my wife Vania Bastajian for her patience and support.

I would sincerely like to thank Adonis Michaelides for his support and genuine friendship.

I would also like to sincerely thank the following people: Aram Bastajian for taking the time to proof read the text; Christos Drakos for his support; Aristos Michaelides for his sincere encouragement; Jason Spencer and Ognjen Prnjat for their friendship and support; Chryssanthos Kouriefs and Melina Georgiou for providing me with hours of entertainment and helping me cope with the work.

My sincere thanks go to all the staff at INSIG Ltd for allowing me to engage in activities that have provided me with the practical experience that compliments the theoretical aspects of this work.

Finally the financial support provided by Martin Stanley and the Department of Electronic & Electrical Engineering at University College London.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Changes in telecommunications technology and the telecommunications environment provide sufficient ground for the proliferation of open service provisioning by third-party service providers. Such advances give rise to a number of issues relating to the control plane of telecommunications networks.

1.1 MOTIVATION

Historically, telecommunications services were designed, developed and implemented with the traditional telecommunications network as the target environment. This is an environment with strictly standardised functional entities that are owned and maintained by the incumbent operator.

The telecommunications environment is changing: market dynamics, regulatory initiatives and technological advances are necessitating changes [Melo97] [EC00] [P1110] [Haya98] [Ishi98]. Such technological and market-related [Walk97] changes have an impact on the way in which control is provided. The control plane can be thought of as a collection of hierarchically distributed functional entities whose aim is to establish, monitor and terminate the calls and connections. In the traditional environment, control and network intelligence is provided primarily by the Intelligent Network architecture. In the new telecommunications environment it is crucial for traditional telecommunications networks and services to inter-work with IP-based networks and services [Stro99]. The inter-operability and, therefore, the convergence process is driven by the demand of end users for seamless service access, regardless of the transport network. Such a level of inter-operability requires a re-think in the way network intelligence is provided.

IP-based networks have shown explosive growth [Asat01][ITU97][ITU00][EC00]. Unlike in the telecommunications domain, the drive for new protocols, services, and capabilities has come from open groups, rather than from standardisation bodies. The IP approach does not require the presence of an explicit control plane architecture that standardises external and internal state models; rather, the client–server architectures of the IP world are sufficient. This is supported by the way standardisation bodies such as the Internet Engineering Task Force (IETF) operate.

1.2 AIMS AND OBJECTIVES

The thesis examines and identifies the control plane mechanisms that enable network intelligence in both telecommunication and data networks. In telecommunication networks, specifically the public switched telephony network (PSTN), a major architecture contributing towards control and intelligence is that of the Intelligent Network (IN). The thesis examines the IN architecture as a case scenario in order to pinpoint the mechanisms through which it enables services and therefore intelligence to be provided in a complex and distributed network such as the PSTN.

The work also aims at examining how the traditional view of network intelligence is affected in the converging environment of telecommunication and data networks. To achieve this, the thesis examines current and evolving IP-based architectures and suggests ways in which inter-operability with network intelligent architectures in the PSTN can be achieved. By doing this, the author presents a taxonomy that classifies numerous packet-switched protocols and their relation to the IN architecture.

The thesis also puts forward a classification of existing IP-based protocols according to their utilisation of state and state-dependent actions. The work shows the way in which state can be implemented and discusses the need for state-utilisation. More importantly, the author identifies some of the complexities that necessitate state-utilisation.

Finally, the thesis looks at the issues relating to the control plane within the framework of open programmable networks and initiatives for service provisioning by third-parties. The aim is to present an architecture for third-party service providers that draws from IN principles for providing control and intelligence.

1.2 THE PROBLEM FIELD AND THE APPROACH

The problem field that this research work focuses on is that of the control plane in next generation telecommunications networks. The control plane is of crucial importance in the converging environment of telecommunication and data networks. Unless there is a clear understanding of the way in which network intelligence can be offered – through the control plane – the inter-working of telecommunications services will be problematic. The problem statement can be summarised as: *"What is the role of network intelligence and the control plane in next generation telecommunications networks?"*

The approach the author has adopted is one that represents a mixture of practical and theoretical work. The theoretical work focuses on the role of the control plane in traditional and evolving telecommunications networks and services. The practical work involves the implementation and simulation of three systems, the aims of which were to: examine the role of existing intelligent network infrastructure; analyse the role of the control plane in a distributed system in an IP-based environment; and to propose an application server architecture that is based on concepts from the IN world, but also draws from the flexibility and openness of the IP domain.

1.3 Summary of Main Contributions

The thesis presents both traditional and emerging network intelligence architectures within the converging telecommunications environment. By doing this it provides a **unique view of the current state of convergence**. This is a useful and important contribution. The classification presented in chapter 3, provides the reader with a chronological snapshot of the convergence between telecommunication and computer science practices, principles and approaches. Such a classification has not been published before, and it is the author's view that the taxonomy is useful for the **categorisation of new computer science and telecommunication-based architectures and their relation to the Intelligent Network**. At the same time, such a classification is important as it enables a reader with little background in either of the two fields (telecommunications or computer science), to quickly understand the relation of existing network intelligent architectures to the Intelligent Network.

The author also presents an application for e-commerce, based on the Intelligent Network architecture, with support for micro-transactions without the need of credit cards. The application was **designed, implemented and simulated by the author**. Furthermore, a **patent application was also filed** [IEPSPat]. The application is entirely based on existing IN capabilities, and requires no change in the telecommunications environment.

Another important contribution is the **classification of IP-based architectures with respect to their utilisation of state**. The author scrutinises numerous IP-based protocols and frameworks and shows that although some systems may be characterised as "stateless", the complexity of their operating environment may impose the introduction of "state" and/or "state dependent actions and events". Specifically, the author argues that in certain systems, the utilisation of state is imposed by the concurrency complexities of the system and not the complexities of the system *per-se*.

The thesis also puts forward an **architectural framework for the communication of third-party service providers** within an open network service provisioning environment. The framework, proposed by the author, conforms to the Parlay API's framework interfaces for allowing the interconnection of third-party service providers. The architectural framework is based on the principles of the IN control architecture and also on the flexibility exhibited by IP-based architectures. Incorporated within the framework are state machines that aim and achieve to simplify the billing and management of services in an open network environment. The framework is necessary as new third-party service providers are

The work presented in this thesis is supported by the following publications of the author:

1. C. Solomonides and M. Searle, "*IN and the INternet*", University College London, London Telecommunications Research Symposium, London, July, 1998.
2. C. Solomonides and M. Searle, "*Relevance of Existing Intelligent Network Infrastructure to the Internet*", University College London, Lecture Notes in Computer Science, Springer-Verlag, 5th International Conference on Intelligence and Services in Networks, IS&N'99, Barcelona, Spain, April 1999.
3. C. Solomonides and M. Searle, "*An Intelligent Network E-Commerce Protocol*", University College London, 38th European Telecommunications Congress, Utrecht, The Netherlands, August 1999.
4. C. Solomonides and M. Searle, "*Evolution Towards the TINA Service Architecture Through PINT and Parlay*", 6th International Conference on Intelligence in Networks, 17–20 January 2000, Palais Des Congres D'Archachon, Bordeaux, France, 2000.
5. C. Solomonides and M. Searle, "*Intelligent Network Application Programming Interface Server Architecture*", University College London, IEEE IN 2000 Workshop, Cape Town, South Africa, May 2000.
6. C. Solomonides and M. Searle, "*Network Intelligence, APIs and Service Creation*", University College London, 39th European Telecommunications Congress, Limerick, Ireland, August 2000.
7. C. Solomonides and M. Searle, "*Application Server Architecture for Open Networks*", University College London, London Telecommunications Symposium, London, September 2000.

1.4 THESIS OUTLINE

Chapter 2 provides a short background to telecommunications concepts that are relevant to this thesis and discusses the traditional approach to network intelligence. The Intelligent Network is presented and the mechanisms through which it provides the control plane for telecommunications networks are analysed.

Chapter 3 aims at examining current research activities in network intelligence and presents these in relation to the Intelligent Network architecture. It also examines the new role of the Intelligent Network: that is, the IN as a control architecture that provides ways to initiate services on the PSTN. Within this context, the chapter discusses ways of using the IN as it currently is but also identifies the additional functionality that is needed within a converging environment.

One of the areas in which the existing IN control architecture can be used unchanged is for providing authentication, certification and electronic commerce capabilities. This is examined in chapter 4, where a specific application area uses legacy IN to provide a desirable, secure and trusted platform for IP-based payments. The strength of the protocol that is described lies in its ability to handle micro-transactions on behalf of the end users, without the need for credit cards. The implementation and simulation of the protocol is presented in chapter 5.

Chapter 6 examines the notion of "state" in IP-based protocols and architectures. The chapter presents the ways in which state can be maintained and implemented. Following this the chapter provides a classification of various IP-based architectures with respect to their reliance on state-dependent actions. The work presented in chapter 6 is applied in chapter 7 by presenting a network management system which aims at examining the control aspects in a distributed system. The network management system presented in chapter 7 draws from traditional telecommunication principles, but also from techniques traditionally found in the data networks environment.

Chapter 8 presents an application-server architecture based on principles from the IN environment, but also from computer-science practices. The application server that is presented in aimed towards third-party service providers. The chapter examines the need for such an architecture within the overall framework of open network service provisioning.

Finally, chapter 9 gives the conclusions regarding the research work presented and provides suggestions for further work that may be undertaken as a result of the author's work.

CHAPTER 2

TRADITIONAL APPROACH TO NETWORK INTELLIGENCE

The aim of this chapter is to develop an understanding of the traditional role of network intelligence in telecommunications. This is done by outlining some key concepts such as switching, the role of signalling and their relation to control architectures such as the Intelligent Network. In doing this, the chapter presents a useful background to the new research conducted.

2.1 INTRODUCTION

In chapter 1 it was mentioned that one of the aims of the work is to examine the role of the control plane in traditional telecommunication networks. Chapter 2 supports this aim by examining the way in which control has been achieved in traditional telecommunications networks. Hence, a large section of this chapter deals with the control mechanisms and architectures that form part of today's telecommunications networks.

Network Intelligence is a term given to the group of architectures that provide enhanced control services. A number of these will be examined in this thesis. One important example of network intelligence is called the Intelligent Network (IN) [Q.1200–Q.1205], discussed in this chapter. The IN is implemented as an overlay to the voice network and facilitates advanced call control services through its capability sets (CSs).

Telecommunications networks have been developed to support voice services. The basic services of such networks are discussed in section 2.1.1, while section 2.1.2 discusses the way service logic is distributed in an IN and non-IN environment.

Section 2.1.3 gives a brief history of the development of the telecommunications network. The motivation for providing a historical walkthrough is to understand that the evolution of a telecommunications network has been driven by the need to provide voice services with an extremely high quality of service. This basic requirement has also driven the characteristics of the traditional telecommunications network. The fundamental characteristics of a telecommunications network are at odds with the features of an Internet Protocol network.

A telecommunications network typically comprises a number of planes. From the point of view of this thesis, the main ones are the transport and control plane. The transport plane provides the bearer connections on which voice channels are transported and the control plane supports the bearer channels through the signalling services. Section 2.2 examines the control plane and, in particular, the switching and signalling aspects of a telecommunications network. The key signalling protocol is Signalling System No. 7, which is discussed in section 2.3.

The IN control architecture is presented in detail in section 2.4, where the focus is to identify the major characteristics of any network intelligent architecture. An essential element of the IN is its reliance on state models, which will become apparent through the discussions in this chapter.

While architectures such as the IN focused on standardising the technical issues, others such as the Telecommunications Information Network Architecture (TINA) [TINA-GA] had a wider scope, attempting to address and standardise less technical issues such as the business and managerial structures [TINA-BM]. The TINA model is presented in section 2.5. However, the focus is on the relation of IN to TINA, rather than the TINA architecture *per se*. Finally, section 2.6 focuses on the use of state models in telecommunications networks.

2.1.1 Introduction to Telecommunications Networks

The aim of any communications network is to provide channels for the exchange of information.* In the traditional telecommunications world, the aim of the communications network has been to provide for the transfer of speech. This has driven the structure and characteristics of the network.

Telecommunications networks have evolved significantly since their original conception even though, to a certain extent, the basic service offering has remained unchanged throughout this time. This is because the design requirements of the telephone network were set by the characteristics of human voice, which have not changed.

The telecommunications network is referred to as the "Public Switched Telephony Network" (PSTN). This has evolved from the traditional "plain old telephony service" (POTS) and adds

* In traditional telecommunications involving voice, these "channels" can be thought of as physical connections between the two subscribers.

new services such as freephone, credit card calling, etc. The rest of this section presents the basic concepts of the PSTN.

A network involves the interconnection of resources. The network topology dictates the number of links that are required, the cost of the deployment, and possibly, the robustness of the network. As the network becomes distributed across geographical regions, there is a need for this interconnection if two or more networks are to communicate.

A subscriber in the PSTN can place a call from any geographical location to another subscriber who might be located thousands of miles away. This can be achieved using a hierarchical architecture of interconnected network components (see figure 2.1) between any two parties involved in a conversation. Having achieved the interconnection of these geographically distributed networks, there is a requirement to notify network components (residing in the core of the network) so that one subscriber can initiate a call to another. This is achieved using switching and signalling. The process of switching and signalling results in resources (channels) being allocated between the calling (originating) and the called (terminating) parties. This pre-allocation of resources for the duration of a call[*] is a fundamental principle in telecommunications networks. The PSTN approach is in stark contrast to the approach of data networks such as the Internet, where no pre-allocation of resources takes place. The details involved in switching and signalling are the focus of the work presented in section 2.2.

Making a connection and allocating the resources requires a call to proceed through the following phases:

1. *Pre-selection*: where a new call request is recognised and decisions on how to deal with the call are made.
2. *Call completion:* the originating and terminating parties are connected and the charging process is initiated.
3. *Conversation*
4. *Release:* the call is disconnected, releasing the allocated resources, the billing record is completed, and the network returns to the normal (idle) state.

[*] When channel-associated signalling is used (section 2.2.3), the resources are allocated when the subscriber's handset goes off hook and the "*Setup*" signal is received by the local exchange.

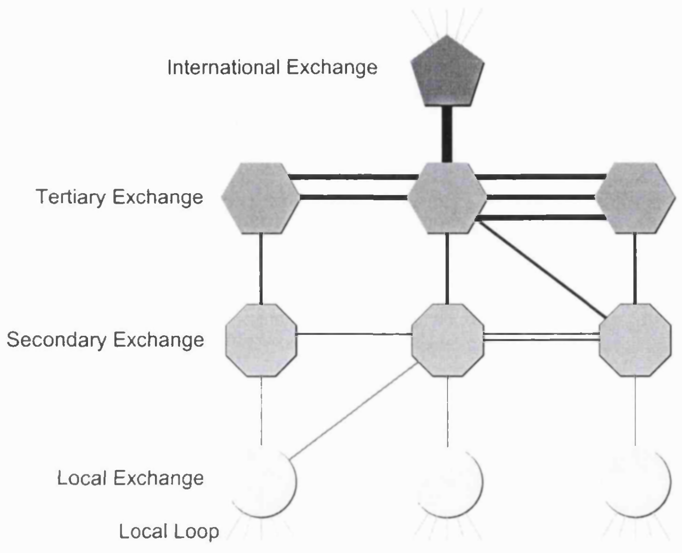


***Figure 2.1:** The hierarchy of the PSTN network*

2.1.2 Distribution of Service Logic

This section discusses the way in which service logic is distributed in two environments. Figure 2.2 illustrates the implementation of a service without an IN control architecture; figure 2.3 illustrates the way a service is implemented with an IN architecture. The section discusses the two approaches to service distribution and implementation and puts forward the principles, advantages and basic operation of the IN.

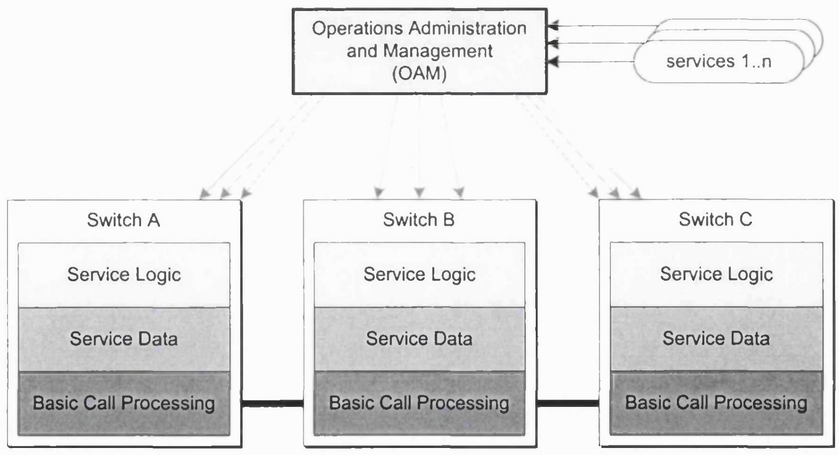


***Figure 2.2:** Switch-based service provisioning in the traditional POTS environment adopted from [Mage96]*

Figure 2.2 depicts the components of the switches and the way service logic is distributed without the IN control architecture. The switches enable the connection of subscribers located at the edges of the network. Its components are the service logic, service data and basic call-processing functionality.

The implementation of services without an IN control architecture has a number of disadvantages [Ambr89][Eple90][Zolz92][Filk00]. The switches may be from different vendors and, as a result, the service logic and data have to be customised for the vendor-specific platform. To introduce new services, the service logic has to be implemented on all the switches and this task is costly due to the geographical distribution of the switches. Subscribers can only access services implemented by the local switch, unless the service logic is identical throughout the network.

To achieve homogeneity at the switch-level means that a substantial proportion of the operator's revenue and time is spent in updating the service logic at the switches. This is costly, time-consuming, and inefficient and, unfortunately, it was the only approach available until the introduction of the Intelligent Network architecture.

With the introduction of the Intelligent Network, it became possible to implement a service in a more efficient manner, as shown in figure 2.3.

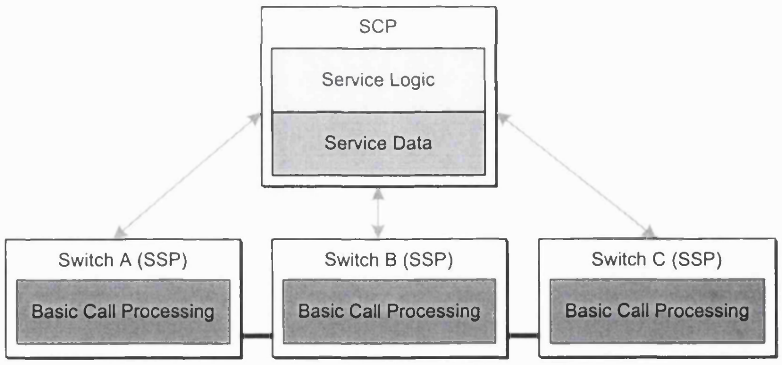


Figure 2.3: *Service provisioning in an IN environment*

The service logic now resides in a central node, called the Service Control Point (SCP). When the individual switches require additional service functionality, that may not present in the switch, it is provided by the SCP.

The separation of basic call-processing from enhanced service logic enables the rapid provisioning of new services [Kun97]. This is because service logic is now centrally located, and there is no need to individually update every single switch; a single update on the SCP is sufficient.

To illustrate the operation of an IN service, the freephone service[*] is presented in figure 2.4. The following communication channels are used: communication between the Service Switching Point (SSP) and the SCP/Service Data Point (SDP) is achieved via the signalling network, while communication across the SSPs (switches) is over the bearer connection. The signalling links are shown in figure 2.4 using dashed lines while the bearer connection is represented using the solid line.

The process begins with the initiating party dialling "0800192192". The local exchange (switch A) recognises that this is a specific service access number as it begins with "0800". It therefore queries the SCP for call handling support. The SCP then queries the SDP, which contains a database, in order to translate the "0800192192" logical number to the specific destination number, for example "02073342123". This translated number is passed back to switch A and call handling resumes. Switch A now has sufficient information to be able to set up the call, i.e. to establish the corresponding bearer connection to the appropriate final destination. When either of the two subscribers hangs up, the call is terminated in the usual manner (i.e. by moving to the *Release* phase).

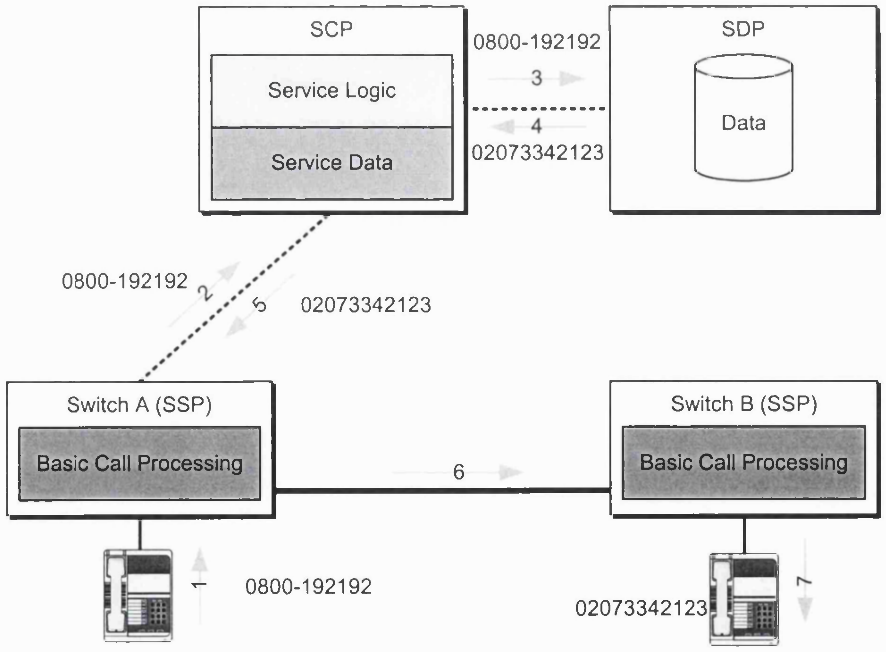


Figure 2.4: *IN implementation of freephone service*

The freephone service proved to be a desirable service with network operators and customers alike. Indeed, British Telecom's reason for adopting the IN architecture was to enable freephone

[*] Freephone enables a calling party to dial a number without incurring charges. The call is charged to the terminating party.

and call-stream services [OBri89]. NTT in Japan, introduced an IN-like architecture in 1985, and the first service that was made available was freephone [Suzu93].

This introductory section provided a short description of the various elements that are involved in a modern telecommunications network. In summary, the main concepts that were introduced include the **hierarchical and distributed nature of the network**, the presence of **a separate signalling network** to that of the bearer connections, the fundamental requirement of the sharing of resources and their **explicit allocation**, **service provisioning** with and without the Intelligent Network, and some of the advantages of the **Intelligent Network**. These concepts are further discussed in the following sections in this chapter.

Firstly, however, there is a short chronological walkthrough of the telecommunications developments up to the point of the introduction of the Intelligent Network to show that, at the initial conception of the telecommunications network, there was intelligence at the core, which was then removed.

2.1.3 Historical Switching and Signalling

The aim of the POTS network was to provide the capability for voice communication among subscribers. At the time (late 1870s), the approach adopted was to interconnect subscribers through local offices. Within these offices, human operators were responsible for physically connecting the circuit between the caller and the destination subscriber using a switchboard.* This process is now known as **circuit switching.**

The presence of the human operator provided the "network" with a very significant capability – that of intelligence. Operators were in a position to know that the doctor was not at home, but at a friend's house – and in this way could re-direct the call. In today's environment, this is call forwarding. Furthermore, unwanted calls were filtered (call-screening) and messages could be left with the operator (voice-mail). When either of the two parties ended the conversation, the operator would terminate the connection. This would release the bearer channels and resources were no longer explicitly allocated to the two parties. It can be deduced that the explicit allocation of resources has had its roots in the original conception of the network.

While the services described above were provided by the human operator, other services, for example, call queuing and call waiting were more difficult (if not impossible) to achieve, because of the limitation in the number of resources (bearer connections) and not due to lack of

* At the time, subscribers were important people in the community, such as doctors, policemen, etc.

intelligence. What is more, as the number of subscribers grew, operators struggled to keep up with the demand for switching [Russ98]. It then became evident that if the switching and connection process could be automated, it would provide lower running costs and therefore increased profits. Increasing the speed of the connection process also meant subscribers would spend more time talking rather than waiting to be connected.

The automation of the connection process required a component that could automatically switch and connect the originating and terminating parties. Unfortunately, the introduction of automated switches also removed the intelligence that was present.

2.1.4 The Strowger Switch

The first automated switch came from an unexpected source. Almon B. Strowger was an undertaker in Kansas City, USA in the late 1890s. The story says [AGCS] [Lesh98] [TeRe00] that there was a competing undertaker locally, whose wife was an operator at the local telephone exchange. Whenever a caller asked to be put through to Strowger, calls were deliberately put through to his competitor. This obviously frustrated Strowger greatly and he set about devising a system that would reduce the reliance on the human part of the equation. Strowger developed a system of automatic switching using an electromechanical switch based around electromagnets and pawls [AGCS]. With the help of his nephew, Walter S. Strowger, he produced a working model in 1888 under US Patent No. 447918 dated 6/10/1891.

There was now however a need to re-introduce the network intelligence and the "advanced services" that were removed. Of course, the electromechanical switch developed by Strowger did not provide the necessary "environment" to be "software-controlled". Software within switches was introduced with the development of Stored Program Control (SPC).

By the early 1960s, advances in computer science* made it possible to develop software which could control devices. In May 1965, the No. 1 Electronic Switching System was put into commercial service in Succasunna, New Jersey. The switch utilised technology with a relatively slow central control, very expensive memory (Ferrite sheet and Twister), and slow peripherals [Memm90]. These factors adversely affected the development of SPC-type switches – the competing crossbar systems were cheaper to manufacture.

In the early 1980s, digital switching began to appear and distributed control was more widely adopted – together with its overheads [Memm90]. Multiple control programs were accompanied

* Such as the introduction of the silicon transistor by Texas Instruments.

by multiple distributed databases and interactions among distributed programs and databases were a challenge. The software development practices in the 1980s did not exactly follow the software engineering principles of abstraction, coherence or decoupling [Somm01]. Instead, software was characterised by the "spaghetti approach" and endless use of the "goto" keyword.

The rest of this chapter focuses on discussing the role of switching and signalling, followed by a detailed description of the IN control architecture.

2.2 Switching and the Role of Signalling

In an ideal world, communicating parties would be physically interconnected across a medium with no loss, delay or bandwidth limitations. In practice it is impossible to physically interconnect all the communicating parties in a mesh arrangement due to scalability, practicality and logistics.

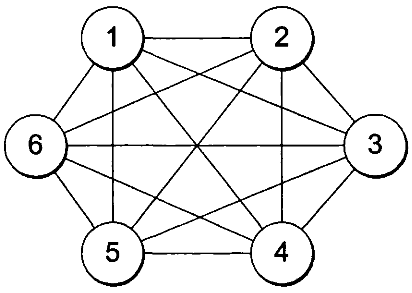


Figure 2.5: *A fully interconnected mesh topology*

Equation (1) presents the total number of links that are required if *n* nodes are to be connected in a fully-meshed arrangement, as shown in figure 2.5.

$$\frac{n^2 - n}{2} \qquad (1)$$

This is expensive if the number of nodes is large, as in the case of the PSTN where the number of nodes is in the order of millions. For this reason, from its infancy the telecommunications network has adopted a hierarchical approach (see figure 2.1).

In order to enable the network to interconnect two subscribers in a hierarchical topology, switches (that are controlled through signalling) must be placed along the transmission path to direct the bearer connections from the caller to the person being called. The process of **switching** allows the physical interconnection of subscribers by directing a call path in a way that enables the caller to establish a bearer connection to the party being called [Russ98]. The size and position of the switches are carefully chosen using network planning and performance measurements, as their cost is very high.

The **signalling network** is used to control the switches and its aim is to establish and maintain control connection paths through the link-by-link setting up of a path [Russ98]. In doing so, valuable resources are allocated by the switches to establish the connection. To maintain the signalling network in an operational state and increase robustness, some of the links at the various levels are interconnected (see figure 2.1).

The interconnection increases robustness in the case of link failure at the physical and transport planes. However, due to the importance of the signalling network it is also vital that the signalling *protocol* maintain mechanisms that provide resilience and robustness.

Signalling protocols such as Signalling System Number 7 (discussed in section 2.3) define ubiquitous state models to assist in ensuring the robustness and reliability of the signalling network. The concept of state models is a central issue to the work presented in this thesis (see particularly sections 2.6, 3.5, 6.6 and 7.3).

The signalling network is divided into two parts: the access network and the core network.

2.2.1 Access Network

The **access network** is defined as the part of the network between the end terminal and the local exchange. Access network signalling in the POTS network utilises a very simple analogue signalling scheme called Dual Tone Multiple Frequency (DTMF) [ETSI TS 101 235]. DTMF signalling uses a set of audible frequency tones in response to buttons being pushed on the telephone. The goal is for the telephone device to signal to the exchange across the user channel.

In the case of private businesses, the customer premises do not have a single telephone line but a large number of telephones in a private network. In such cases, a business often possesses a Private Branch Exchange (PBX). PBXs control the routing of calls within the business organisation and provide customised services between offices of the same company. These services include the allocation of personal number extensions across multiple office sites. The Digital Private Network Signalling System (DPNSS) is a scheme that interfaces the PBX with the local exchange. Access signalling has differing requirements from the trunk (core) network.

2.2.2 Core Network

The **core network** is the part from the local exchange to internal signalling points, such as Signal Transfer Points (STPs). Core (inter-exchange) signalling between exchanges or switches establishes the resources for a call and optionally communicates supplementary service requirements to an intelligent network platform, such as the Intelligent Network. The core network uses core signalling schemes and protocols. SS7, the ubiquitous inter-exchange signalling scheme, has evolved from a previous version named Common Channel Signalling System No. 6, developed by the ITU-TS (formerly CCITT) in the mid-1960s. SS7 employs digital signalling.

Early signalling methods were limited because they used the same circuit for both signalling and voice. The circuit would be busy from the time the caller started dialling until the caller went "on-hook". A solution to this was to separate the signalling and the bearer connections. This way, the call setup and teardown procedures required with every call could be faster. Voice and data circuits could be reserved for use when a connection was possible, instead of maintaining the connection even when the destination was busy.

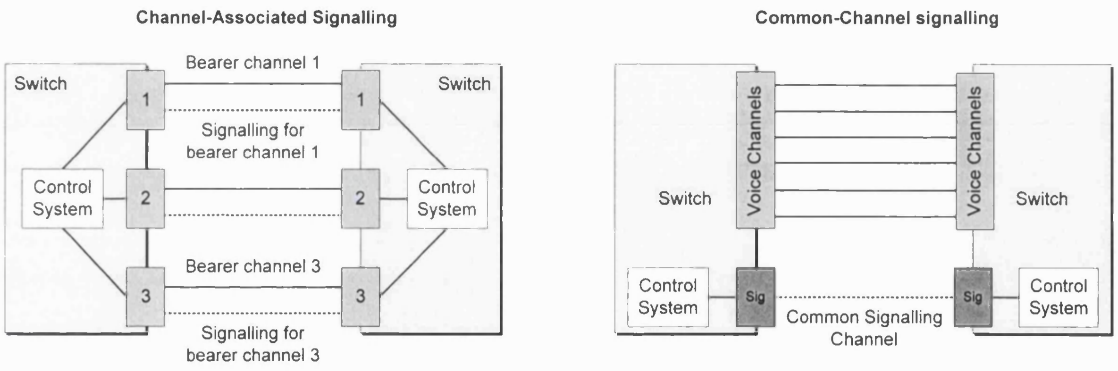


Figure 2.6: Channel-associated and common-channel signalling

2.2.3 Channel-Associated Signalling

In **channel-associated signalling** (figure 2.6, left), the signalling messages are sent across logical signalling channels that are dedicated to each speech channel. In other words, although the signalling channel may be a separate physical channel from the voice (bearer) channel, a dedicated channel is allocated to support the voice call. The disadvantage of this technique is that signalling resources are provided regardless of whether the voice channel needs it. There are times in a call, such as the mid-call period, when less signalling information is required. At such times it would be useful if the signalling channel could be used for another signalling circuit.

2.2.4 Common-Channel Signalling

In **common-channel signalling** (figure 2.6, right), signalling for a number of voice channels is aggregated into a single shared (common) signalling channel. The introduction of separate signalling links means that a bearer (voice) connection is only utilised if a connection can be established. As a result, the availability of voice circuits is higher and the need for additional circuits decreases. Common-channel signalling schemes include SS6 and SS7, as well as the DPNSS and its successor QSig [ECMA-143].

2.3 Common-Channel Signalling System No. 7

Figure 2.7 depicts the Common-Channel Signalling System No. 7 (SS7) protocol stack in relation to the OSI reference model [ISO-IS7498-1]. In contrast to IP networks, SS7 networks do not adhere to the full seven-layer OSI reference model. This is because SS7 was developed before the OSI reference model. SS7 adheres to a four-level model.

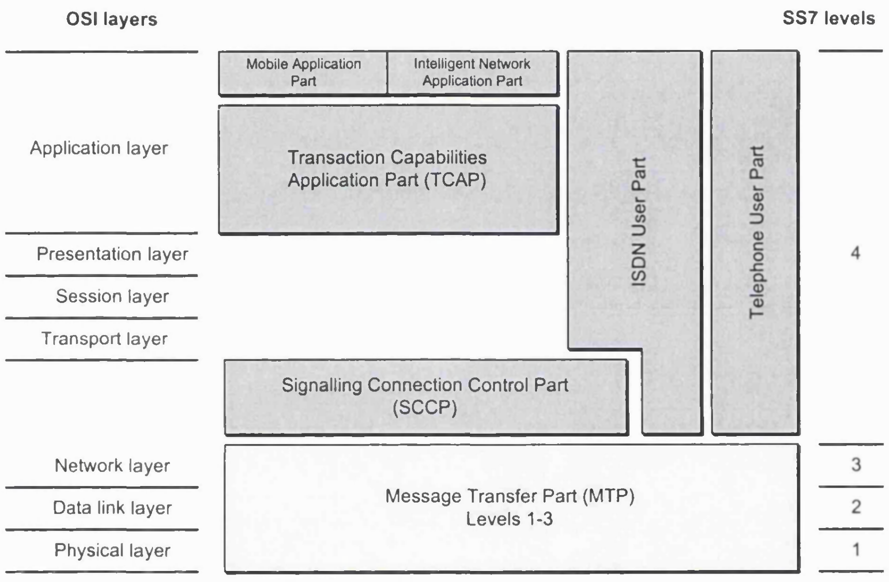


Figure 2.7: *The SS7 protocol stack in relation to the OSI reference model*

- The **Message Transfer Part Levels 1–3** [Q.701] define the physical transport, the data link layer and the network layer. As in the OSI model, each layer is responsible for delivering a service to the layer above it. In the case of call-connection, MTP-3 provides an end-to-end packet-based transfer that routes signalling packets based on a unique element address called a Point Code (PC).
- The **User Parts** are shown as the shaded areas on the right. The **Telephony User Part** (TUP) [Q.721–Q.725] is a set of well-defined messages for the establishment, conversation and termination phases of a call. Many national variants of TUP exist including the BTUP in the UK and SSUTR2 in France. In the USA, the **ISDN User Part** (ISUP) [Q.761–Q.767] is used and supports ISDN services including the establishment and control of data channels.

- The **Signalling Connection Control Part** (SCCP) [Q.711–Q.716] provides connection-oriented and connectionless services.[*] A significant feature provided by the SCCP layer is **global title translation**[†] [Q.711]. Through global title translation, it is possible for any signalling node to communicate with any other signalling node, even if the address of the destination node is not known by the originating node. For instance, there may be cases when Service Switching Points (SSPs) need to communicate with Service Control Points (SCPs). If the SSP does not know the address of the destination SCP, the Signal Transfer Point (STP) provides the address, through global title translation.
- The **Transaction Capabilities Application Part** (TCAP) [Q.771–Q.775] is designed for non-circuit related messages. TCAP messages are destined for database entities as well as actual end-office switches. TCAP therefore allows end-to-end client–server invocation of a service. The first usage of the TCAP protocol was for freephone number translation (see section 2.1.1).
- The **Intelligent Network Application Part** (INAP) [Q.1208] is the application layer. It provides the information flows between IN elements.

The physical elements of the SS7 network are named Signal Points. As in IP-based networks, the SS7 network uses packet-switching for transferring messages across the network. Signal points can also perform message discrimination and route messages to another signal point.

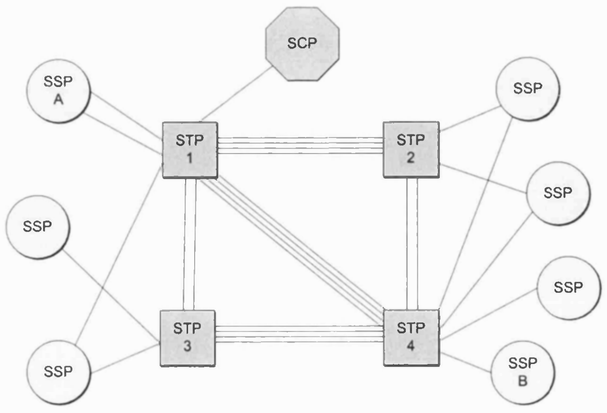


Figure 2.8: *Signalling points within the SS7 network*

---

[*] Connectionless services use parameters to emulate a connection-oriented service.

[†] Arguably, global title translation is similar to the service provided by Domain Name Servers (DNS) and the Address Resolution Protocol (ARP) in the IP domain. A domain name server in one zone initiates a request to a domain name server in a different zone, while global title translation requires a "number translation service" for a specific point-code. A network node makes an ARP request to a logical IP address which is matched to a physical Ethernet (hardware) address, while an STP refers a point-code request to a higher-level STP; and this is similar to a referral in the DNS.

There are three different types of signalling points (as illustrated in figure 2.8). These are the:

- **Service Switching Points** (SSPs) represent the local exchange in the telephone network by converting signalling from the voice switch into SS7 messages.
- **Signal Transfer Points** (STPs) serve as routers and are responsible for directing requests to and from the SSPs.
- **Service Control Points** (SCPs) serve as an interface to databases.

Figure 2.8 illustrates a key functional requirement of the SS7 network, which may be apparent from the links. This is to remain operational at all times and, as far as the physical layer is concerned, it is achieved by using alternative links to some destinations. For instance, for SSP-B to be completely isolated, all links to STP-4 must fail. This redundancy means that all signalling points can be accessed even if some of the links fail.

2.4 The Intelligent Network

This section provides a detailed description of the Intelligent Network architecture. The Intelligent Network architecture is of importance to the overall work presented in this thesis because it shows how control and network intelligence has been achieved in a complex network, such as the PSTN, through the extensive use of state models.

One of the aims of this section is to provide the reader with the technical details and an appreciation of the architectural complexities within it. Arguably, the complexities of the IN architecture have been both its strength and its weakness. In the former case, the rigid, clear and concise definition of the architecture in ITU Recommendations in Series Q.1200 that span thousands of pages have enabled the IN to provide a common architecture (in theory) for the robust control of the PSTN network. However, at the same time a "limiting" or "failing" factor has been the time delay that is inherently present if a body such as the ITU[*] tries to standardise[†] on such a huge and ambitious architecture as the Intelligent Network. This is in stark contrast to

[*] Standards produced by standards organisations such as the ITU-T and ISO are called *de jure* standards. These are stable and easily socialised in the global sense. In the past, *de jure* standards usually took a long time to set. For example, CCITT (ITU-T's predecessor) Recommendations were approved by the Plenary Assembly held once every four years [Asat01]. Furthermore, according to the same reference, "one of the biggest challenges ITU-T faces is to cope with market demands in competitive areas." As a result, the ITU-T is aiming to shorten the approval time for new standards from 4 years (1998) to between 4 weeks to 9 months in 2001 [Asat01].

[†] For an appreciation of the standards process, refer to [Jako01].

the IP domain and the work from the IETF, where the approach is to standardise on protocols and the architecture of the functional entities is left to each proprietary implementation.

According to [Q.1201], the Intelligent Network (IN) is a:

> "telecommunications network **service control architecture** that is a generic platform for open, distributed, service-independent communication. Its goal is to provide an open platform supporting the uniform creation, introduction, control, and management of services beyond the basic telephony services in the telecommunications environment."

This definition of the IN is far more complex than what the IN architecture offered at the time of its conception. Initial versions of the IN infrastructure aimed simply to enable services such as freephone and local-call. This initial aim was desirable, and indeed British Telecom's initial reason for deploying IN infrastructure was to enable services such as Freephone and Callstream Services. Furthermore, BT realised that the IN "simplified service administration and data could now be **centralised** – rather than replicated at various locations in the network, thereby avoiding unnecessary use of valuable computing resources" [OBri89].

The IN has developed from the "single database" application, such as freephone, it offered at the time of its conception. This section describes the present capabilities of the IN, as defined by the ITU Q.12*xx* series of Recommendations [Q.1200].

Today's IN is a service-oriented network architecture that separates service control functions from service switching functions, with typically both types of functions being implemented in different physical equipment.

It is as a direct result of this separation that it is possible to introduce new services rapidly without the need to change the functionality of the switches [Veni00]. Through this separation, the intelligence required for the provision of a service is now placed in dedicated IN servers instead of every switch of the network. The switch functionality is restricted to basic call-processing, to the identification of IN service calls and to the routing of these calls to the IN servers.

What is more, through the adoption of re-usable components, the IN can achieve service independence. Service components (i.e. Service Independent Building Blocks, or SIBs), such as "authentication", "number translation", and "charge" [Q.1201], can be re-used. The principle is to be able to introduce new services by combining basic SIBs where appropriate.

The creation of new services requires a Service Creation Environment (SCE). **Service creation** is defined as "an activity whereby supplementary services are brought into being through the specification phase, development phase and verification phase" [Q.1201]. Essentially this is the process of transforming service descriptions into service logic. The IN does not standardise on the *approach* that must be followed in order to create a new service but rather provides the capabilities and tools for rapid service creation and provisioning. For this reason, service creation and SCEs attracted numerous interests from the research community.

For example, the work of the TINA Open Service Creation Architecture (TOSCA) [TOSCA-IS] aims at incorporating TINA [TINA-GA] principles (see section 2.5) to further the capabilities of the traditional SCE. Further work on SCEs can be found in [Lodg97][Ku94][Gill94][Niits95].

2.4.1 The Intelligent Network Conceptual Model

The Intelligent Network Conceptual Model [Q.1201] provides a set of viewpoints* for looking at the IN architecture. It reduces the complexity of the IN service modelling, analysing the problem from four different points of view, called planes (refer to figure 2.9). Each plane provides an abstraction of a problem that can be studied independently from the other points of view. The four planes derive from a top down analysis of the IN architecture, starting from the service point of view down to the physical point of view [Q.1201]. The four planes address

- Service aspects (the service plane),
- Global functionality (the global functional plane),
- Distributed functionality (the distributed functional plane), and the
- Physical aspects (the physical plane), of an Intelligent Network.

The **service plane** describes the services from the user point of view without any reference to an IN based implementation. Services are described in terms of service features. A **service** is "a stand-alone commercial offering, characterised by one or more core service features, and [it] can be optionally enhanced by other service features" [Q.1201].

A **service feature** is defined as "a specific aspect of a service that can also be used in conjunction with other services and service features as part of [a] commercial offering. It is either a core part of a service or an optional part offered as an enhancement to a service" [Q.1201].

* The viewpoint here is based on the Open Distributed Processing terminology defined by the International Organisation for Standardisation (ISO).

The **global functional plane** (GFP) models the network from a global perspective, hiding the details related to the distribution of functional entities. The GFP expresses a service in terms of **Service Independent Building Blocks** (SIBs). "A SIB is a standard reusable network-wide capability residing in the Global Functional Plane used to create service features" [Q.1203].

In [Q.1201] the **Basic Call Process** (BCP) is defined. The BCP is responsible for providing basic call connectivity between parties in the network. The BCP can be viewed as a specialised SIB which provides basic call capabilities including connecting and disconnecting the call and retaining the Caller Instance Data [Q.1203].

An IN service can be represented as a chain of SIBs connected to the BCP. A Point of Initiation (POI) is the BCP functionality needed to launch a chain of SIBs, while a Point of Return (POR) is the functionality needed to terminate the chain. The POI, POR and the BCP are further discussed in section 2.4.2.

The **distributed functional plane** (DFP) models the distributed view of an IN structured network in terms of computational objects called functional entities (FEs). ITU [Q.1204] defines a **functional entity** as

> "a unique group of functions in a single location and a subset of the total set of functions required to provide a service. One or more functional entities can be located in the same physical entity. Different functional entities contain different functions, and may also contain one or more of the same functions. In addition, one functional entity cannot be split between two physical entities; the functional entity is mapped entirely within a single physical entity. Finally, duplicate instances of a functional entity can be mapped to different physical entities, though not the same physical entity."

The FEs may perform atomic functional entity actions (FEAs) and, as a result of the FEAs, exchange messages called information flows (IFs).

The **physical plane** models the physical aspects of an IN structured network, including the detailed design of physical network elements, Physical Entities (PEs) (i.e. switches, general-purpose computers that contain databases, etc.). The FEs of the DFP are assigned to PEs and the IFs between the communicating FEs in different PEs are mapped into the protocol messages.

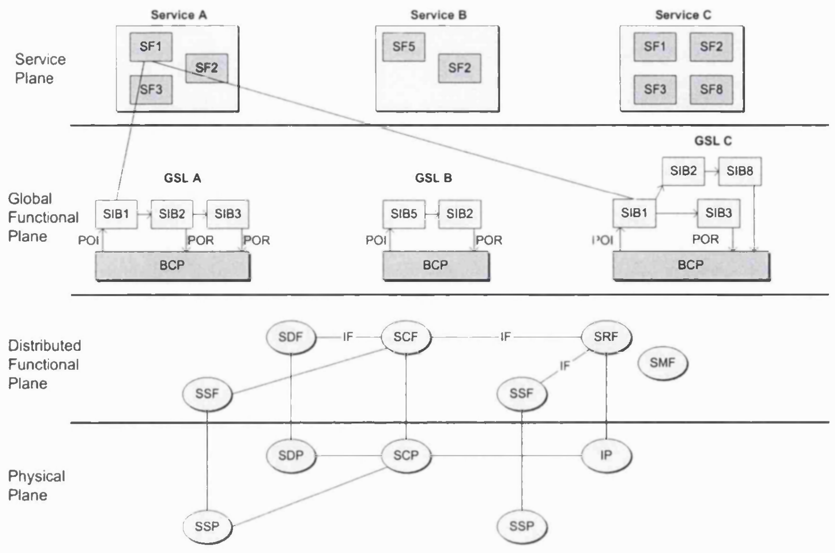


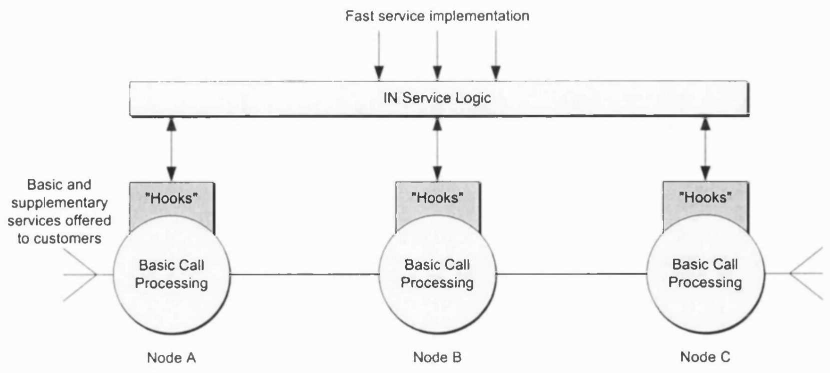
Figure 2.9: The four planes of the IN conceptual model

Figure 2.9 depicts the result of the IN service decomposition through the four IN conceptual model planes. Examples of FEs in figure 2.9 are the SSF, SCF and SDF; examples of PEs are the SSP, SCP, and SDP.

2.4.2 The IN Service Processing Model and the Basic Call Process

In a non-IN world, the switching nodes deliver service to the parties involved in a call. Services are programmed within the switching nodes (SSP) and may be basic or supplementary. Essentially, the service logic is contained within the switching nodes.

In the IN world, the service logic for the supplementary services may be developed and executed outside the switching nodes. Figure 2.10 illustrates the IN service processing model, with the switching nodes represented by circles.



Fast service implementation
IN Service Logic
Basic and supplementary services offered to customers
"Hooks"
Basic Call Processing
Node A
Node B
Node C

Figure 2.10: The Intelligent Network service processing model [Q.1201]

Supplementary services, when requested, trigger the execution of the IN service logic by means of software "hooks". One of the aims of IN is to standardise these "hooks" and the messages between them.

The **Basic Call Process** (BCP) [Q.1203] represents the well-defined call control process. It is related to the basic call state model in that the trigger points of the BCP correspond to the entry points, Points Of Initiation (POI), into the Global Service Logic, that is an entry point into an IN service.

The **Global Service Logic** (GSL) [Q.1203] can be characterised as the "glue" that defines the order in which SIBs will be chained to accomplish services. Each instance of GSL (figure 2.11) is (potentially) unique to each individual call, but uses common elements:

- BCP interaction points (POI and POR).
- SIBs.
- Logical connections between SIBs and between SIBs and BCP interaction points.
- Input and output data parameters, service support data and call instance data defined for each SIB.

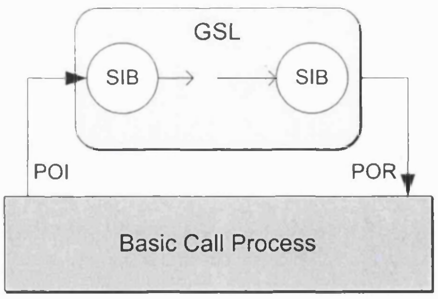


Figure 2.11: *The BCP in the global functional plane*

Since the BCP is based on a state-dependent process, IN services can only be created within the boundaries set by the BCP. There are many consequences for the creation of services within these boundaries in terms of service control. For example, at one extreme, invoking the service logic only when the call has been completed does not leave the service logic much to do. At the other extreme, the ability of a service logic programmer to activate a trigger anywhere in the BCP may easily lead the system to a chaotic state.

2.4.3 IN Capability Sets

Capability Sets (CS) [Q.1202] are the phases of IN evolution that give rise to particular service capabilities. Each IN CS is defined by the hardware and software elements described in the standards. Each new IN CS adds functionality and thereby increases the scope for services. At the top level, each CS introduces benchmark services. A benchmark service is not standardised but is "a stand-alone commercial offering, characterised by one or more core service features, and it can be optionally enhanced by other service features" [Q.1202].

2.4.3.1 IN Capability Set 1 (CS-1)

IN CS-1 [Q.1211] capabilities are intended to support services and service features that fall into the category of "single ended" or "single point of control" services referred to as Type-A, while all other services are placed in a category called Type-B. Single-ended is defined as [Q.1211]:

> "A single-ended service feature applies to one and only one party in a call and is orthogonal (independent) of both the service and topology levels to any other parties that may be participating in the call. Orthogonality allows another instance of the same or a different single-ended service feature to apply to another party in the same call as long as the service feature instances do not have feature interaction problems with each other."

Single point of control is defined as "a control relationship where the same aspects of a call are influenced by one and only one service control function at any point in time" [Q.1211]. The **single point of control** characteristic of IN CS-1 services restricted a service process to only one call party (single-ended). This was a limiting factor and has changed with the introduction of IN CS-2 which supports both Type-A and Type-B services (see section 2.4.3.2).

The IN CS-1 benchmark services identified in [Q.1211] are:

Abbreviated Dialling	(ABD)	Security Screening	(SEC)
Account Card Calling	(ACC)	Selective Call Forward on	
Call Distribution	(CD)	Busy/Don't Answer	(SCF)
Call Forwarding	(CF)	Split Charging	(SPL)
Call Rerouting Distribution	(CRD)	Tele-voting	(VOT)
Credit Card Calling	(CCC)	Terminating Call Screening	(TCS)
Destination Call Routing	(DCR)	Universal Access Number	(UAN)
Follow-me Diversion	(FMD)	Universal Personal	
Malicious Call Identification	(MCI)	Telecommunications	(UPT)
Mass Calling	(MAS)	User-defined Routing	(UDR)
Originating Call Screening	(OCS)	Virtual Private Network	(VPN)
Premium Rate	(PRM)		

Figure 2.12 depicts the DFP for IN CS-1. The functions that have been defined as a first subset of a target IN architecture are related to traditional call handling (service switching and triggering), service execution (service control) and service management.

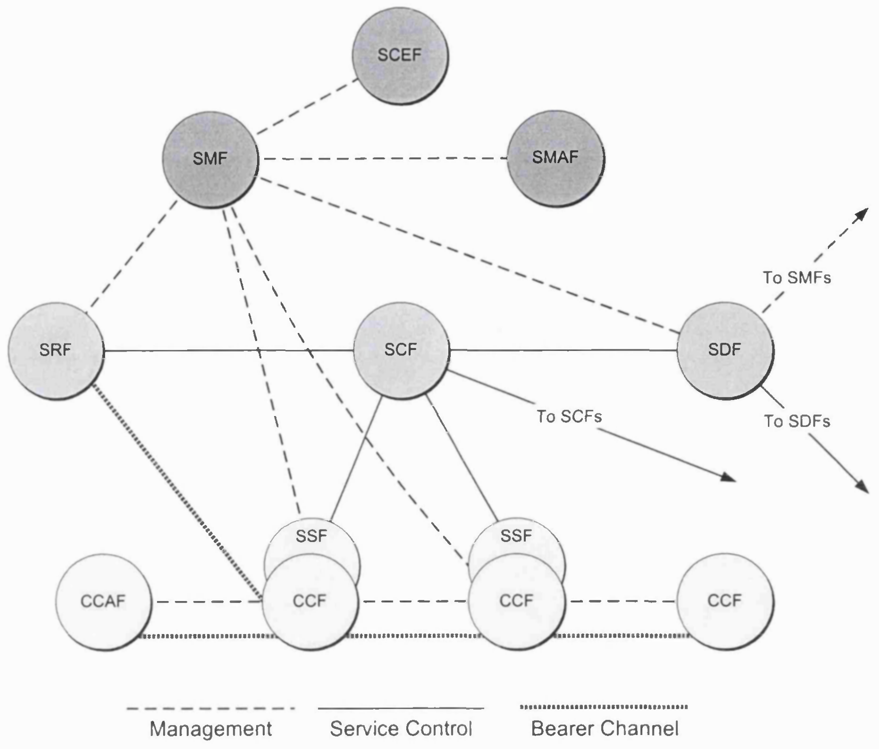


Figure 2.12: *Distributed functional plane for IN CS-1 [Q.1211]*

The IN CS-1 DFP functions can be grouped into the following categories [Q.1211]:

- **Basic call-handling functions**

  The **Call Control Agent Function** (CCAF) [Q.1214] represents the user terminal function and hence provides access to the network. The CCAF accesses a call control function (CCF) that provides basic call-processing functionality and can thus be considered as a traditional "switch". Central to the operation of the CCF is the basic call model that is discussed in section 2.4.4.

- **Service execution functions**

  These functions provide supplementary services. A **Service Switching Function** (SSF) [Q.1214] represents additional functionality for controlling switch resources and provides a well-defined, service-independent interface to the service control function (SCF) that controls resources in a switch or peripherals, based on an appropriate service logic program. A **Service Data Function** (SDF) contains the service data and provides standardised real-time access for SCFs to service data. Finally, the **Specialised Resource Function** (SRF) is used for controlling resources such as speech synthesisers and voice recognition systems.

- **Service management functions**
  The **Service Management Function** (SMF) [Q.1214] supports service introduction and maintenance; it is accessed by a **Service Management Agent Function** (SMAF) that provides the man-machine interface to the SMF. Also, an additional **Service Creation Environment Function** (SCEF) allows for the specification, testing and introduction of services in the IN.

As discussed in section 2.4.1, IN Services, i.e. service features, are composed of SIBs in the GFP. The monolithic view of a SIB in the GFP has to be decomposed in the DFP into an interacting set of capabilities. Each functional entity in the DFP may perform specific operations, referred to as **Functional Entity Actions** (FEAs). Thus each SIB is decomposed in the DFP into a set of "client–server" relationships between one or more functional entities, with the client being the SCF and the server being one of the other FEs, such as the SDF, SRF, or SSF [Q.1211].

Consequently, different functional entities in the DFP must exchange messages to perform a desired SIB functionality. These client–server information exchanges between the functional entities are called Information Flows (IFs). The total set of IFs between any two functional entities in the DFP will be a number of such client–server information flows. Some examples of functional IFs are the following:

- Initial detection point – the SSF starts a dialogue with the SCF and requests further instructions (based on the occurrence of a specific trigger event).
- Play announcement – the SCF instructs the SRF to send an announcement to a user.
- Prompt and collect user information – the SCF instructs the SRF to collect some dialled information from the user.

2.4.3.2 IN Capability Set 2 (CS-2)

IN CS-2 is defined in the ITU Recommendation of series Q.122*x*. IN CS-2 consolidates and enhances the features of IN CS-1 by the addition of functional entities and information flows and by specifying a number of additional services and service features that include mobility services and B-ISDN services [Feyn97]. More importantly however, IN CS-2 includes services that allow the end user to control the services to which they are subscribed [Feyn97].

IN CS-2 also acknowledges a situation that may be desirable or undesirable. This is called **feature interaction** and is defined as the interaction of features that could be desirable or not [Q.1221]. A desirable interaction is defined as **feature cooperation**, while an undesirable one is termed **feature interference** [Lin98]. However, feature interaction has come to be synonymous with feature interference. The issue of feature interactions has attracted considerable research activities [Peng98][Ever97][Cape96][Naka95][Kell94][Brot93].

The increased capabilities of IN CS-2 are reflected by comparing the DFP for IN CS-1 (see figure 2.12) with that of IN CS-2 (shown in figure 2.13).

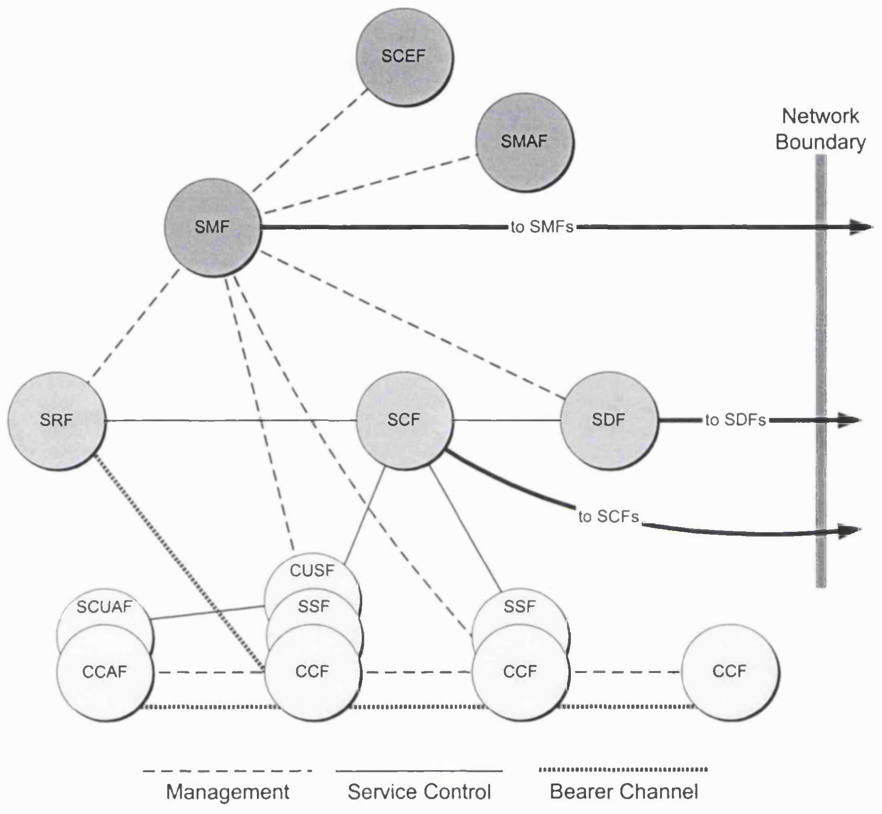


Figure 2.13: *Distributed functional plane for IN CS-2 [Q.1221]*

IN CS-2 defines two additional functional entities: the **Service Control User Agent Function** (SCUAF) and the **Call Un-Related Service Function** (CUSF). The SCUAF provides the user access to the network CUSF [Q.1221]. The CUSF allows the specific triggering of services *outside the basic call process*, and only supports connection-oriented communication between a user and the network. This is a significant capability as it can be used for billing applications on different networks, such as IP.

The IN CS-2 also defines the **Basic Call Unrelated Process** (BCUP), which is the counterpart of the BCP for modelling the capabilities implemented through actions that are not performed on behalf of a particular call, even though they are necessary for supporting calls. The BCUP is defined as "a specialized SIB which provides the call unrelated capabilities. These capabilities enable the use of GSL as well as other SIBs to completely describe IN CS-2 services and service features" [Q.1223].

Figure 2.14 shows a possible mapping of the IN CS-2 DFP onto the physical plane. The figure is included for completeness to provide an appreciation of the number of protocols and functional entities that are involved in a simplified IN implementation.

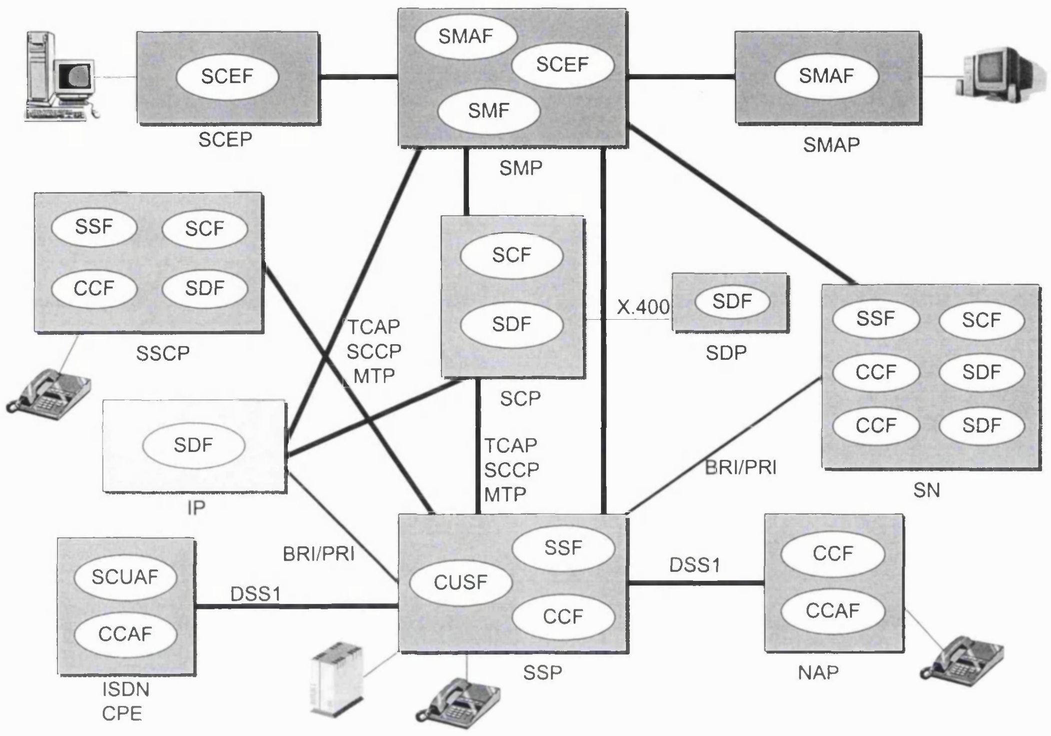


Figure 2.14: *Example of a possible mapping of IN functional entities into IN physical entities supported by IN CS-2 [Veni98]*

The protocols that are depicted in figure 2.14 include parts of the SS7 core signalling protocol. Examples of these are TCAP, SCCP and MTP. Also shown are access signalling protocols such as DSS1 [Q.931].

2.4.3.3 Inter-working Functions in Future Capability Sets

A particular area of research and development in IN CS-2 is in the enhanced telecommunications **Inter-working functions.** Inter-working functions enable service features to be made available outside the home network and reflect the converging telecommunications environment. The interest shown in inter-working functions as part of the convergence process is supported by a number of publications [Scho98][Zhen98][Deci97][ETSI EG 201-722]. The IN must be in a position to provide inter-working capabilities. Inter-working is the process by which several networks (potentially of different types such as IN-structured and non-IN-structured, public and private) work together to provide a service.

The DFP sees the introduction of standardised interfaces for the inter-working of inter-domain SCFs and SDFs. Additionally internetwork management interactions and distributed data handling processes are supported.

Specifically, IN CS-2 allows SDF–SDF, SDF–SCF and SCF–SCF communication across network boundaries. For example, the SDF–SDF inter-working interface can serve two purposes: it provides a mechanism to copy data between networks and it provides transparent data access. Similarly, the SCF–SCF interface (across network boundaries) allows two service logics to communicate; for example, a network can handle a call without having full knowledge of the service logic, as long as it can find another network that can help.

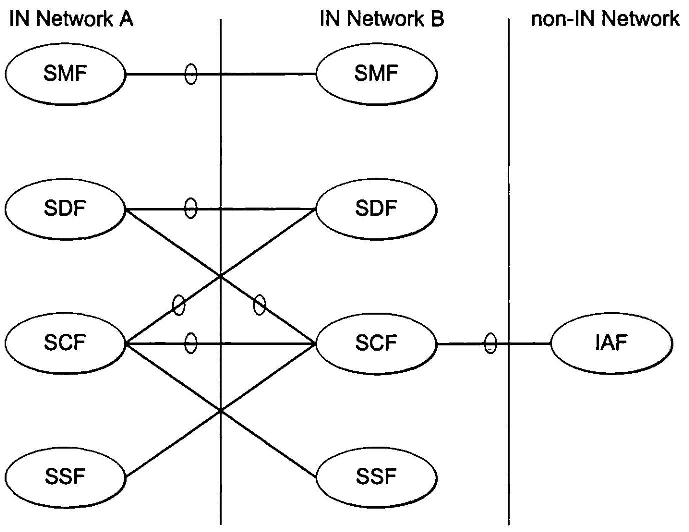


Figure 2.15: IN and inter-working functional relationships

Figure 2.15 depicts the possible IN network inter-working functional relationships. Although the functional relationship SCF–SSF is outside the scope of IN CS-2, the SCF–SCF relation is new. However, it is not possible for the SCF to interact with a remote SCF that is directly interacting with the call. This is a limitation, or the absence of a requirement, for multiple point-of-control capabilities in IN CS-2. The top interface defined in figure 2.15 is the SMF–SMF interface. This protocol follows the Telecommunications Management Network (TMN) generic protocols and the TMN X-interface [M.3320].

A different inter-working capability is emerging. This is the need to inter-work with non-IN structured networks. Such a form of inter-working requires the introduction of additional functional entities, such as the **Intelligent Access Function** (IAF) [Q.1224]. The IAF is responsible for providing access to and from the SCF of the IN-structured network, and for mapping the information between the internal and external representations.

Comparing the DFP of IN CS-1 and the DFP of IN CS-2, one notices that the IN has been incrementally adding functionality to reflect the changing telecommunications environment conditions. Of course, this is a natural step in any developing system. As a result, the role that IN is asked to play in the future may be far from the initial aims and objectives of Recommendation Q.1201. The area of inter-working and the new role of the intelligent network within the wider scope of network intelligence is examined in chapter 3.

2.4.4 The Basic Call Model

The primary goal of IN services is to enhance the basic call process, thereby creating new services such as premium rate. To achieve this, IN services have to control network resources in a flexible and efficient way through a standard resource control interface, the SSF–SCF.

This approach requires the switches, i.e. the connection control function/service switching function, to be capable of providing visibility and control on detailed call events. Hence, a corresponding model is required within the DFP that identifies all possible points in basic call-processing, as seen in a switch, from which IN services can be invoked, i.e. when interactions between the SSF and SCF can take place.

This model is called the Basic Call Model (BCM). The BCM represents a standardised view of call-processing functions to external service logic and provides the framework for IFs between the SSF and SCF. This means that IN service logic located in the SCF "sees" a call only by means of the information it receives from the SSF, i.e. the received IFs, based on the states identified in the BCM (figure 2.16).

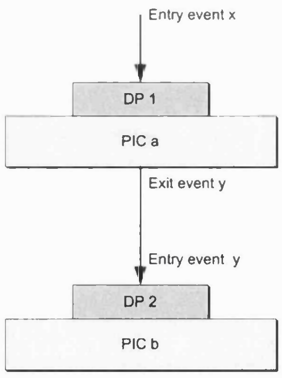


Figure 2.16: *The BCM, detection points and points in call*

The **Basic Call State Model** (BCSM) identifies the logical points in basic call-processing where the IN service logic located in the SCF is permitted to interact with basic call control capabilities provided by the switch [Q.1204]. The states that need to be visible to the IN service logic are identified by the BCSM.

The BCSM is built from three fundamental components, discussed below:

- **Points in Call** (PIC) provide an external view of a call-processing state or event to IN service logic. PICs are vendor independent, providing a standardised view of call-processing behaviour. A PIC is characterised by means of entry events, exit events, actions performed within the PIC, and information available at the end of the PIC.
- **Detection Points** (DPs) are placed between the PICs. A DP (also referred to as a trigger check point) is associated with a particular PIC. DPs indicate states in the basic call/connection processing where the control can be transferred to the IN service logic with or without the CCF suspending processing. If the CCF processing is suspended, the DP is called a *Request DP*, otherwise it is a *Notification DP*. A DP can be *armed*, i.e. the monitoring of a CCF processing state can be requested, statically on a configuration base (in which case the DP is referred to as a *Trigger DP*) or dynamically on IN service logic request base (in which case the DP is referred to as an *Event DP*).
- **Transitions** indicate the normal flow of basic call/connection processing from one PIC to another.

Any IN service involves two separate sets of basic call-processing logic in the switching network [Feyn97]. These are closely related, as illustrated in figure 2.17.

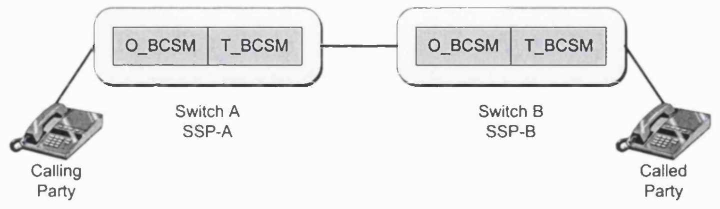


Figure 2.17: *Separation of the BCSM into an O/T_BCSM*

An **originating call model** supports the call's originating side (i.e. the calling party). This is modelled by the originating basic call state model (O_BCSM). On the terminating side, there is a **terminating call model** for the call's terminating basic call state model (T_BCSM). Both sides of the call state model are active within an IN SSF; even an intra-switch call requires both call state models.

2.4.4.1 The Basic Call State Model for CS-1

The O_BCSM (figure 2.18) specifies six PICs and ten DPs. One non-IN event, *Route_Busy*, is considered. The event is caused by either a corresponding indication from the T_BCSM (if this is a local switch) or a reception of the call "rejected" message (indicating the selected route is busy) from another switch. Depending on the switch application, this event may result in either a transition to PIC 2 (in order to select another route) or exception processing.

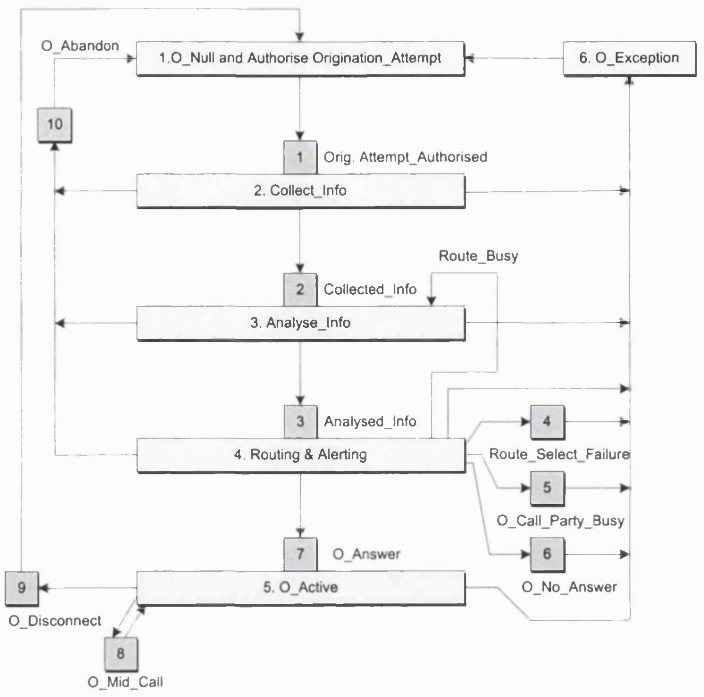


Figure 2.18: *The O_BCSM for IN CS-1 [Q.1214]*

The T_BCSM (figure 2.19) specifies five PICs and seven DPs. Note that a transition from PIC 8 to PIC 9 has no DP associated with it. The event that causes this transition (i.e., the start of the alerting process) is the only strictly non-IN event visible to T_BCSM.

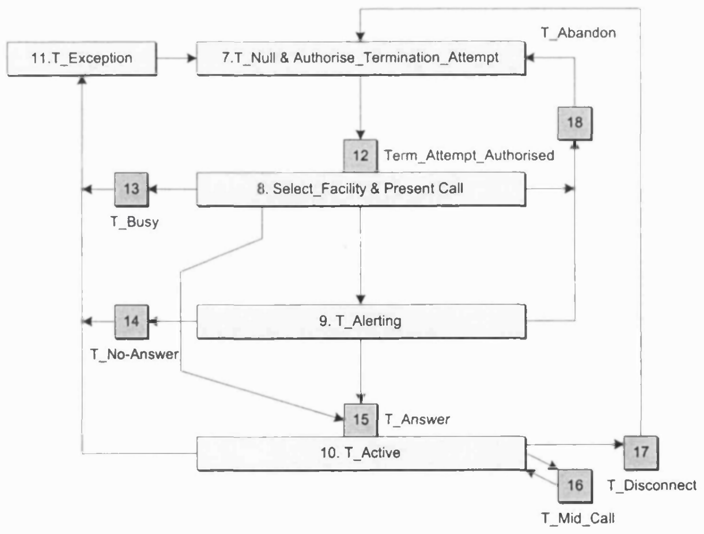


Figure 2.19: *The T_BCSM for IN CS-1 [Q.1214]*

2.4.4.2 The Basic Call State Model for CS-2

The BCSM for IN CS-2 (figure 2.20) is visibly richer than its IN CS-1 counterpart. This section points out the main differences between the two models. A complete description can be found in the ITU Recommendations Q.122*x* and Q.121*x*.

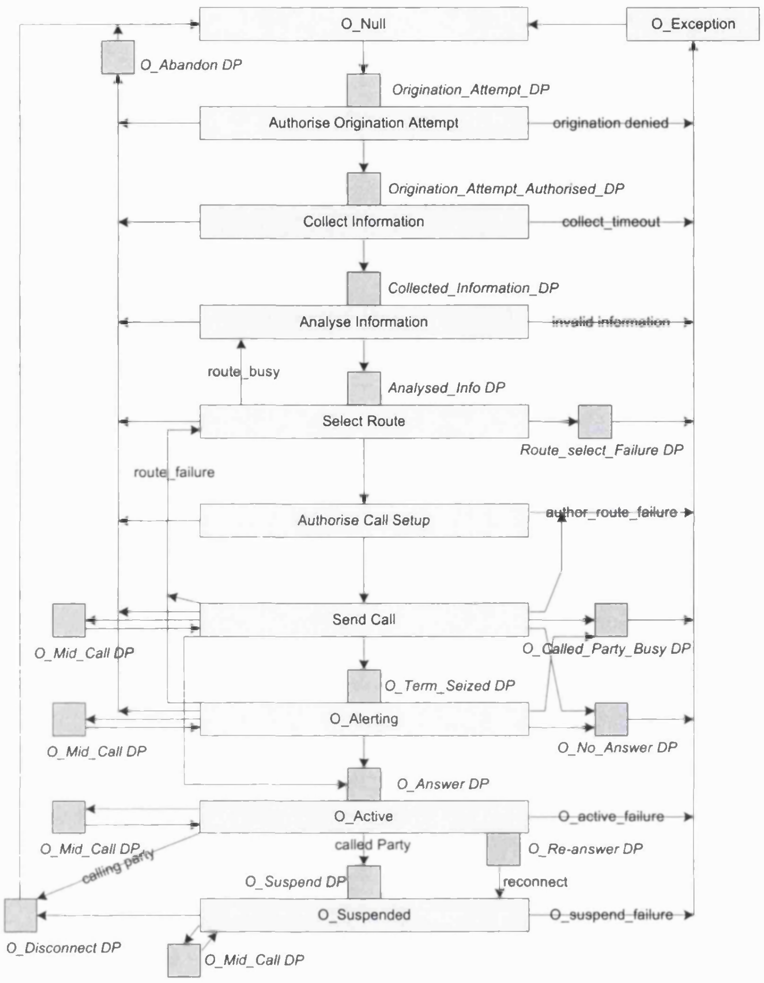


Figure 2.20: *The O_BCSM for IN CS-2 [Q.1224]*

Non-functional modifications include the removal of numbers from DPs in IN CS-2. More substantial modifications are manifested in the new functions of the PICs and DPs, and their applications to services.

For example, the *O_Disconnect* and *T_Disconnect* PICs have been replaced in the originating and terminating models by the *O_Suspended* and *T_Suspended* PICs. The justification for this change is dictated by a special way in which many services are set up; unless the called party explicitly wishes to disconnect the call, it should not be immediately disconnected. This allows for the called party to hang up momentarily and pick up the phone again (**reanswering**). In order to reanswer the call, it cannot be disconnected even though the called party hangs up. The state of the call is captured in the model as **suspended** and it is treated differently. While the call is suspended, both the calling and called parties are placed in the *O_Suspended* and *T_Suspended* PICs. On the terminating side, the physical resources associated with the call remain connected and the appropriate timer is started. If the terminating side reanswers before the timer has run out, T_BCSM sends the *reanswer* indication to the O_BCSM and the call becomes active again, otherwise the T_BCSM sends the *disconnect* notification.

2.4.5 The IN Switching State Model (IN-SSM)

The IN-SSM is defined in [Q.1204] as a means of providing the "finite state machine description of SSF/CCF IN call/connection processing in terms of call/connection states". The IN-SSM is a class of objects that corresponds to the SCF view of call and connection processing within the SSF/CCF.

The call segments defined in [Q.1204] are expanded to include the new types of objects called Legs and Connection Points as well as the BCSM objects (i.e. O_BCSM or T_BCSM) associated with them.

- A **leg** represents a connection with an "addressable entity", that is, an end user, the SRF, or another SSF/CCF [Q.1204]. Any leg may be either active or passive. In IN CS-1, only a leg that represents an access interface may be active.
- A **connection point** is the object that associates two legs so that the information entering the SSF/CCF through one leg is carried out (via that object) on the other leg and vice versa [Q.1204].

According to [Fayn97] in IN CS-1, the role of the IN-SM has not been emphasised and, consequently, neither has that of the IN-SSM, though the technical case for their future exploitation was given. IN CS-2 defines a more powerful IN-SSM that focuses on connection control issues. It therefore contains objects that are abstractions of switching and transmission resources.

The CS-2 IN-SSM uses the **Call Configuration** (CC) model as a tool to represent the CCF activities. The CC is a model that categorises the status of one or more network connections. It is based on the **Connection View** (CV), in the sense that CV objects provide the SCF with a generic view of call-processing and each Call Configuration models a CV state in an SSF.

The purpose of defining call configurations is to produce a set of examples used to describe IN call manipulation services. Figure 2.21 illustrates the notation and the objects used in describing the call configurations.

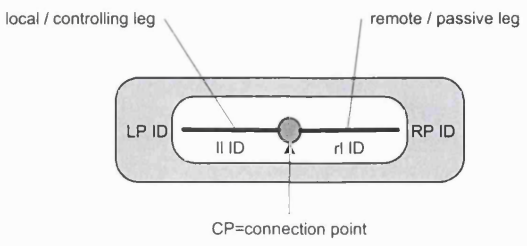


***Figure 2.21:** Representation used in a call configuration*

Attached to a connection point (CP) are one or more legs, with the leg to the left being the controlling (or local) leg. The legs to the right are defined as passive (or remote) legs, of which there can be one or more. Associated with each of the passive legs is a BCSM.

Figure 2.22 depicts the **Call Association Object** (CAO) that is used for the representation of multiple calls associated with a particular user as perceived by the serving node associated with that user. The CAO allows the graphical representation of the condition when a user is engaged in more than one call.

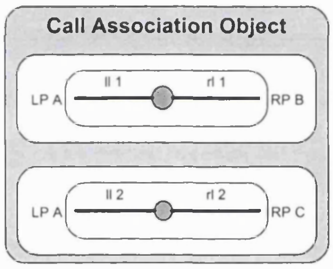


***Figure 2.22:** Representation of a call association object*

This approach has been developed to provide IN multiparty handling capability, and it can be used as a tool for associating different calls.

2.4.6 IN CS-2 Call Party Handling

IN CS-1 was limited to Type-A services. This meant that it could only manipulate specific aspects of a single leg of a call. As a result, for instance, capabilities to alter the Calling or Called number were provided, but it was not possible to alter how one leg of a call related to another leg of the same call.

The introduction of **Call Party Handling** (CPH) [Q.1224] service features in IN CS-2 is arguably the most important addition [Hink98][ORei98][Haze98]. CPH allows multiple bearer connections to be split, merged and manipulated more flexibly during the course of a telephone conversation. Therefore, CPH provides the capability to manage and control individual parties in a call involving two or more parties. The current call topology is maintained by the SCF, while atomic commands from the SSF can alter this topology. Typically, instructions from the SSF add, delete, join or separate bearer channels from other parties in the call.

At the highest level, the SSF consists of a **Call Segment Association** (CSA). A CSA represents half a call. The CSAs communicate with each other using the SSF–SCF INAP operations defined in IN CS-2 [Q.1228]. Communication events among the CSAs consist of inter-BCSM events (e.g. *Setup*, *Answer*).

A CSA consists of one or more **Call Segments**. The call segments communicate with either the O_BCSM or the T_BCSM, and the SSF Finite State Model (FSM).

The Call Segment can be in four possible states [Q.1228]:

- **Joined** – indicating that a path is joined to the connection point which enables the user to communicate with other users in the call segment
- **Pending** – indicating that a path is in the process of being set up
- **Surrogate** – indicating that a leg supports a communication path towards a virtual party in the network, rather than towards an external end party
- **Shared** – indicating that a controlling leg is absent from a call segment and is present in the associated call segment.

The CPH finite state model (see figure 2.23) is examined here using an example of the transitions involved in setting up a two-party call.

A call begins from the "Null" call segment that indicates that no Call Segment Connection View (CSCV) instance exists and one can be created. The state represents the condition where no call-processing is active and there is no controlling or passive leg connected to the connection point. When the SSF detects a *SetupInd* signal, the CSCV is represented by the "Originating Setup" CSCV. The *SetupReqInd* finally causes the CSCV to move to the "Stable 2-Party" CSCV. The "Stable 2-Party" CS represents a stable two-party call, and is either an originating or a terminating call from the perspective of the controlling user with a "joined" controlling leg and a "joined" passive leg with either an O_BCSM or T_BCSM associated with it.

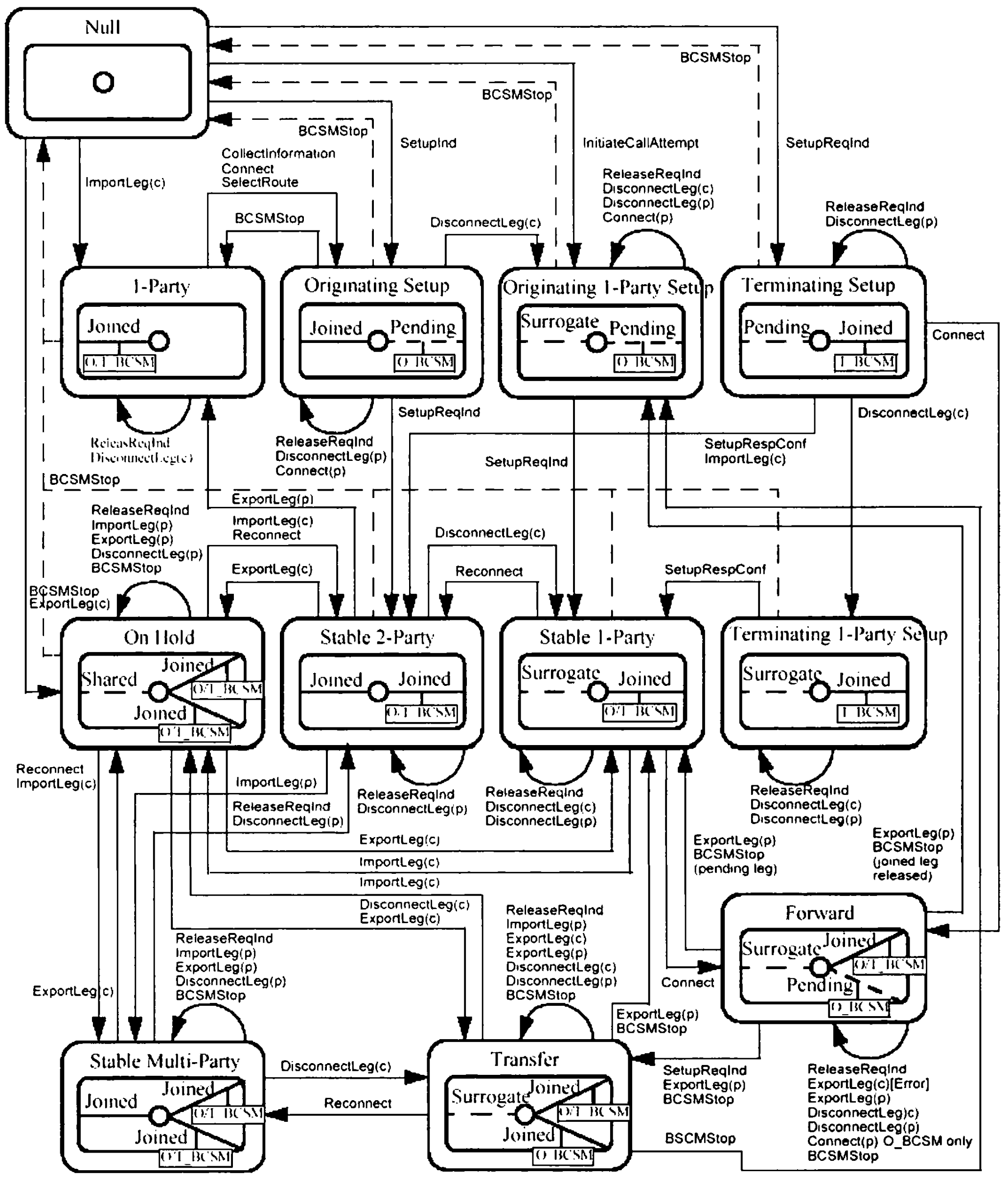


Figure 2.23: Finite state model for CPH of IN CS-2 [Kumm98]

2.4.7 IN Summary

In this section a summary of the IN architecture is presented. The discussion on the IN control architecture presented and examined issues such as:

- The advantages of the IN architecture are the result of the separation of basic call-processing from additional service logic. This separation also provides the ability to introduce new services rapidly.
- The BCM provides a standardised representation of call-processing functions to external service logic. As a result the only view provided to an external IN service is based on the states identified by the BCP.

- Through DPs, control can be transferred to the IN service logic with or without the CCF suspension.
- IN CS-2 provides a much richer and more complete set of service features than IN CS-1, in particular the introduction of Type-B services. Type-B services in IN CS-2 were examined by analysing the Call Party Handling service feature.
- The IN architecture relies on distributed state models. These form an integral part both of the signalling protocol (SS7) and of the IN architecture itself.
- Traditional network intelligence was focused on providing advanced services by enhancing the basic call state model.

2.5 THE IN, THE TINA REFERENCE ARCHITECTURE AND NEW TECHNOLOGIES

The Telecommunications Information Networking Architecture (TINA) defines a framework for the development of service and network management applications [TINA-GA][TINA-GU]. TINA defines four architectures:

- The **computing architecture** [TINA-CA] defines a set of concepts and principles for designing and building distributed software and the software support environment, based on object-oriented principles.
- The **service architecture** [TINA-SA] defines a set of concepts and principles for design, specification, implementation and management of telecommunication services. It identifies three main concepts: session, management and access.
- The **network architecture** [TINA-NA] defines a set of concepts and principles for design, specification, implementation and management of transport networks.
- The **management architecture** [TINA-MA] deals with software systems that are used to manage services, resources, software and underlying technology.

The TINA approach foresees that service components are deployed on a **Distributed Processing Environment** (DPE), which may in turn be implemented on top of different network infrastructures [TINA-DPE]. The DPE is used for both the transmission of control information and the multimedia streams that may flow between user applications. Therefore, the network has to satisfy the connectivity requirements of different types of services, such as multimedia, multiparty, or multicasting services.

From the computational point of view, TINA considers that the service and network management and control facilities are deployed on a distributed, object-oriented processing environment, such as the Common Object Request Broker Architecture (CORBA) [CORBA95].

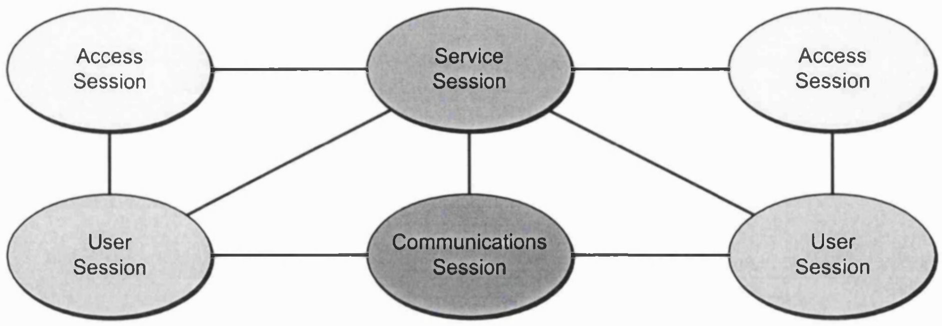


Figure 2.24: *The TINA service architecture [TINA-SA]*

The TINA Service Architecture (figure 2.24) identifies four major types of sessions as follows:

1. The **Access Session** models the user actions for accessing a service
2. The **Service Session** represents the control and management actions needed for the support of a specific service from the system point of view
3. The **Communication Session** encompasses the set of control and management procedures from the network's resource point of view in order to support the needs of services making active use of the underlying network infrastructure
4. The **User Session** reflects the activities performed and the resources allocated by one user for one specific service session.

The TINA architecture employs an object-oriented view [TINA-GA]; as such, it is fundamentally different to the function-oriented IN architecture. For instance, rather than defining functional network elements for the distributed implementation of SIBs, as is done in the IN architecture, the DPE supports the arbitrary distribution of TINA service components, i.e. the computational objects. Hence, there is no need for specific network nodes with dedicated functionality within TINA. Furthermore, the information flows between IN functional entities (via the signalling network) are replaced by TINA computational object interactions, through operational interfaces (supported by the DPE).

While most of the efforts surrounding TINA pertain to how an existing architecture could migrate towards TINA, such as [P508], it has been argued [Solo00a] that the TINA architecture can be used as a reference model for viewing new technologies. This role is examined next.

It has been argued that the communications session of the TINA service architecture reflects the traditional role of the telecommunications network operators [Solo00a]. The service session distinguishes the growing role of the provider of application services as a separate entity apart from the communication session. The access session co-ordinates the access of users towards a set of services, by providing services such as password authentication, or terminal capability negotiation.

The author takes the view that the TINA service architecture is a useful and desirable way of understanding the roles of the players in the future communication environment. Of course, this does not imply that the underlying technology is that specified by TINA [Solo00a], for instance that one has to develop a Distributed Processing Environment (DPE) approach.

The IN CS-2 increases the importance of the SCF over IN CS-1 for establishing connections. One of the most significant developments in CS-2 is that of call party handling where fine-grained manipulations of multiple-associated call paths is possible. Services such as multicasting or multimedia rely on these services. In future capability sets, such as IN CS-3/4 there will be enhanced support for connection control. The SCP will be able to establish a communication session independently without the intervention of the end user. This is a very significant development because it will allow greater control of the TINA Service Session through the SCP [Solo00a].

Mobile networks and specifically Global System Mobile (GSM), provide a set of IN services without the explicit use of an IN architecture. Services such as mobility are much sought after in the PSTN but are seen as basic services in the GSM. However, to achieve service differentiation in GSM and support customers while they are roaming in other operator's networks, Customised Applications for Mobile Enhanced Logic (CAMEL) [ETSI TS 101-285], was developed. The CAMEL architecture is outlined in figure 2.25.

In CAMEL, the Home Location Register (HLR) bears similarities to the SCP but does not allow customised service logic that interacts with the basic call process. CAMEL is necessary to allow operator specific services. A CAMEL Service Environment (CSE) allows the customisation of services and is equivalent to an SCP in the PTSN.

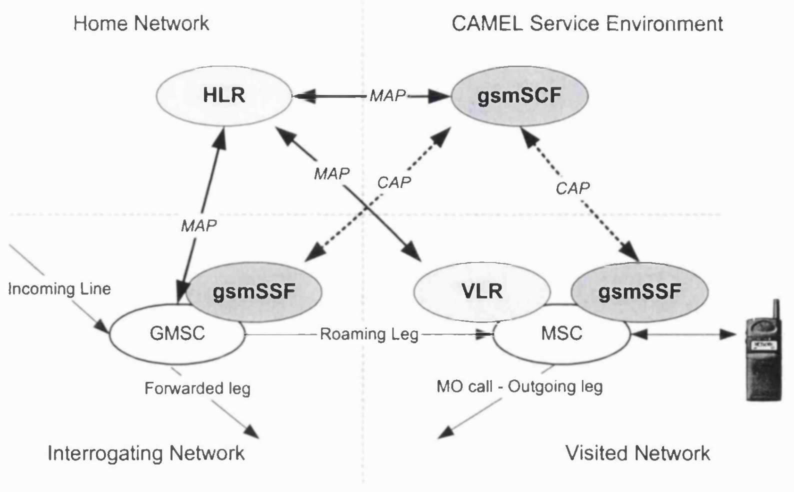


Figure 2.25: *The CAMEL architecture*

Section 3.6 follows the discussion of this role by mapping IP-based technologies onto the TINA Service Architecture and putting forward the conclusions of this part of the work.

2.6 State Machines in Telecommunications Networks

The extent to which the state machine model is used to describe the behaviour of network intelligent architectures has been identified. The state models explicitly describe in a standardised manner the way in which the functional entities interact. This section identifies the importance of the state models within the control plane of the PSTN network.

The PSTN network is based on the principle of providing controlled access to a limited bearer channel resource which, arguably, represents the main source of revenue for a network operator.[*] Therefore it is crucial for the network operator to use the bearer connections in a way that allows the greatest return on investment. After all, without an operational bearer connection[†] no utilisation of the network can take place, leading to loss of revenue. The importance of the bearer connection is reflected by the fact that the network is performance engineered, with design focused on limiting down-time and ensuring that network load does not

[*] This is within the context of fixed networks, i.e. PSTN. In the GSM case, text messaging has proved to be an additional significant source of revenue that does not require the use of a bearer channel.

[†] Recent examples of network failure include the brownout of the AT&T American network [McDo92][Hatt97] with a financial loss amounting to 1 billion dollars, the 1998 integrity breach in the AT&T Chicago frame relay network [Meht98], and the latest example is the brownout in the BT network in February 2000, which blocked millions of calls [BBC00].

hinder performance. To achieve this, it is vital that a resilient control-plane architecture be in place.

Furthermore, it was put forward that the PSTN network is a huge distributed network with components from different vendors. Each of these components may potentially have different capabilities but nevertheless it is essential that they interact to form the network. It is critical that a common underlying capability enables the interconnection* of these vendor-specific components and, again, this is provided by the control-plane architecture.

A control-plane architecture can be characterised by two key features: firstly, it provides the robust foundations for the communication of the components and, secondly, it provides a resilient environment for the execution of advanced services. **The common controlling element in both of these features is the state machine**. While the communication part is handled by the signalling network, through the SS7 protocol, the resilient environment is provided by a network intelligence architecture, such as the standardised architecture of the Intelligent Network.

Having provided a control-plane architecture, there is now a need to allow communication with the end users based on their activities. These activities are unpredictable but, nevertheless, it is the responsibility of the network to implement measures that capture this changeable behaviour of the users in a formal manner. This must be done in a way that does not severely limit the user experience. By definition, an unpredictable behaviour cannot be captured in a model and, as a result, there needs to be a trade-off between the capabilities available to end users against the complexities involved in capturing parts of this unpredictable behaviour. The IN CS-1 has provided sufficient benchmark services that enable a wide range of service capabilities; through the IN CS-2 these services have been enhanced and widened to include capabilities such as multi-party calls.

The state-machine approach in the PSTN has proved to be a technique that is capable of providing resilience for the IN architecture and also robustness for the signalling layer.

In the case of the IN architecture, the state machine is the fundamental controlling component from which services can be triggered, and in the IN architecture the state machine is manifested in the form of the O/T_BCSMs.

* The regulatory framework relating to inter-connection and inter-operability is addressed in [Oftel99].

Although there are significant differences between various IN implementations a number of key features apply to each of them. These key features enable the network-of-networks to be controlled through similar actions, and hence provide a common denominator in the convergence of these architectures. The presence of fully defined state machines on all the interacting FEs and the BCM provide for the resilience needed in the PSTN environment.

Controlling the bearer connections is achieved in the PSTN through the

- extensive use of state machines
- ubiquitous description of the BCM
- complete encapsulation of the call process in the BCSM
- robust nature of the signalling network and its protocols.

The IN mandates the use of FSMs for the description of protocol-related behaviour of the FEs.

All possible actions by the FEs are rigorously tested before any attempt is made to deploy a new service. The specification of the FEs is done using the Specification and Description Language (SDL) [Z.100], which is a standardised language for describing systems that are reactive, concurrent, real-time, distributed and heterogeneous* [Haug95].

If a scenario arises that may result in an invalid transition in the BCP, the IN is able to achieve graceful degradation of service, which ensures that the network remains operational.

The features provided by timers aid in maintaining the network in an operational state: for example, the user dialling an insufficient number of digits and replacing the handset should not leave the system with resources permanently reserved – the timers ensure this. If a single state machine view can represent the complex call setup within a switch then small controlled changes in the state machine can create new services. Therefore, a resilient network demands that operators and vendors develop a complete understanding of the BCP.

The traditional approach to providing network intelligence was by providing enhancements to the basic call state model. This is supported by looking at the development of the capability sets of the IN. For instance, the introduction of Call Party Handling (section 2.4.6) in IN CS-2, allowed further call manipulation by the end user, even though the service offering was still that of telephony. It must be kept in mind, however, that the additional functionality still operated within the "constrained environment" of the BCSM and the BCP.

* A complete description of SDL can be found in [Ells97].

Chapter 3 examines the changes to the IN architecture in a converging environment where, for instance, telephone calls can be initiated on an IP-network and terminated on the PSTN network. Specifically, section 3.5 discusses the use of state models within a converging environment.

2.7 CHAPTER SUMMARY

This chapter provided the background needed for the discussion of the new research work that is presented in the rest of this thesis. Initially, the reader was introduced to the area of telecommunications, with later sections focusing on specific architectures of the control plane, such as the signalling network and the Intelligent Network architecture.

The IN architecture is a PSTN control-plane architecture that is transparent to the PSTN's switching procedures, and has minimal impact on existing equipment. The main points regarding the IN architecture are summarised below.

- The IN, through separation of service logic from the call-processing logic, provides for a service-oriented view of the network. This separation and the adoption of re-usable components (SIBs) allows the fast deployment of new services (section 2.4).
- The IN Service Processing Model highlights the IN approach for service execution: basic call-processing is carried out in the switches, whereas the supplementary service logic resides on a higher level. The communication between the two planes is achieved through detection points and triggers (section 2.4.5).
- IN CS-2 enhances IN CS-1 by allowing communication across network boundaries.
- The basic call model (section 2.4.4) represents a standardised view of call-processing functions to external service logic and, as such, provides the framework for IFs between the SSF and SCF. This means that IN service logic located in the SCF "sees" a call only by means of the information it receives from the SSF, i.e. the received IFs, based on the states identified in the BCM.

This chapter has focused on the traditional role of IN: to provide the means of enhancing the basic call process, by providing a control plane for the PSTN network. This control is heavily based on finite state models, with the robust architecture of the underlying SS7 protocol.

CHAPTER 3

TOWARDS A NEW ROLE FOR NETWORK INTELLIGENCE IN THE CONVERGING ENVIRONMENT

The aim of this chapter is to look at the ways the IN architecture can inter-work with IP-based protocols and architectures – in the current state of convergence – in order to present how network intelligence architectures can fit together.

3.1 INTRODUCTION

The traditional PSTN network is converging with IP-based networks. This has necessitated a re-think of the traditional role of the PSTN and the way services are to be implemented in the IP domain. In order to appreciate this change, this chapter looks at the role of the various layers in the traditional PSTN domain, including the control plane and the Intelligent Network (IN) architecture.

Within the converging environment, PSTN services have to be re-evaluated because the IP domain is adopting PSTN services and making them available. Therefore, as part of the convergence process, the PSTN is being more integrated in the IP domain rather than the other way round. Part of this re-evaluation concerns the analysis of the new role of the IN in a converging environment.

Examining the way in which the IN environment can be used in the converging environment is complex not because the IN and IP architectures are complex *per se*, but because such a proposition gives rise to a large number of new architectural offerings. The complexity in examining these architectures is increased further as there is a large number of basic components that contribute towards a single architecture. For example, the PSTN and Internet Inter-working framework (PINT) [RFC2995] architecture (which is discussed in section 3.3.2.1) is based on the SIP [RFC2543] and SDP [RFC2327] protocols. The inter-dependence exhibited by the majority of the architectures requires that key complementary protocols also be discussed.

Section 3.2 presents IP-based networks and highlights the various characteristics of the network, architectural details, and the IP-approach towards service provisioning. The taxonomy reference model is then presented in section 3.3. The taxonomy presents the various ways the inter-working propositions come together. The network intelligence architectures (i.e., the protocols and the architectures) that populate the taxonomy represent ongoing research activities by various organisations, working groups and industry bodies.

In chapter 2, the TINA model was presented with the focus being on the relation of TINA to IN. Section 3.4 extends the work already presented by examining a role of the TINA service architecture in the converging environment. As in chapter 2, the work presented is not focused on the TINA architecture, but rather on where TINA could be positioned within a converging environment.

Section 3.5 provides a discussion on network intelligence, state models and the role of IN in a converging environment. Finally, section 3.6 presents a summary of the chapter and identifies research contributions from the work discussed.

3.2 Internet-Protocol Based Networks

In recent years, the Internet has developed to become a global data network, which attracts numerous user types (home users and business users; young and old) with the variety of information and multimedia applications offered online [USDOC][ITU97].

It is important to note that the primary access to the Internet is the PSTN [ITU97]. The PSTN, the largest telecommunications network worldwide, represents an immense investment in infrastructure, and carries all channel-switched public, and substantial corporate, voice and data traffic* [ITU98a][ITU98b].

The Internet was developed mainly for the **data networks environment** [ISOC] where a very different set of service requirements exist.† The Internet is characterised by the interconnection of heterogeneous transport networks and other layer-two technologies. These networks are viewed by users as a single, unified-network architecture, through the use of the IP protocol [RFC760] in association with the User Datagram Protocol (UDP) [RFC768] and the

* This enables the PSTN (through its control plane) to provide some revolutionary services (see section 3.3.3.2 and the specific application presented in chapters 4 and 5).

† Recall that section 2.1.1 highlights the fact that the design requirements of the PSTN were provided by the characteristics of human voice.

Transmission Control Protocol (TCP) [RFC793]. What is important is that there is very little control within the core of the Internet; intelligence is placed at the edges of the network instead.[*]

By contrast, telecommunications networks are performance engineered [McMil96][Scer97] and rest on the tradition of services that are essential to the economic well-being of the country [Cout97][Davi97]. Therefore, IP networks present a challenge to the accepted views of telecommunications developers who are rooted in the performance and reliability requirements of the classic telecommunications sector.

The acceptance of IP-based networks by the business community is a fact, as is the increased use of IP to support PSTN services such as telephony [ITU00]. This presents a number of significant challenges, especially in the short to medium term. Regardless of the level of the dominance of IP in the long term and the effect this may or may not have on the role of ATM, in the short term, there is a need to support the inter-working of widely heterogeneous networks [Deci97][Solo99b]. A call may now originate on a mobile network, be transported across the PSTN, and terminate on a business IP phone. Quite apart from the problems of billing for such services, a voice user has come to expect a range of call control services. A business caller dialling a freephone number from an IP phone should not have to pay for the call regardless of the network type or the chain of network operators through which the call is routed. A corporate Virtual Private Network (VPN) is a business offering that may still be demanded by a company making use of IP calls. These services are traditionally supported by IN platforms but they must now be introduced on IP-based networks.

The confidence of data network providers has become almost unbounded as new markets open up for such services. It is therefore a natural next step for data network operators to look towards the core markets of telecommunications operators [Stro99]. One compelling argument is that the bandwidth of data services is very large in comparison to the voice service [ITU00]. Corporations are being shown that they can make use of spare bandwidth in the data network to support voice [ITU97][Wong99], implying that voice services will almost be free[†]. Whether one

---

[*] Although some routers may offer packet-discrimination services, these are not "intelligent" services according to the definition of network intelligence in section 2.7.

[†] In [Wong99], analyst B. Kasrel of Forrester Research believes that "the U.S. Federal Communications Commission will further erode Internet telephony's price advantage in 2001 by imposing long distance access charges to Internet calls." As a result, "telecommunications companies will have to invent unique services to better compete and stave off a perpetual price war."

believes that it is possible to support voice over IP with a universal-carrier class of service is still the subject of debate and trial. The business case alone is very compelling.

In such a consolidated environment, the business and technical case for the widespread use of IP-based networks to support telecommunication services is now very strong. This view is supported by a number of highly publicised announcements [QOSNet][BTPr01][FT01a] [FT01b][FT01c][Poncin98] by traditional telecommunications vendors pledging their commitment to the development of IP-based products and services. In most cases, this requires more than an expansion of the company's product portfolio. It also necessitates a complete change in business practice. What is driving companies to such a drastic change in policy is a combination of factors [ITU00]:

- IP-based networks are demonstrating their ability to support services, such as voice, that were previously seen to be firmly in the province of traditional telecommunications operators.
- The market for fixed voice telephony is growing at a modestly linear pace and is not demonstrating the exponential demand for equipment and bandwidth that is seen in the case of the Internet or, more generally, IP-based applications [Moha99].

It is projected that the IP market will continue to grow at an even higher rate for several years to come and, in July 2000, the ITU Council selected IP Telephony as the topic of the third World Telecommunication Policy Forum [ITU00]. This emphasises the importance of IP Telephony, which is driving the convergence of circuit-switched and packet-switched networks.

Regardless of the growth and expansion of IP-based networks, the architectural differences between IP-based and telecommunication-based networks still remain. These differences are clearly rooted in their origins (section 2.1.1). Telephone networks have been carefully engineered to provide extremely reliable, high-quality voice transmission, making real-time conversations possible. IP communications are typically connectionless and stateless [Stev94][RFC768][Veer99].

Current IP Telephony developments seek to imitate the more connection-oriented, state-based, PSTN-like circuits [Q.1224][Q.701]. As a result, current IP telephony standards activities attempt to replicate long-established technical practices in the PSTN, such as call set-up and tear-down, IN services and guaranteed QoS. For example, inter-connectivity between the PSTN network and IP Telephony networks has been accomplished by utilising a gateway. The gateway is state-based and converts and forwards calls in one direction or the other [H.323] [H.245][Haer99][Databeam98].

3.3 A Taxonomy Reference Model for the IN Architecture

This taxonomy reference model presents the various ways current research in network intelligent architectures can inter-work with the Intelligent Network. The model is a "snapshot" of the various technologies that utilise the IN control architecture at the point in time when the taxonomy was developed.

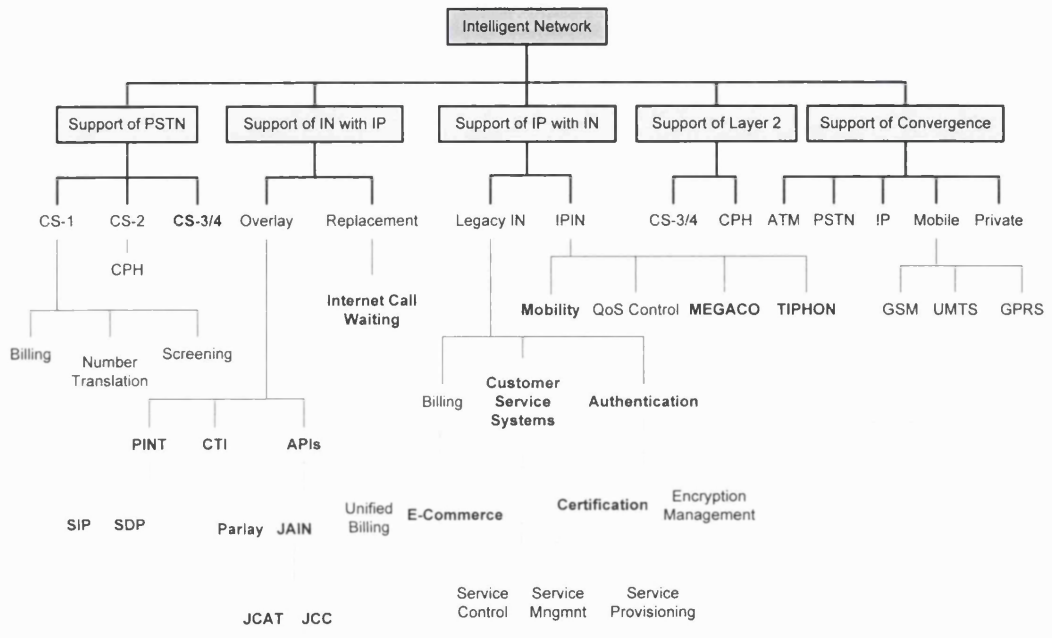


Figure 3.1: *IN control plane taxonomy reference model*

The five areas depicted at the top of the taxonomy classify the network intelligent architectures from the perspective of how they can inter-work with the IN.

- **Support of PSTN:** Essentially, this is the area encapsulated by the original concepts of the IN architecture.
- **Support of IN with IP:** This is the area where IP architectures can be used in order to support the traditional capabilities of the IN model. PINT (discussed in section 3.3.2.1.3) provides services in the IP domain that initiate requests on the control plane of the IN; through PINT, the control plane of the IN can be accessed from the IP domain.
- **Support of IP with IN** This is further divided into two areas, those that involve legacy IN components and those that require new functional entities on the IN side, e.g. IPIN. For example, databases included in legacy IN can be used to provide authentication functionality. IPIN, on the other hand, requires enhancements to the IN architecture.
- **Support of Layer 2**

- **Support of Convergence**

3.3.1 Support of PSTN: The IN as the Control Architecture

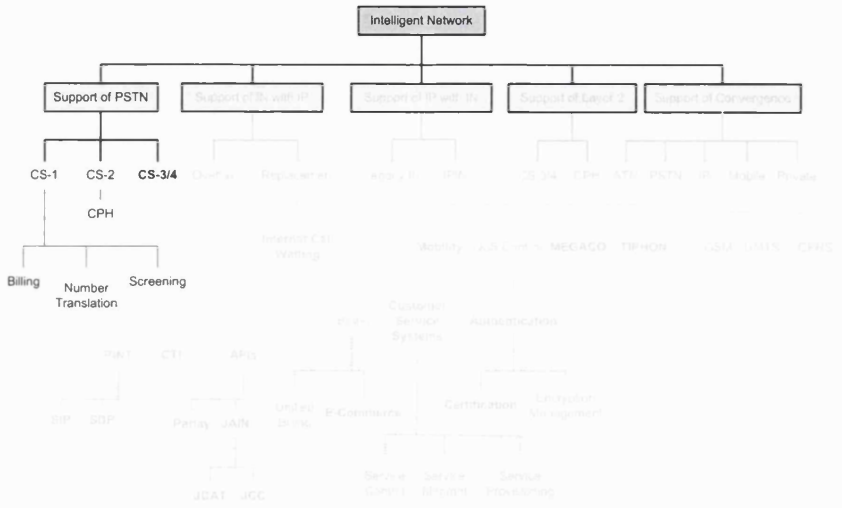


Figure 3.2: *The Intelligent Network providing support to the PSTN*

This branch (represented in bold in figure 3.2) represents the area captured by the **traditional role** of the IN architecture. Example services include billing, number translation, and call screening. Some of these (e.g. number translation) were discussed in section 2.1.2; this section focuses on new functionality that may be provided by future capability sets such as IN CS-3 and IN CS-4.

The IN and IN CS-1 and IN CS-2 were described in chapter 2. This section focuses on capability sets that are under development (i.e. IN CS-3 and IN CS-4) and highlights the service features that are being considered by standards bodies [ETSI EG 201-766][ETSI ETR 199] [ETSI TR 101-779].

One area being considered is that of voice recognition. In this area, a class of services require complex voice processing and little or no additional call routing. Such services are currently provided by interactive voice response (IVR) systems.* Such functionality is now being considered for incorporation in the core of the network. If services such as these are to be implemented in the core network, there is a need for a broadband intelligent peripheral (B-IP) . The Specialised Resource Function (SRF) would provide the interaction functionality with the subscribers, while the service control function (SCF) would manage database functions and perform the "traditional" tasks, such as call routing.

* An IVR unit would be located outside the core network of the operator. Therefore, it would be limited to voice processing, without offering any routing capabilities.

3.3.1.1 Broadband Multimedia Services

An additional area of interest is that involving broadband multimedia services. Multimedia services are characterised by a multimedia session, which is generally defined as being composed of three entities:

- The Multimedia Controller is the entity responsible for the control of the session.
- The Control Connection is established between each user and the Multimedia Controller to exchange session control information.
- The Media Connection is the connection between the users which allows them to exchange media information.

Broadband multimedia services can be implemented using an enhanced SRF that interacts with image and voice signals [ETSI ETR 101-799]. Such an SRF can provide services including the following:

- Video on Demand – the SRF acts as the communication partner to the set-top-box during user-interactive dialogues.
- Video Conference – the SRF acts as a server to merge and to distribute flows to the users in the conference.

For such functionality, the SRF should contain commands to control a media stream, for instance to go backwards and forwards, pause, and skip [ETSI EG 201-766]. The SRF could implement these procedures in the form of User Interaction scripts (UI-scripts). According to [ETSI ETR 199] UI-scripts are advantageous because they:

- allow the grouping of the user interaction parts of the service into functional blocks which use SRF resources in the most efficient way.
- lead to a substantial decrease of network traffic over the SCF–SRF interface since only message exchanges related to the triggering and reporting of script execution actions are necessary.
- represent a generic action which may be parameterised, thereby reducing implementation complexity.

3.3.1.2 Internet (IP)-Based Services

Services in this category include for example, "click-to-fax", "click-to-fax-back" and "voice access to content." For these services, the SRF requires protocol conversion capabilities in order to convert text-to-fax content or text-to-speech synthesis as well as voice recognition.

This group of services is described in section 3.3.2. In the case of PINT (section 3.3.2.1.3), the services are implemented using PINT servers in the IP domain. From the IN perspective, they are mentioned here to illustrate that such services are possible with an enhanced SRF function.

3.3.1.3 Other Multimedia Services

The SRF could be used to implement a Voice Calling Card service, whereby the user is identified based on a speaker verification function rather than personal identification numbers.

3.3.2 Using IP-Based Architectures to Support the IN Model

This branch of the taxonomy (represented in bold in figure 3.3) represents the areas where open standards, framework models and other industry initiatives have proposed protocols which can be used to support the existing IN architecture.

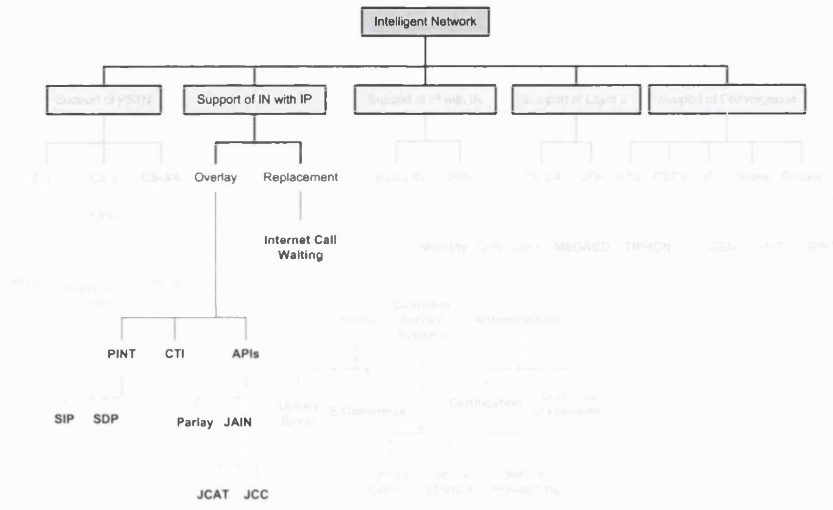


Figure 3.3: *The Intelligent Network is supported by new IP technologies*

This set of initiatives can be further divided into two areas:

- Overlay initiatives, which enhance the IN capabilities and are discussed in sections 3.3.2.1 through to 3.3.2.4.
- Replacement initiatives, which propose a different solution to an already implemented IN service, discussed in section 3.3.2.5.

3.3.2.1 PINT and IETF Protocols and Architectures Requiring IN Inter-operability

In order to discuss inter-operability issues between PINT and IN, the protocols on which PINT is based must first be presented. Therefore, this section presents protocols defined by Working Groups of the Internet Engineering Task Force (IETF) [IETF]:

- The Session Initiation Protocol (SIP), presented in section 3.3.2.1.1,
- The Session Description Protocol (SDP), presented in section 3.3.2.1.2,
- The PSTN and Internet Inter-working Framework (PINT), presented in section 3.3.2.1.3,

- IN CS-4 developments in relation to PINT are discussed in section 3.3.2.1.4.

3.3.2.1.1 The Session Initiation Protocol

The **Session Initiation Protocol** (SIP) is an application layer protocol that "is used to establish, modify, and terminate multimedia sessions" [RFC2543]. SIP is a text-based client–server protocol that relies on HTTP-type requests and can run over TCP or UDP[*]; however the message format is independent of the transport protocol.

An SIP server can operate in two modes: proxy mode and redirect mode [RFC2543]. In proxy mode, the proxy server returns responses on behalf of the user; the server takes care of the location of the user and in this way the process is transparent to both clients. In redirect mode, the SIP server locates the user and returns this information to the initiating client, who then contacts the terminating client directly. The media to be exchanged in a SIP session is described by the Session Description Protocol.

3.3.2.1.2 The Session Description Protocol

The purpose of the **Session Description Protocol** (SDP) is to "convey information about media streams in multimedia sessions to allow the recipients of a session description to participate in the session" [RFC2327]. A multimedia session, for these purposes, is defined as "a set of media streams that exist for some duration of time" [RFC2327]. SDP includes information about the type of media (video, audio, etc.), the transport protocol (RTP[†]/UDP/IP, H.320) and the format of the media (H.261 video, MPEG video).

3.3.2.1.3 The PINT Architecture

The **PSTN and Internet Inter-networking** Working Group of the IETF (PINT WG) defined the PINT reference architecture, depicted in figure 3.4 [RFC2995]. PINT aims to study the architecture and protocols needed to support services in which a user of the Internet requests initiation of a telephone (i.e., PSTN-carried) call to a PSTN terminal.

The PINT WG has examined services that are initiated in the Internet domain and carried out in the PSTN domain. Examples of the initial proposed PINT services [RFC2995] are:

- Request to call (Click-to-dial): a request is sent from an IP host to initiate a phone call.

[*] If UDP is used as a transport layer protocol, the application layer must implement mechanisms to provide reliability such as re-transmissions and loss detection mechanisms.

[†] This is the Real-Time Transport Protocol, defined in [RFC1889]

- Request to fax (Click-to-fax): a request is sent from an IP host to deliver a fax to a fax machine. The request must contain a pointer to the fax data (which could reside in the IP network or on the PSTN).
- Request to hear content (Click-hear-content): a request is sent from an IP host to make a phone call to a user and dictate some sort of content. The request must either contain a URL pointing to the content or the content itself.

A PINT system consists of the following functional elements (figure 3.4):

- PINT Client: An IP host that sends requests for invocation of a PINT service.
- PINT Gateway: An IP host that accepts requests for PINT services and dispatches them to a PSTN network.
- Executive System: A system that interfaces the IP network to a PSTN network that is executing the PINT service.

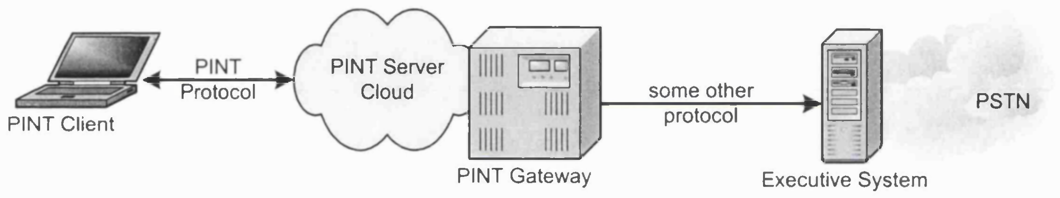


Figure 3.4: *PINT architecture*

It is important to emphasise that PINT services always involve two networks: the PSTN and the Internet. As a result, the control of a PINT service resides in both the IP and IN domains and communication between the two networks is required. This is the role of the PINT gateway.

The system of PINT servers in figure 3.4 is represented as a cloud to emphasise that a single PINT request might traverse a series of location servers, proxy servers and redirect servers before finally reaching the PINT gateway that can actually process the request by passing it to the executive system on the PSTN network.

The PINT gateway might have a physical PSTN network interface, or it might be connected via some other protocol or Application Programmers Interface (API) to an Executive System that is capable of invoking services within the PSTN network cloud. The relation between PINT, the IN and the role of the Executive System is discussed further in section 3.4.

3.3.2.1.4 PINT and IN CS-4

The interfaces between PINT and the IN are under development by the ITU-T Study Group 11 whose area of responsibility includes, amongst others, signalling requirements and protocols for

IP-related functions and enhancements to the existing recommendations on access and "internetwork" signalling protocols of ATM, N-ISDN and PSTN.

The distributed functional plane (DFP) for IN CS-4 as defined in [ETSI ETR 199][Q.1244] is an extension to the IN CS-2 functional model (section 2.4.4.2) and is intended to be included in the IN CS-4 standards. IN CS-4 will include a new component, a service control gateway function (SCGF), that transmits service requests and responses between the two networks. Figure 3.5 shows the proposed functional architecture.

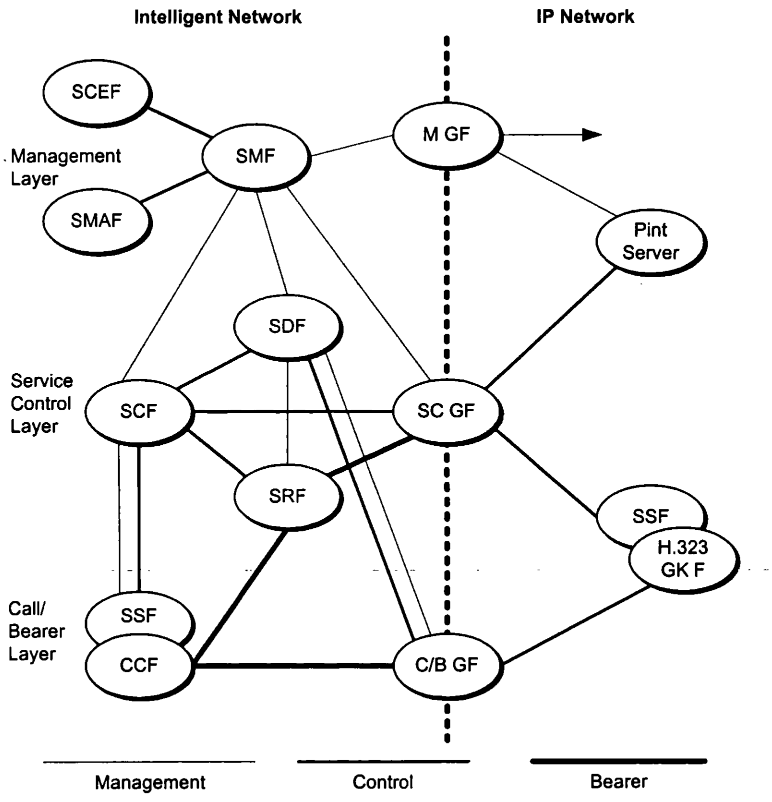


Figure 3.5: Enhanced functional architecture for IN support of IP networks [Q.1244]

MGF	Management Gateway Function
SC GF	Service Control Gateway Function
C/B GF	Call/Bearer Control Gateway Function
H.323GKF	H.323 Gatekeeper Function

The proposed functional model (figure 3.5) is an extension of the IN CS-2 functional model. It is intended to support IN CS-3/4 benchmark services, Internet-based service customisation and termination of VoIP to reach users in the telephone domain, as well as general IN management capabilities.

The functional entities introduced by the model include:

- The SCGF, which allows the inter-working between the service control plane in the IN and IP networks
- The H.323 GKF, which can be seen as a logical switch (CCF) that deals with
    - call control signalling [H.225], [Q.931] and
    - connection control signalling H.245 for VoIP (transferred via the Gatekeeper which makes the network routing decisions)
- The C/B GF, which is equivalent to a composite function combining both the Media Gateway and the Media GW controller as defined in [ETSI TS 101-313].

In order to provide full inter-operability and inter-working of PINT services, a number of additions need to be made to the existing IN functional entities. For example, the SRF has to be extended with capabilities:

- that exchange data with gateway functions to IP networks,
- that support the specialised resources it needs for some of the services, with **media transformation functions**[*] such as Text-to-fax and Text-to-speech, and
- that enable the SCF to access a database-like entity with service-related information to be shared between the IN and the IP network.

A similar approach that aims to provide inter-operability across the two networks is proposed in [Lebo00]. The Soft-Switch [SOFTSWITCH] acts as an overlay between the IP telephony call control and the IN layer provided by the SSF and the SCF. The soft-switch provides the necessary mapping between the SIP protocol state machine and the IN BCSM. This is similar to the functionality of the SCGF described by [Q.1244].

The soft-switch approach defines a **Call Manager Function** (CMF), which acts as a mediation node and is responsible for passing service related information to and from the IN service plane [Lebo00]. This approach is under consideration by the ITU SG-11 and there is ongoing work to define such a CMF.

The CMF is a functional entity responsible for handling call signalling on either network and appears, to the CCF on the IN side, as another CCF. The CMF is responsible for passing service-related information to and from the IN service plane, namely the SCF, and managing the service control relationship [Lebo00].

[*] This is covered in section 3.3.5.2 of [Q.1244] as the TTS function.

The CMF also contains a **Session Manager** responsible for managing the IP network services. The Session Manager is responsible for Security and Authentication, real-time data collection and triggering of services in the IN or IP domains [Lebo00]. To implement such functionality, the session manager may contain SSF-like functionality or a subset, to model the pre- and post-conditions that are required to interact with an SCF.

Also introduced in [Q.1244] is a Soft Service Switching Function (Soft-SSF). The Soft-SSF interacts with the IN SCF and IP representation of the CMF, mapping the Call Control protocol into the INAP events, trigger points and procedures. The Soft-SSF differs from the classical SSF as follows:

- Processes such as call control, database and billing are retained or enhanced,
- SIP-server inter-working functions are introduced and
- circuit-switching and ancillary processes are removed.

In order to provide such inter-operability, it is essential for the interface between the SIP-server and the Soft-SSF to carry sufficient call data for the SSF to function correctly, and to deliver the necessary information to the SCF so that service logic decisions can be made.

3.3.2.2 Computer Telephony Integration Call Model

Computer Telephony Integration (CTI) enables a private enterprise to route calls within the organisation and to manage caller information that is stored within databases, for example in customer management systems. Devices in a CTI environment are application-specific, predominantly proprietary such as Interactive Voice Response Systems (IVR), Voice Mail Systems, E-mail Voice Gateways, Fax Servers, Switches (PBX), Automatic Call Distributors (ACDs) and Predictive Diallers. Figure 3.6 shows a possible CTI network configuration.

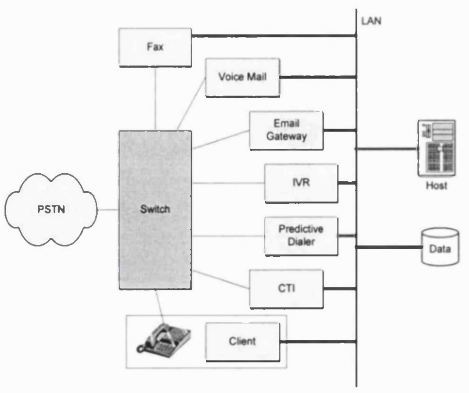


Figure 3.6: *Possible CTI network configuration*

In [ECTF97] the definition of **connection state** as applied to CTI is that the "connection has an associated connection state, an attribute that characterises the relationship of a call and a particular device to each other. Connection states must transition in conformance with the following diagram." The connection state model that CTI is based on is presented in figure 3.7.

The CTI Connection state model re-enforces the different approaches that are adopted by the IN world and the IP world. Comparing the model with that of IN CS-2, it can be seen that it is more "flexible", with numerous transitions.

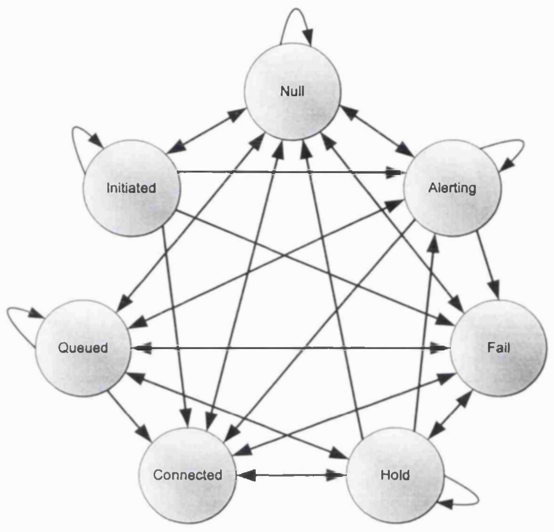


Figure 3.7: *Connection state model for CTI [ECTF97]*

Within the CTI space, Java Telephony API (JTAPI) [JTAPI] is an example protocol that is used for controlling PBXs. Although it does not communicate directly with any IN control elements, it is discussed here because, as will become apparent, the state models of the JCC API (which is used for communicating with the IN) are identical to those of JTAPI.

3.3.2.3 Java Telephony API

The Java Telephony API (JTAPI) focuses on call processing and applications for a private branch exchange (PBX) or call centre environment [Dawk97]. In such an environment, processing and control tend to be centralised in comparison to the distributed nature of the IN.

A call within the context of JTAPI refers to a communication session among two or more parties. Each party (as in the IN world) is said to be participating in one leg of the call. Moreover, a call has as many call legs (connections) as the number of parties in the call.* [JTAPI].

* This is very similar to the O/T_BCSM discussed in section 2.4.4.

The telephony classes that are included in the core model and their relationships are shown in figure 3.8. This section provides a brief discussion of the objects within the model.

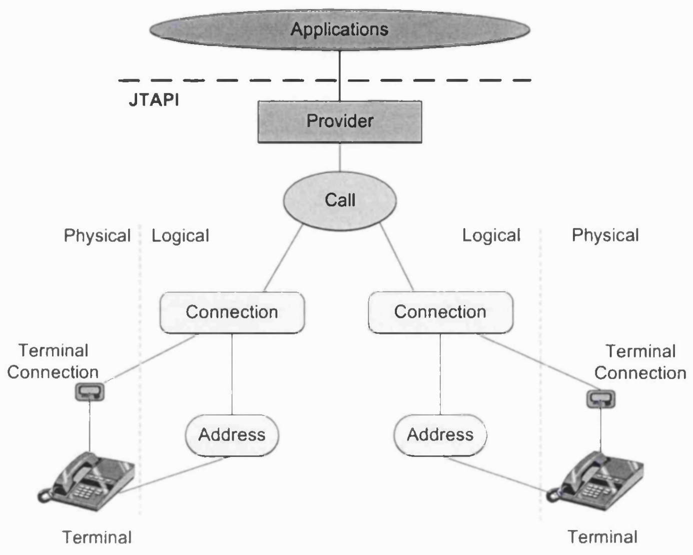


Figure 3.8: *Objects within the JTAPI model [JTAPI]*

The *Provider object* represents an abstraction of the telephony service provider. The *Call object*, which is managed by the *Provider object*, represents a telephone call. A telephone call comprises a *Call object* and zero or more connections.[*] For instance, in the case of a three-party call there would be three *Connection objects* associated with the *Call object*. A *Connection object* models the (logical) communication link between a *Call object* and an *Address object*. An *Address object* is therefore a logical endpoint. The *Terminal object* represents a physical device (such as a telephone) and its associated properties. Multiple *Terminal objects* can be mapped onto the same *Address object*.

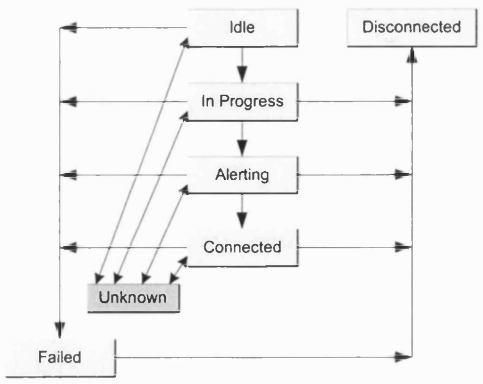


Figure 3.9: *State model for Connection object*

[*] As is the case for CSAs in IN CS-2 discussed in section 2.4.5.

A *Connection object* is responsible for modelling the state that reflects the relationship between a *Call* and an *Address object*. The state transition diagram for the *Connection object* is depicted in figure 3.9.

The states are described in table 3.1.

IDLE	The initial state for all *Connection objects*. An idle connection indicates that the party has just joined the telephone call.
IN PROGRESS	Indicates that a telephone call is currently being placed to this destination endpoint.
ALERTING	Indicates that the destination party of a telephone call is being alerted to an incoming telephone call.
CONNECTED	Indicates that a party is actively part of a telephone call.
DISCONNECTED	Indicates that a party is no longer a part of a telephone call. No methods are valid for *Connection objects* in this state.
FAILED	Indicates that a telephone call placed to the endpoint has failed.
UNKNOWN	Indicates that the *Provider object* cannot determine the state of the *Connection object*. A connection may transition in and out of the UNKNOWN state at any time, unless it is in the DISCONNECTED or FAILED states.

Table 3.1: *Description of states for a Connection object*

Similarly, figure 3.10 depicts the state model for the *Terminal Connection object*.

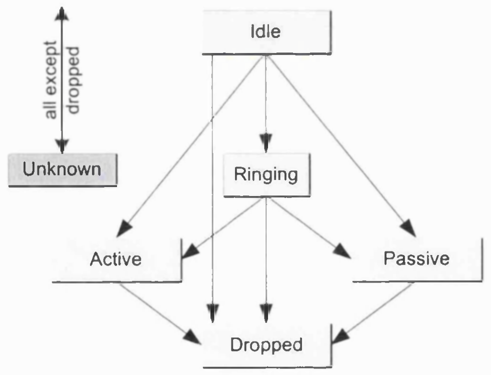


Figure 3.10: *State model for Terminal Connection object*

The states are described in table 3.2.

IDLE	The initial state for all new *Terminal Connection objects*.
ACTIVE	Indicates that a *Terminal* is actively part of a telephone call. This often implies that the terminal handset is off-hook.
PASSIVE	Indicates that a *Terminal* is part of a telephone call, but not actively so. Indicates that a resource on the *Terminal* is being used by this telephone call.
RINGING	Indicates that a *Terminal* is signalling to a user that an incoming telephone call is present at the *Terminal*.
DROPPED	Indicates that a *Terminal* was once part of a telephone call, but has since dropped off from that telephone call. This is the final state for all *Terminal Connection objects*.
UNKNOWN	Indicates that the *Provider* cannot determine the state of the *Terminal Connection object*.

Table 3.2: *Description of states for a Terminal Connection object*

A point of interest in the above state models is the UNKNOWN state. As shown in table 3.2, the UNKNOWN state indicates that the *Provider* cannot determine the state of the *Terminal*

Connection object. In IN, no such case exists and invalid transitions have graceful degradation by moving to an exception state. This indicates the different approaches adopted in call control.

The IN control architecture is characterised by robustness and strict control, while the CTI approach is more relaxed. For the two approaches to inter-work, there needs to be a way to map the robust transitions of the one side to the less demanding approach of the other.

It has been argued [Jain00] that JTAPI overcomes several of the limitations of the IN. According to the same reference, in the IN world "there is no explicit abstraction offered to allow the programmer to manipulate entire calls, or legs of a call, or the principal logical entities in the call (e.g. the calling or called party's address), and certainly not in any object-oriented fashion." It is the author's view that the reasons for this are to do with the fact that the telecommunications network was never open to third-party service providers. It is easier to provide an API-type interface to a CTI environment, where the equipment and the network are owned by the organisation that may be developing the services.

The author agrees that JTAPI offers the programmer clear and explicit abstractions for manipulating calls and the logical entities in a call in an object-oriented [Jaco92] manner. However, it must also be kept in mind:

- that JTAPI is mainly applicable to PBXs or large-scale VPNs. Such systems are much smaller rather than the average IN implementation,
- JTAPI systems are centralised (e.g. a PBX), and that
- a *Provider object* is assumed to be in control of all the legs of a call, which is clearly impractical in integrated next-generation networks.

Moreover, comparing the JTAPI models with the BCSMs of IN CS-1 and CS-2 (section 2.4.4), one can quickly realise the simplicity of the FSMs employed by JTAPI in comparison to the IN BCSMs:

- JTAPI does not currently capture all the states that the IN model does (refer to section 2.4.3).
- JTAPI has no concept of **triggers** or **detection points** [JTAPI].
- it is impossible to suspend call processing at a defined state in the FSM, invoke an application (supplementary service logic), and return results.
- JTAPI includes a number of unknown states and "a connection may transition in and out of the UNKNOWN state at any time, unless it is either in the DISCONNECTED or FAILED states" [JTAPI].

In order to enhance the capabilities provided by JTAPI, the JCC/JCAT API is attempting to incorporate the "powerful aspects" of JTAPI (such as object-orientation) and the flexibility and robustness of the IN models. This is discussed in section 3.3.2.4.3.

3.3.2.4 Open Inter-working Standards

Within telecommunications networks, the desire for new business growth has been a major driving force towards the development of open network APIs, such as the **Parlay** API [ParlSpec00]. The Parlay API enables network operators and third parties (external companies, operating outside the secure domain of the network operator) to build new applications that rely on real-time control of network resources.

A second community, driven mainly by Sun Microsystems, is developing the Java APIs for Integrated Networks (JAIN) [JAIN]. **JAIN** defines a Java implementation of the Parlay API to bring the benefits of the Java language to the Parlay API. This section looks at the background and rationale behind the work of Parlay and JAIN.

Traditionally the network operator, in conjunction with network equipment providers, has designed, developed, deployed, and administered applications that run above switched voice and data networks. These applications typically suit mass-market demand for services such as virtual private networks (VPNs), inbound services, and unified messaging [ParlBuss99].

According to [ParlBuss99] the potential for innovation that lies outside the network operator's domain has remained unexploited until now. By enabling third-parties (such as service providers) to build and deploy new applications on their network, the network operator can reap the benefits of increased network revenues while not enduring the overhead costs of deploying specialist applications. APIs provide increased network traffic through greater exploitation of network intelligence capabilities by a much wider development community, and the opportunity to charge* for access to these via the API. Moreover, from the service provider's perspective there is significant market opportunity for new services and the ability to address niche market requirements. Furthermore, end users can benefit from the vastly reduced time from identifying a requirement for a new service to that application solution being developed. A further discussion of the issues surrounding open network access can be found in chapter 8.

* There are issues with charging for such services. These are identified in chapter 8.

3.3.2.4.1 The Parlay Group

The Parlay group was formed in March 1988, by BT, Microsoft, Nortel, Siemens and Ulticom [PARLAY]. The initial set of APIs was published in December 1998 as the Parlay API 1.0 specification. The latest specification of the API is version 2.1 dated March 2001.

The Parlay API defines a set of technology-independent interfaces that specify methods, events, parameters and their semantics to allow external (untrusted third-parties) and internal (traditional network operators) application developers the control over core network resources and capabilities [ParlSpec00].

The applications execute on the enterprise domain utilising the network capabilities offered via the API. The latter defines object-oriented interfaces on both the network and client application sides of the API in the form of network interfaces (e.g., IparlayCall) and client application callback interfaces (e.g., IparlayAppCall). The third-party application vendor implements callbacks as part of the application to handle remote methods that are called from the network to the client application during a Parlay session.

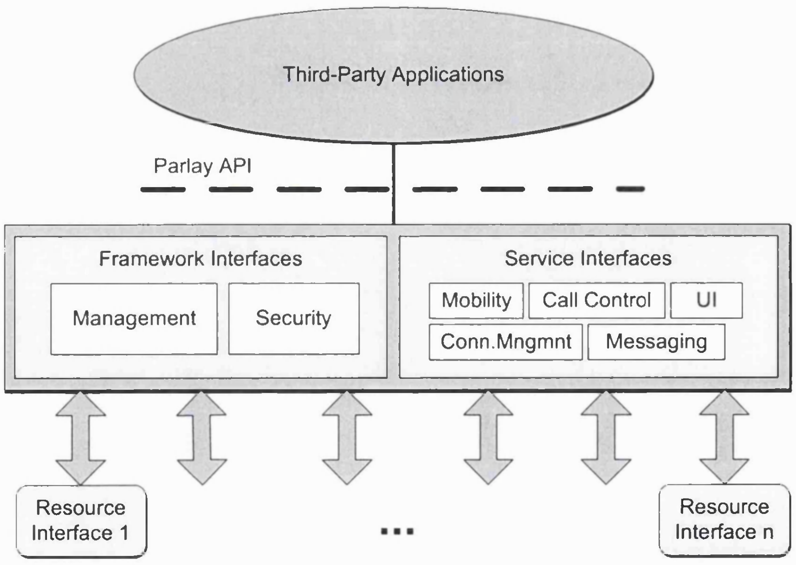


Figure 3.11: *Architecture of the Parlay API*

As shown in figure 3.11, the Parlay API is composed of two sets of interfaces:

- **Framework interfaces** provide the capabilities necessary for the Service Interfaces to be open, secure, resilient and manageable [ParlSpec00]. The framework can be considered as a number of functional building blocks and is independent of any of the Parlay services. In order to access the Parlay framework, the network must authenticate itself because ultimately the client application shares private data with the network. Similarly, the network must prove its identity to the client for repudiation reasons.

- **Service interfaces** provide the mechanism by which applications can access underlying network capabilities. Parlay has defined five services: Call Control, Mobility, User Interaction, Messaging and Connectivity Management [ParlSpec00]. The service interfaces, such as the generic messaging service (GMS), provide access to the capabilities necessary to support intelligent network, integrated services digital network user part (ISUP), H.323 and unified messaging applications. The high level of abstraction of these services ensures that both existing voice networks and VoIP networks can be controlled in the same way, giving independence of the technology. Each service has a service manager responsible for control, object creation and event notification.

The Parlay Group's prime focus is to define a computing and distributed technology-independent API for controlling voice and data networks. The Parlay API has utilised distributed computing technology specifications in order to make it applicable to the real world. Such examples include the definition of the Parlay API using the Distributed Component Object Model (DCOM) [DCOM] and CORBA [CORBA95]. This approach enjoys the flexibility of being able to map to multiple programming languages such as C++, C, Java and Visual Basic. Advocates of Parlay have argued that the application developer must be skilled in three areas: the Parlay API, distributed computing techniques, and programming languages [Bedd00]. The specification of a language-dependent API on the client side removes the distributed computing element and even more so if the API is platform-independent. One such API is the JAIN initiative from Sun Microsystems.

3.3.2.4.2 JAIN: Integrated Network APIs for the Java Platform

The objective of the JAIN initiative is "to create an open value chain from third-party service providers, telecom providers, and network equipment providers to telecom, consumer and computer equipment manufacturers" [Bhat00]. JAIN builds on Java portability by standardising the signalling layer of the communications networks into the Java language, and defines a communications framework for services to be created, tested, and deployed. According to [Keiz00] the strengths of JAIN are in service portability, network convergence, and secure network access. Firstly, the uniform use of Java interfaces is utilised to deliver portable applications. Secondly, the JAIN call model [JAIN] includes facilities for observing, initiating, answering, processing and manipulating calls irrespective of the underlying multi-network path of the call. Finally, through the use of the JAIN Parlay interface it is possible to enable untrusted services, residing outside the operator's trusted network, to access network resources directly and carry out specific actions or functions inside the integrated network.

The JAIN architecture comprises a **JAIN Application Server** and the **JAIN Softswitch Platform** [Bedd00]. The Application server is responsible for implementing the JAIN Service Provider APIs. These allow secure access to network resources. The JAIN Softswitch Platform ensures that there is a mapping between the JAIN Call Control elements to the network and signalling layers. The network layer includes the IN functional entities, as well as SS7 including ISUP, INAP, TCAP protocols, wireless networks with access to the MAP layer, as well as the Internet with access to SIP, the Media Gateway Control Protocol (MGCP) and H.323. The Signalling layer includes access to functional entities such as SSPs, MSCs, and also H.323 gatekeepers in the IP domain.

Figure 3.12 depicts the position of JAIN APIs within a communications platform. It shows the hierarchical use of APIs at various levels: the protocol APIs, call control APIs and service APIs.

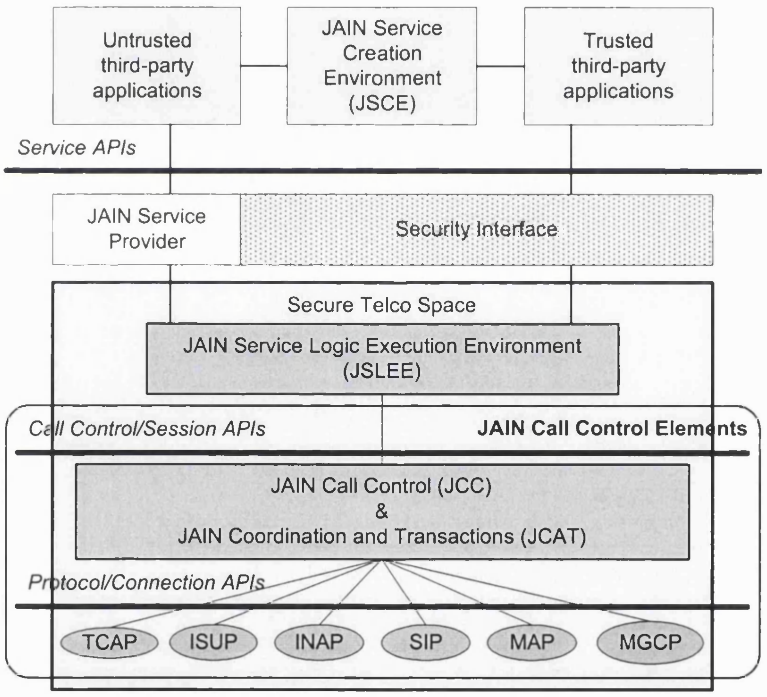

Untrusted third-party applications
JAIN Service Creation Environment (JSCE)
Trusted third-party applications
Service APIs
JAIN Service Provider
Security Interface
Secure Telco Space
JAIN Service Logic Execution Environment (JSLEE)
Call Control/Session APIs
JAIN Call Control Elements
JAIN Call Control (JCC) & JAIN Coordination and Transactions (JCAT)
Protocol/Connection APIs
TCAP
ISUP
INAP
SIP
MAP
MGCP

Figure 3.12: *Location of JAIN within a communications network [JAIN]*

3.3.2.4.3 Integrating Control Elements from JCC, JCAT and JTAPI

Presented here are the details relating specifically to the finite state model representation of the call model within JCC and JCAT. JCC is responsible for the basic call-processing and control. The JCAT extension package is responsible for providing coordination and transaction-related methods. According to [Jain00], the call model within JCC is identical to that of JTAPI Release 1.2 (section 3.3.2.3).

JCAT additions include an extension of the *Connection object* FSM to become a richer FSM similar to that for IN. There is also a proposal to provide a version which includes both the O_BCSM and T_BCSM of the IN model in a single FSM. Furthermore, IN-style triggers will be added. These could be implemented by:

- requiring applications to register with the *Address object;* in this way, the application would be invoked when a particular trigger in the *Connection object* FSM fires.
- treating each transition as a trigger; an application registered with that transition would execute, while call processing would be halted until the application returns and call processing can resume.
- implementing callback objects and interfaces; this however, needs to deal with the problem of feature interaction – which is beyond the scope of the JCC/JCAT API.

It remains to be proved if JCAT can capture the richness, completeness and robustness provided by the IN state models.

Having briefly introduced the architecture of the JAIN APIs, the next section discusses the work that is currently under development by the JAIN Parlay Edit Group in order to provide a Java implementation of the Parlay API.

3.3.2.4.4 JAIN and Parlay

JAIN set up what is known as the JAIN Service Provider API (SPA) Group to look into the development of a Java technology API for Parlay. The goal of the SPA group is to "provide the industry with a standard Java technology version of the Parlay APIs" [JAIN]. To achieve this, the group looked initially at how a JAIN/Parlay implementation client can interact with a JAIN/Parlay implementation server, and how that maps onto the existing JAIN standards.

Figure 3.13 shows the JAIN Parlay Edit Group API operating on a third-party client's machine. The client's machine is connected to the network operator across, for instance, an IP network. The transfer mechanism for the messages between the JAIN/Parlay implementation client and the JAIN/Parlay implementation server is implementation-dependent. The JAIN/Parlay implementation server interacts with the JSLEE, JCC and JCAT APIs to use the JAIN community service plane and control plane capabilities.

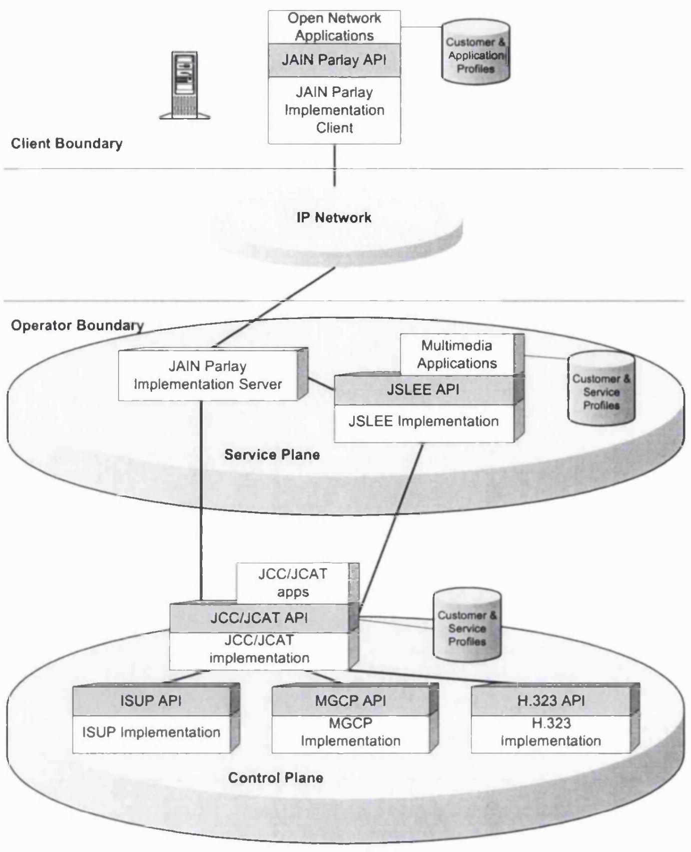


Figure 3.13: *JAIN/Parlay Interactions*

3.3.2.5 Internet Call Waiting: Replacement Initiative to IN Call Waiting

Internet Call Waiting (ICW) is a service that enables a subscriber engaged in an Internet dial-up session to be notified of an incoming call to the same telephone line, to specify the desirable treatment of the call and to have the call handled as specified [Brus98].

In [RFC2995] the desirable features of ICW are identified. Here is a summary:

- Incoming call notification – The subscriber is notified of an incoming call over the Internet, without having any effect on the telephone line that is being used by the modem.
- Online Incoming Call Disposition – Once informed of the incoming call, the subscriber has various options for handling the call.
- Automatic Incoming Call Disposition – Incoming calls are automatically handled based on dispositions pre-defined by the subscriber without real-time intervention.
- Multiple Call Handling – Multiple calls arrive during call disposition processing. With multiple call handling, the subscriber is notified of the multiple calls one by one.
- Call Logging – A detailed log of the incoming calls processed during the ICW service is kept.

In order to highlight the inter-operability and inter-working issues that arise from the ICW service, this section presents the implementation adopted by Korea Telecom [RFC2995]; the network architecture of the Korea Telecom ICW service is presented in figure 3.14.

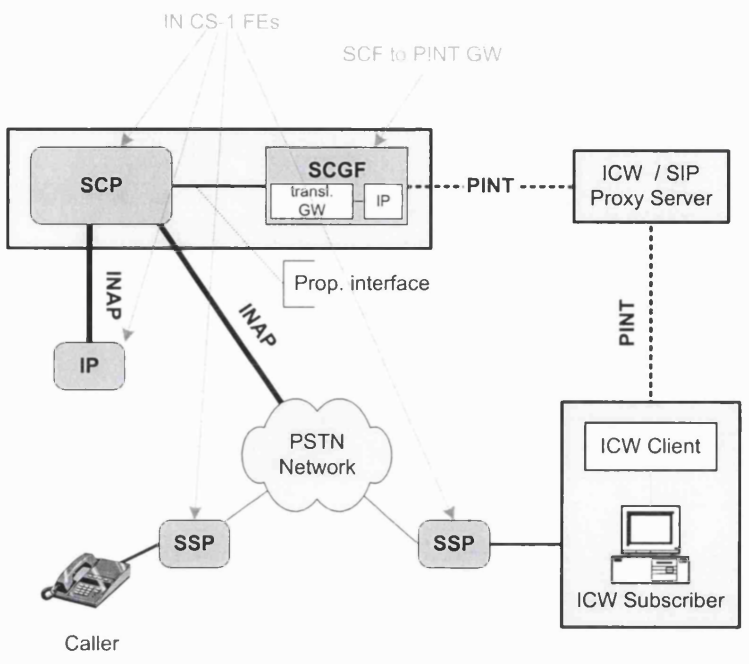


Figure 3.14: *ICW based on existing IN CS-1 FEs [RFC2995]*

- The SSP is a standardised IN CS-1 SSP. On detecting that the called party is busy, it sends a query to the SCP and processes the call under the control of the SCP.
- The SCP processes the call based on the logic associated with that service. In the case of the ICW service, the service logic includes the notification of a waiting call to an online ICW subscriber and the disposition of the call. The service logic requires that the SCP inter-works with the ICW proxy server.
- The SCP-ICW protocol is PINT. The translation between INAP and PINT is performed by the SCGF. For this, a proprietary protocol is used between the SCP and the SCGF, whilst on the other end the SCGF is an IP-endpoint.
- The IP is a standardised SRF. When necessary, it utilises the *Play Announcement* IF to inform the caller according to the settings of the ICW subscriber.
- The ICW server is a SIP proxy or redirect server for message routing between the ICW client and the SCGF. The ICW server is also responsible for:
    - managing the ICW clients that are connected to it,
    - monitoring the connection status of the registered ICW client and
    - managing profiles for each ICW subscriber.
- The ICW client is an application program running on the subscriber's PC. The application monitors the Internet connection status of the PC and, upon connection, sends a registration request to the SCGF via the ICW Server, which is eventually passed to the SCP.

From the above architecture, it can be observed that:

- the implementation depends on a proprietary function – the SCGF – and therefore a proprietary interface between the SCGF and the SCP
- the implementation of the IP-side of the SCGF is specific to the version of SIP that is being used and
- the implementation of the ICW client application is specific to the version of PINT.

In order to resolve some of these limitations, the "Service in the PSTN/IN Requesting InTernet Services" SPIRITS IETF WG [RFC2995] has proposed a standardised architecture for future SPIRITS services [Fayn99]. The SPIRITS architecture proposed in [RFC3136] aims at providing inter-operability between existing commercially available systems, such as the one discussed here.

The SPIRITS WG [SPIRITS] would resolve the last two issues associated with the above architecture, but it would still not tackle the issue of the proprietary interface between the SCGF and the SCP. This could either be resolved using the architecture presented in section 3.3.2.1.4 of the proposed IN CS-4 SCGF or by adopting a Parlay-based approach that would enable third-party service providers to directly access the SCP through a Parlay-compliant API and thus overcome the limitations of the proprietary interface.

3.3.3 Supporting IP-Based Services using the IN Model

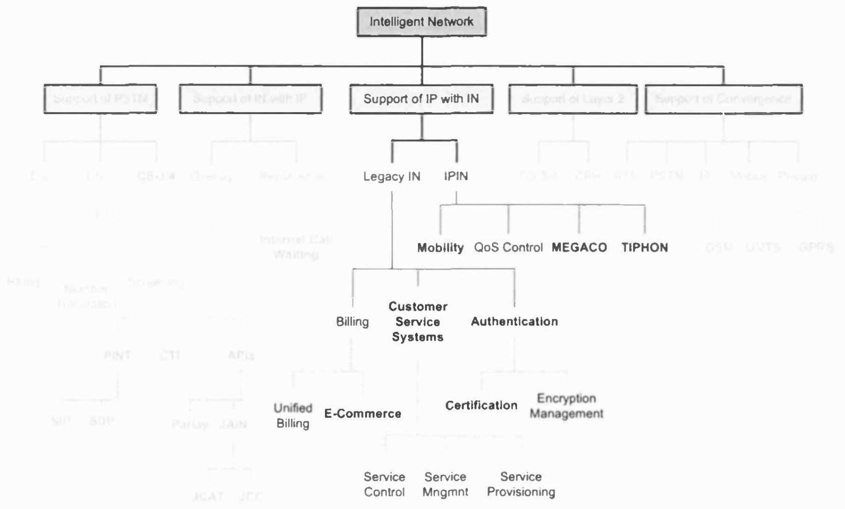


Figure 3.15: *Support of IP with IN*

This section identifies IP services that can be implemented using the functionality provided by existing (i.e. legacy) IN architecture, as well as by introducing new (i.e. IPIN) components on the IN-side. Figure 3.15 illustrates the branch (in bold) of the taxonomy this section focuses on.

3.3.3.1 Customer Service Systems

The IP architecture could be used in conjunction with IN for Customer Service Systems. This includes web-based customer service control and management. Users could in this way subscribe and unsubscribe to services "on-the-fly" via a secure web-based interface.

3.3.3.2 Authentication, Certification, Billing and E-Commerce Services

Such service capabilities use information which is already contained within the IN environment. The IN model and its billing capabilities can be used in order to utilise the important information and "trust" set within the traditional telecommunications network and enhance this by promoting the IN billing capabilities over an IP platform [Solo98].

In this respect, the SCP is treated as a general **certification authority**[*]. The SCP can provide access for all manner of secure information (such as passwords) that could be used in Internet applications. Chapter 4 discusses in detail an example application for this role.

3.3.3.3 DNS for Mobility

This involves utilising the IN infrastructure in order to allow outside control to IN and intelligent use of information within the IN network for services such as mobility. DNS [RFC1591] offers a one-to-many mapping from a globally unique, hierarchical identifier to one or more host names or IP addresses. Therefore, the DNS server provides a means to access location information for servers. This key element in the IP network could be used in conjunction with IN, in order to provide mobility through DNS.

In such a scenario, the SCP can treat the DNS as an SDF. Hence, since IN CS-2 provides enhanced support for user interaction and service profile customisation the user could register a binding between the IP address and URL through the IN. Number translation services are relevant to the Internet. The DNS currently serves as a resolution protocol allowing the translation of fully-qualified domain names to IP addresses. In this sense, the DNS can be viewed as an IN platform [Solo99b]. The difference is that the end terminal (rather than SSP) interrogates the DNS.

[*] A certification authority is an independent party that verifies the credentials of a public key. A complete description can be found in [Ganl01].

3.3.3.4 MEGACO and H.248

Before the protocols of MEGACO, H.248 and the ETSI Project TIPHON are introduced, a brief background to the origins of these protocols is given. Table 3.3 presents an overview of the protocols developed by two of the major standardisation bodies.

Standards Body	**URL**	**Major Standards**	**Notes**
International Telecommunications Union (ITU)	www.itu.int	T.120	Real Time Data Conferencing
		H.248	Gateway control protocol (same as IETF MEGACO)
		H.320	Narrow-band visual telephone systems and terminal equipment
		H.323	Packet-based multimedia communications systems
Internet Engineering Task Force (IETF)	www.ietf.org	SIP	Session Initiation Protocol
		RSVP	Resource Reservation Protocol (prioritises packet traffic by use)
		Diffserv	Differentiated Services
		MEGACO	Same as ITU H.248

Table 3.3: *Standards for inter-working among IP Telephony and the PSTN*

Two major components of the H.323 architecture [H.323] are the Media Gateway (MG) and the Media Gateway Controller (MGC). The MG performs simple encoding and decoding of analogue voice signals, compression and conversion to/from IP packets. The MGC contains all control intelligence, analyses how calls are to be handled and performs functions similar to the SS7 network in the PSTN environment. The MGC needs to understand various signalling systems such as SS7 and GSM in order to ensure PSTN inter-connectivity.

Competing with H.323 is the IETF-developed standard SIP (discussed in section 3.3.2.1.1). A number of papers have been published that make this comparison (either directly or indirectly), such as [Henn], [Liu00] and [SIP].

Although the ITU Recommendations of series H.323 intended to standardise both the media gateway and the media gateway controller, an industry initiative called Media Gateway Control Protocol (MGCP) gained momentum in further decomposing media gateway controllers from media gateways. A result of this initiative was the formation of a working group, named MEGACO [MEGACO], by the IETF. The resulting H.248/MEGACO protocol defines a client–server protocol to control media gateways that can pass voice, video, facsimile, and data traffic between PSTN and IP-based networks [RFC2705]. H.248/MEGACO supports various "packages" that interface with conventional PSTN switches and IN services, with plans to support a range of existing signalling protocols including ISUP, and MAP. The MEGACO architecture is presented in figure 3.16.

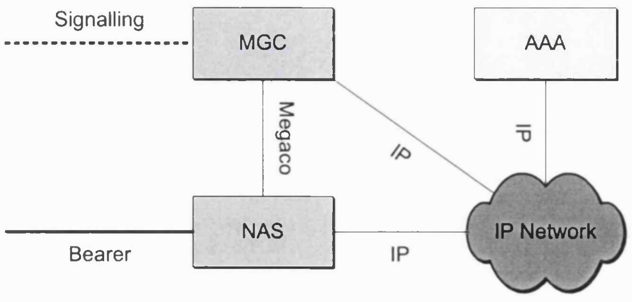


Figure 3.16: MEGACO NAS reference architecture

In figure 3.17, depicting the approach adopted by H.248/MEGACO, the IP, signalling and bearer connections are physical, whilst the interface between the MGC and the Network Access Server (NAS) is logical. The signalling path between the MGC and the NAS server is through the IP network. The NAS is an access gateway, or MG, which terminates modem signals from a network (e.g. switched-circuit network or xDSL network) and provides data access to the packet network.

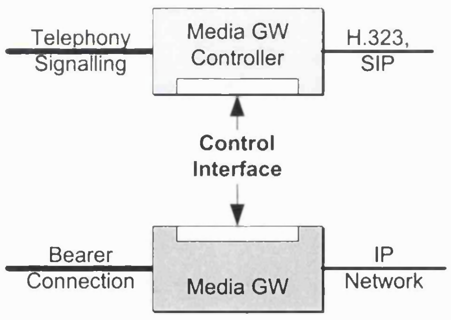


***Figure 3.17:** H.248/MEGACO*

One of the technical challenges raised by the ever-closer integration between circuit-switched and packet-switched networks concerns how to address calls that pass from one to the other. Generally, it is assumed to be desirable that a single integrated global addressing system exists. For example, the same ITU E.164 telephone number would reach a subscriber regardless of whether IP-based or PSTN network technologies are used. Indeed the concept of being "technologically independent" suggests that any global numbering or addressing plan should be abstracted as much as possible from the underlying lower-layer technologies. These issues are addressed by the ETSI Project TIPHON. The following section highlights the work of TIPHON that is related to the control plane.

3.3.3.5 ETSI Project TIPHON

The Telecommunications and Internet Protocol Harmonisation over Networks Project (TIPHON) by ETSI aims at "specifying the inter-operability mechanisms and related parameters to enable multimedia communications to take place, to a defined quality of service, between switched-circuit networks (SCNs) and IP-based networks and their associated terminal equipment" [TIPHON]. TIPHON also supports mobility and roaming within IP-based networks as well as with other networks [ETSI TR 101-300].

[ETSI TR 101-300] identifies five scenarios for different cases of traditional and IP Telephony:

0. IP-phone to IP-phone over an IP network
1. Source on IP network to destination on SCN network
2. Source on SCN network to destination on IP network
3. Source and destination on SCN network using an IP transit network
4. Source and destination on IP network using an SCN transit network.

TIPHON Release 1 defines the architecture presented in figure 3.18.

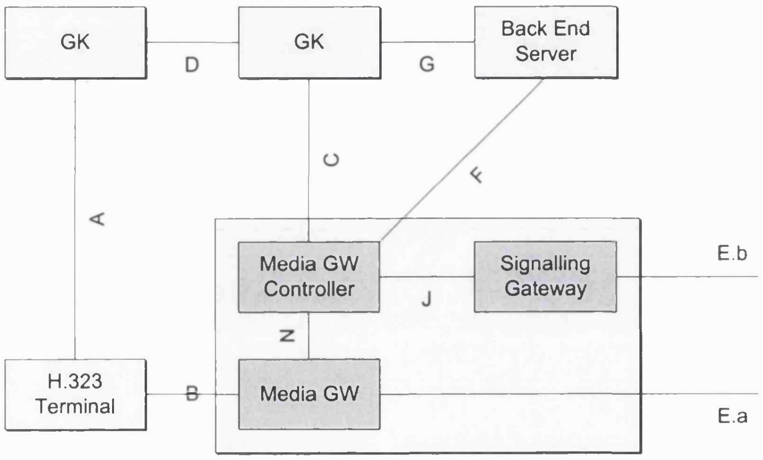


Figure 3.18: *TIPHON release 1 architecture with reference points [ETSI TS 101-312]*

Note the Back End Server (BES) represents services provided by third parties. The BES is further examined in section 3.4.

In order to provide a structured analysis of the requirements, the concept of "functional planes" is adopted by TIPHON (see figure 3.19). Each functional plane contains a high level grouping of functionality.

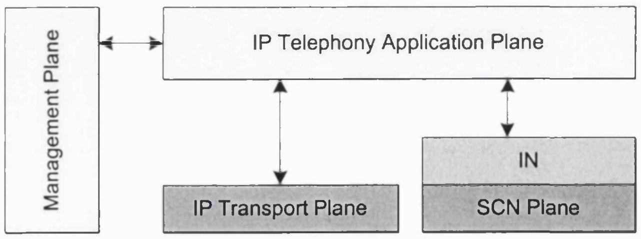


Figure 3.19: *TIPHON release 2 functional planes*

The IP Telephony Application plane is further decomposed into the functional layers depicted in figure 3.20.

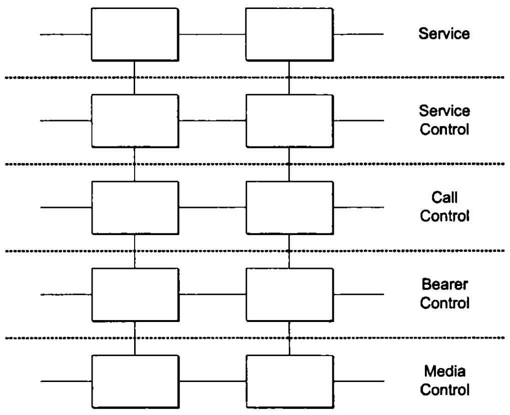


Figure 3.20: *Functional layers in the IP Telephony Application plane [ETSI TS 101-314]*

The services functional layer supports a range of services (e.g. authentication). The service control functional layer contains the functionality that is needed for the calls but may have a life span that is longer or shorter than the duration of the call (examples are terminal registration, call routing). Additionally, the service control functional layer provides number portability, called user location, name-to-name translation, name to address translation, and call access authorisation.

The call control functional layer is responsible for maintaining a call context, which allows the services offered by the bearer control functional layer to provide the connections and capabilities requested by the customer. More importantly, it maintains the call state, as well as providing services that change the call state, for instance call hold, suspend, three way, and conferencing.

A mapping between the IN functional planes and TIPHON's functional planes is given in figure 3.21.

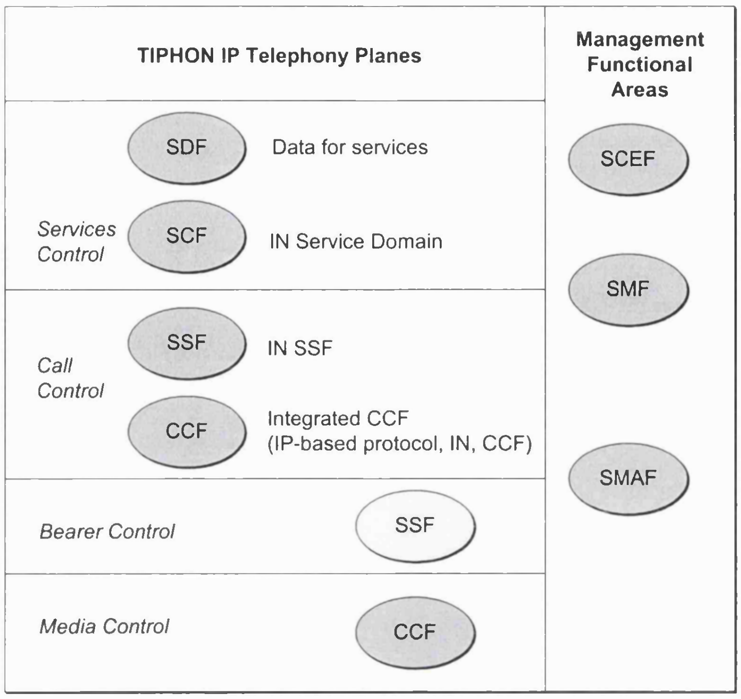


Figure 3.21: *Mapping of IN functions onto TIPHON functional architecture*

As previously mentioned, the original protocols of IP Telephony have limited the inter-operability of IN and IP services. It is vital that IP users also benefit from services in the IN network if a means is found to permit IN service access from IP endpoints. This arises from the fundamental issue of allowing full inter-operability as a natural step to the convergence process [Cian99].

The next section discusses a fundamental issue of providing inter-operability in the context of call models. As call models and their respective FSMs represent building blocks for IN architectures and IP-based services, it is essential that the inter-working among them is achieved without compromising robustness in either domain.

The inter-working of two different call models is a problem of **Call Model Integration** (CMI). A fundamental requirement for CMI is to have a unified view on the two call models (i.e., the IN BCSM and the IP-based FSM, e.g. H.323 or JTAPI).

A call model, in its most general form, consists of three basic components [Vemu99]:

- the **Call Control Element**, which handles call processing related functions,
- the **Service Switching Element**, which handles all service access related functions, and
- the **Feature Interaction Manager Element**, which resolves conflicts in feature access and execution.

Every entity that processes calls possesses these elements in some form. "The CMI issue arises because the base domain has its own FSM representation of these elements, as does the external domain of interest" [Vemu99].

As different services are triggered at different states within the context of a given call, it is essential to be able to maintain a unified view of call state across the two domains. In order to achieve this, the two state models must operate in lock-step (i.e., state changes in one FSM are accompanied by corresponding state changes in the other) [ETSI TS 101-314].

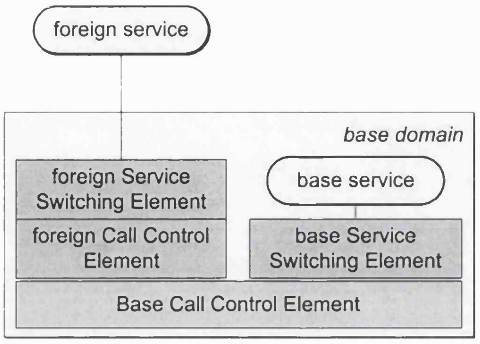

foreign service
base domain
foreign Service Switching Element
base service
foreign Call Control Element
base Service Switching Element
Base Call Control Element

Figure 3.22: Call model integration framework

However, this lock-step behaviour is sometimes difficult to implement, especially since, in most cases, two call model FSMs do not have the same number of states and there may be a one-to-many relationship across the two call models. The concept of state models and their implementations is discussed further in chapter 6. The next section focuses on open inter-working standards using APIs.

3.3.4 Using the IN Architecture for Layer 2 Operations and to Support the Convergence Process

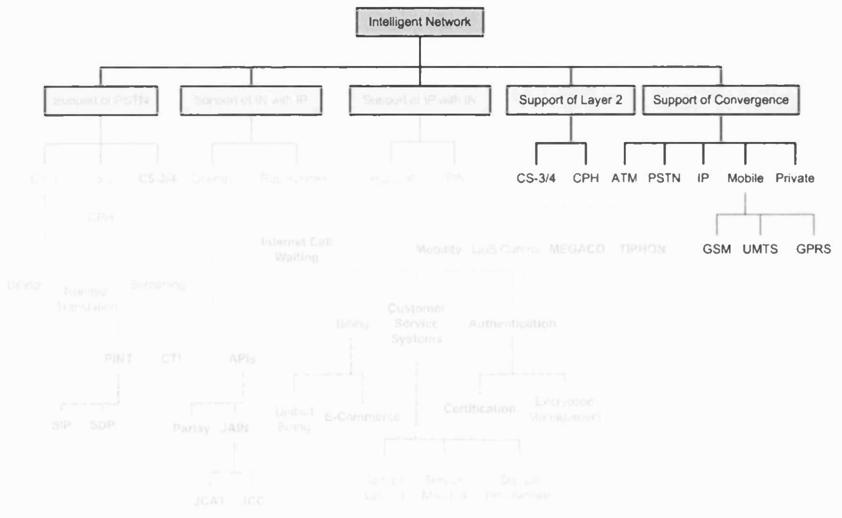


Figure 3.23: ICW based on existing IN CS-1 FEs

The last two parts of the taxonomy (see figure 3.23) identify areas where the IN may provide additional support as a control architecture.

Firstly, to support layer-2* operations, the IN could provide additional QoS control over the ATM AAL2 using additional functionality that may be provided in future IN capability sets.

Secondly, the IN could be viewed as a control architecture that is supporting the convergence process. The number of architectures that need to inter-work in order to provide transparent services are increasing. For example, the following architectures are currently available for mobile telephony: GSM, GPRS and the slow evolution towards UMTS. Similarly, there are IP networks over ATM, as well as the traditional PSTN networks. The PSTN network still is the primary access network for home users, therefore the primary gateway (from the viewpoint of control) to home users is through the IN control architecture.

It is the author's view that the IN has a role to play in supporting this convergence process. Of course, it remains to be seen whether the name "Intelligent Network" will withstand the test of time. What is important however is that the concepts, capabilities and architecture of the IN are contributing and impacting the design of IP-based technologies.

The next section moves on to re-examine the TINA service architecture (section 2.5) within the context of the converging environment and the various protocols and architectures that have been presented in this chapter.

3.4 The Role of the TINA Service Architecture in a Converging Environment

Section 3.3.3.5 presented the ETSI Project TIPHON and its architecture. As already mentioned, TIPHON aims at "standardising the interfaces between SCN networks and IP-networks to enable their inter-operability for voice and multimedia applications" [ETSI TR 101-300]. The TIPHON architectural configuration defines, among other entities, a *Gatekeeper* (GK), whose main function is number translation, and a BES (section 3.3.3.5, figure 3.18).

This section examines TIPHON, PINT [RFC2995] and the new role of IN within the context of the TINA service architecture (section 2.5).

* Layer 2 here refers to the ATM Adaptation Layer

In a converged environment QoS guarantees must be made between homogeneous network types as well as heterogeneous network types. For example, one may wish to establish a call between three parties, each originating from a different network type. It has been argued [Solo00a] that the IN architecture provides an existing and evolving set of standards that will facilitate the migration towards these types of scenarios. Hence, IN should be viewed as an important architecture for the communication session as illustrated in figure 3.24. The figure shows the view of the communication session controlled by the SCPs. The service session is seen as a distinct entity.

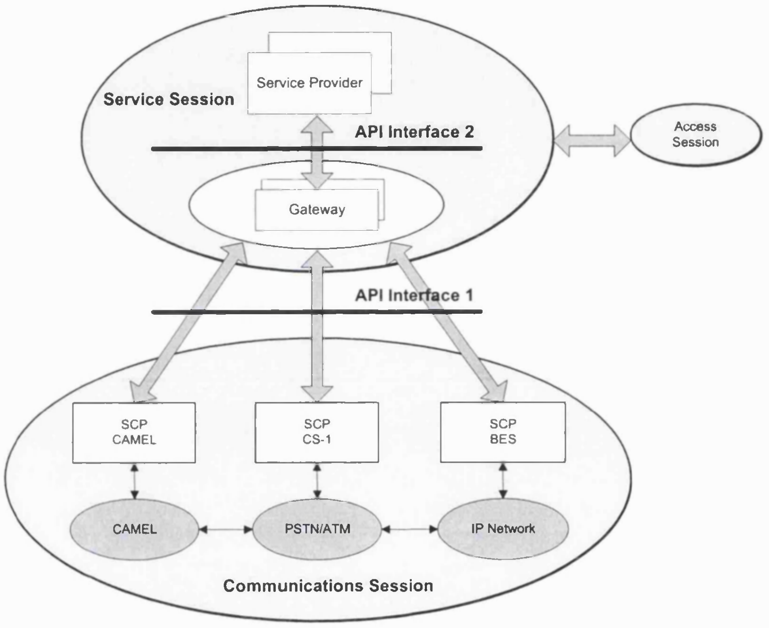


Figure 3.24: *SCPs responsible for the communication session*

The two API interfaces represent the work discussed in section 3.3.2.4. Specifically, the two interfaces are the subject of the work of PINT, Parlay, and JAIN. Figure 3.24 shows the relationship between PINT, Parlay, and JAIN.

Although it is not very clear at this point the direction PINT will follow, it seems to be one of functional decomposition of the required blocks in order to achieve the initiation of PSTN services from the IP domain. It is the author's view that taking an object-oriented approach to the development of PINT services can result in an easier integration between these and the TINA architecture, specifically the DPE. By placing the DPE as a means of allowing access to the PINT gateway it provides a number of enhancements to the basic PINT services [Solo00a]. This for example could allow a number of service providers to make use of the gateway and provide customised solutions. PINT provides an interface with the SCP that can be controlled by a service provider. It provides a connection between the service session and the communication session.

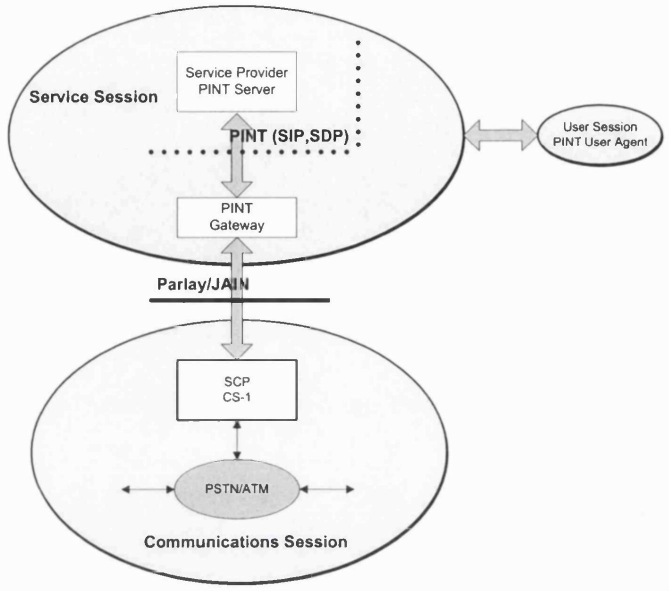


Figure 3.25: *The TINA service architecture, PINT, Parlay, and JAIN*

Furthermore, the use of APIs to open up the interface between the communication session and the service session (chapter 8) is an important step forward. It allows a service provider to write applications that can incorporate features as necessary [Solo00a][ParlBuss99]. In the case of Parlay, the service creator need only implement the required software interface for the underlying capability to be made available. This opens up important possibilities for service development. Firstly, by making network resources available as required, service creation may be viewed as a Network Capabilities Service Creation Environment [Solo00b]. As new network capabilities become available, they can be made available as class libraries. As such, it is not necessary to standardise on information flows as in the case of IN CS-1 and IN CS-2. It is necessary only to standardise on the description of the API, as is the case for Parlay.

Convergence requires the inter-working of services across the different network types or different operator domains. It is not yet clear whether the API approach will provide a solution to the problem or even scale across multiple domains. However the work presented in [Solo00b] [Solo00c] and [Solo00d] points to an approach where a second set of APIs or even overlaid state models be made available to provide service providers with a higher level of service interaction. This issue is further examined in chapter 8.

The following section discusses the role of state machines in a converging environment, and builds upon the views presented in section 2.6 where the role of state machines in traditional telecommunication networks was discussed.

3.5 STATE MACHINES IN CONVERGING NETWORKS AND THE ROLE OF IN

Chapter 2 examined the state models and control plane of the PSTN. As part of the convergence process, the state models on the IP side need to inter-work with the less well-structured models of the IP network.[*]

IP as a connectionless service apparently has no need for a basic call state model. In fact, there is no state information stored in traditional IP routers.[†] In general, IP routers are designed to achieve high packet throughput with little or no attention to the service the packet is transferring[‡]. This makes it unnecessary for an SCP-like architecture to support the connection process. In addition, many of the CS-1 services (e.g. call screening) supported by the SCP are better placed on the edge of the IP network and handled by intelligent terminals.

However, through the discussion of the taxonomy, it was suggested (section 3.3.2) that the state models that are available in the IP world need to inter-work with the IN. Section 3.3.2.4.3 identified that there are limitations to this. It is the author's view that the well-defined state models of the IN world are unlikely to be relaxed in any way to support the convergence process. Certainly, there has not been an indication that this may be the case as such a move could compromise the integrity of the PSTN, the major carrier [ITU97][ITU00] for a large proportion of IP networks.

The approach that is adopted to provide such inter-operability is one whereby additional functional entities (such as gateways) are introduced in the IN domain to support the translation functions. Such functions were discussed in section 3.3.2.

Within the context of the converging environment, there may be reluctance to adopt IN principles in the IP domain. Contrary to this, there are a number of areas where IN is likely to play a role. Although such services may be provided with architectures that do not adhere to the

[*] The arguments presented in sections 3.2 and 3.3.2.2 indicate that this is the case.

[†] Some routers offer packet discrimination services that involve maintaining state information. Such routers are generally deployed in dedicated networks that are designed with QoS in mind [Adis98].

[‡] In fact the datapath functions of the router that are performed on every datagram that passes through the router, are often implemented in special purpose hardware. In trying to improve the per-packet performance of a router [Part96], the datapath functions are optimised using parallel processing and special-purpose hardware [McKe00][Wang01]

IN standards the author's viewpoint is that there are good reasons why IN should be adopted in the IP environment.

The first of these is to do with the support of QoS sensitive services. For example, RSVP [RFC2205] and Diffserv [RFC2475] define state machines for the establishment of resources on behalf of the user. If a service session is being defined and the streams for the session have to be controlled within the communication session then there is a role for an SCP [Solo00a]. In addition, in section 3.4, it was argued that a combination of the Gateway and the BES effectively performs the function of the SCP and that other services may be established on behalf of the end user through the "SCP."

The second issue is to do with the requirements for Universal Personal Telecommunications (UPT) [UPT]. UPT requires that a service should be made available to any user anywhere within the network [ETSI ETR 055-1]. In the short term, this means allowing registration of the user at any terminal on any network and allowing the user's portfolio of services to be made available subject to the terminal capabilities or other specialised resources. For example, a registered user may be contacted from and connected to any of the other networks for which the user has a subscription. This will require the interchange of user data between networks of the same type (i.e. PTSN–PSTN) and different types (PSTN–IP). The requirement for the IP case is that the same rules apply. Whether one calls the device a BES or an SCP, it still needs to exist in the IP [Solo00a].

What can be extracted from this section is that the traditional role of IN is undergoing an evolutionary change as a result of the convergence process. The new role of IN is one where it should be viewed as a control architecture that enables the convergence of PSTN and IP networks.

3.6 Chapter Summary and Research Contributions

As noted in the introduction to this chapter, there is a clear transition from single service networks to integrated service networks. This entails the inter-working of networks to achieve a level of inter-operability which a few years ago was unheard of, as it did not fit within the regulatory telecommunications environment.

The inter-operability and inter-working of these networks is a requirement imposed by the service characteristics. Different services have different transport requirements. The transport capabilities of networks and available network resources have advanced over the last decades allowing a convergence of fixed telephony, data transmission, multimedia and mobile services. One of the driving forces behind this integrated network has been the market deregulation, technological advances and of course the availability of funds within the telecommunications environment (such as the well-publicised funds spent towards acquiring 3G licences).

Within this framework, an aggregation of the technological, market and evolutionary issues surrounding the convergence process and its impact on the control plane has been provided.

A large part of the work presented focused on the new role of the IN control architecture. This was presented using the taxonomy reference model. The author believes the model is a useful way to understand the interactions of the architectures participating in the convergence process. At a high level, the model can be viewed as presenting the two extremes towards network intelligence. On the one side control is provided to the PSTN through the IN and, on the other, through the more relaxed approach of the IP domain. These two architectures must inter-work in order to provide the services end users have come to expect regardless of the underlying network. For this reason, architectures such as PINT are emerging to support such hybrid services. To achieve this the inter-operability requires inter-working of the state models.

The state models for JTAPI, CTI and JCC have been discussed and it has been determined that although JTAPI offers the programmer clear and explicit abstractions for manipulating calls and the logical entities in a call in an object-oriented manner, it lacks certain desirable functionality, such as the single association of call legs.

In section 3.3.2.3, the JTAPI Call model was presented. It has been argued [Jain00] that JTAPI overcomes several of the limitations of the IN. According to the same reference, in the IN world "there is no explicit abstraction offered to allow the programmer to manipulate entire calls, or legs of a call, or the principal logical entities in the call (e.g. the calling or called party's address), and certainly not in any object-oriented fashion." But the author has pointed out that JTAPI does not currently capture all states that the IN model does and in JTAPI there is no concept of triggers or detection points [JTAPI]. Moreover, it is impossible to suspend call processing at a defined state in the FSM, invoke an application (supplementary service logic), and return results, although some improvements are suggested by JCC (section 3.3.2.2).

Another application that was discussed within the context of the taxonomy is the utilisation of IN infrastructure to support billing, authentication, and payment systems within the area of electronic commerce. This application is the focus of the work that is presented in chapter 4 and chapter 5.

CHAPTER 4

UTILISING EXISTING IN INFRASTRUCTURE FOR IP-INITIATED BILLING & ELECTRONIC PAYMENT SYSTEMS

The previous chapter identified that legacy IN control elements can be used to provide authentication, billing and certification services for electronic commerce. This chapter discusses the design of a system for electronic payments that is based on the capabilities of IN CS-1.

4.1 INTRODUCTION

The system that is described in this chapter utilises existing IN CS-1 infrastructure and capabilities to enable home users to engage in electronic commerce. At a functional level, the Intelligent Electronic Payment System (IEPS) provides two important capabilities. Firstly, it provides authentication capabilities for both end users and shops and, secondly, it performs the billing through the existing PSTN connection of the user. Through the IEPS, users can charge for **micro-payments** on their telephone bills [Solo98][Solo99a][IEPSPat].

The operation of the IEPS is based, for the most part, on existing IN CS-1 functional entities and information flows. The additional component that is required to provide the interface between the IN and IP worlds is a gateway.

Section 4.2 looks at existing electronic payment systems, and identifies some of their limitations. Section 4.3 provides an overview of their desirable characteristics of electronic payment systems. Following this, section 4.4 provides a description of the proposed system and section 4.5 describes the operation of the protocol. Section 4.6 briefly describes the security features of the IEPS. Section 4.7 evaluates the system as an electronic payment system by comparing its operation to the desirable characteristics of electronic payment systems presented in section 4.2. Finally sections 4.8 and 4.9 examine the inter-operability between the IEPS and the capabilities provided by the IN CS-1. Section 4.10 provides a summary of the chapter.

4.2 Existing Electronic Payment Systems

In the past, most European and US operators have invested heavily in IN platforms to support key bearer-type services such as Freephone and Premium rate. Such systems have complex structures for interfacing with existing billing systems that are resilient and trusted* by the end users. In contrast, there is still much mistrust over the security of the Internet. This is indicated by market research and polls regarding online shopping as well as by the Bank of International Settlements in the 1996 report [BIS96].

In the past decade, rapid advances in communications, electronic service networks, multimedia and interactivity have been opening up new opportunities for business. They are contributing to new and effective ways of disseminating information, promoting products and services and, more recently, carrying out electronic business transactions, both in the business-to-business and business-to-consumer sectors [Kapr01].

The Internet is one example of a communications network that is transitioning from an inexpensive medium for advertising, marketing, and customer support to a common platform for transactions and business applications [ITU00]. At the same time, technological and commercial developments are melding together information, communications, commerce, and entertainment into one large, consolidated industry. Part of the reason for this evolution is that more consumers are accessing the Internet using multiple devices and over multiple communications networks [Amar01].

Electronic payment systems, first introduced in the late 1990s, such as ecash™, cybercash™ and barclayCoin™ require the use of a credit card.† At the same time however, there is a reluctance by individuals to divulge their credit card details online, despite the efforts made to promote such usage [WResInc98].‡ In any case, this framework is limiting, both for end users and for businesses. The reasons for this are discussed in the following section.

* Not many people are known to individually check an itemised phone bill.

† While ecash and cybercash have now evolved into a more mature system, barclayCoin has been completely dropped by Barclays. At the time of its development, only 12 retailers signed up to accept it as a form of online payment.

‡ In cases where fraud is reported, the issuing bank for the credit card provides refunds quickly, thus encouraging confidence among end users. An analysis of the legal aspects of electronic commerce and security can be found in [Baum92].

4.2.1 Limitations with Traditional Payment Systems

The term "traditional" in the context of this section refers to payment systems that were not designed for use on the Internet or for electronic payments. Examples of these include cheques and credit cards. To the lay person, credit cards may appear to be quite efficient for online commerce. It is however, a misconception to accept credit cards as a system designed specifically for online payments – after all, credit cards* were in existence long before online commerce.

These traditional payment methods are inadequate for real-time payment interaction [Furc96]. Here, "real-time" means a transaction is triggered and committed when the consumer hits the "proceed" button on a Web page. Even with credit cards, a number of systems require the intervention of personnel to authorise the transaction. This incurs overhead costs as well as inconvenience to the end user. Furthermore, a large number of traditional payment methods generally require that the consumer leaves the online platform and uses the telephone or sends a cheque in order to make a payment. This is particularly true with small companies, which may not accept payment by credit card, or where the transaction value is too small to use one. It may also be expensive in terms of time to enter the credit card numbers [Furc96][Kala97]. Some companies continue to require users to send information offline and arguably, these companies have not been very successful.

One of the reasons for requiring the user to submit his payment details offline is the lack of security of the system. It is widely accepted that most users do not feel safe when submitting their credit card information over the Internet [DOC00] and, furthermore, that the most important factor in promoting electronic commerce is trust [OECD97b].

In [OECD97a], the Organisation for Economic Co-operation and Development identifies that "trust is central to any commercial transaction" and that "typically it is generated through relationships between transacting parties" and "familiarity with procedures." The report furthermore identifies that "some observers fear that unless action is taken very soon to bolster trust in electronic commerce, it will never assume its place as an important channel for commerce." However [OECD97b] continues by saying that "such fears [of fraud] are exaggerated and that in time as consumers become more familiar with the technologies, [they] will gain confidence in electronic commerce." The author agrees that, for non-technical users,

* According to Encyclopaedia Britannica, the first universal credit card, one that could be used at a variety of stores and businesses, was introduced by Diners Club, Inc., in 1950.

familiarisation with procedures will overcome a large obstacle in the growth of electronic commerce.

However, the author takes the view that there is a different notion to trust. Trust can be viewed as a trade-off between actions (desire), consequences, and risk. For instance, if an end user is potentially risking fraudulent use of their credit card account by shopping online for a service of low intrinsic value, then that is a very high risk.

A further characteristic that adds to the limitations of any payment system is its coverage. This is defined as the ability of a store to accept credit cards as a form of payment. This makes credit cards applicable only with signed-up shops. Furthermore, this form of payment does not generally support consumer-to-consumer or direct business-to-business payment transactions. Therefore, the payment method is limited in its acceptability.

The coverage and acceptability characteristics are also compounded by the eligibility of end users to qualify for a credit card. Not all potential buyers have suitable credit ratings to allow them access to credit cards. It therefore follows that both acceptability and coverage are hindered in these groups.

A major disadvantage of credit-card-based payment methods is the lack of support for micro-transactions. Micro-transactions are defined as transactions that are of low intrinsic value. Many payments made over the Internet are of lower value than the cost of a phone call or even a letter (for sending payment information). What is more, the time it takes to type the required information may be too high an overhead. The cost of handling these payment methods is often too high for the seller to break even. This is a very important limitation. Credit card companies usually charge commission for every transaction they process – and very often impose a minimum transaction charge. This automatically rules out the use of credit cards for micro-transactions. Even if credit-card fees are brushed aside, it is still impractical for the user to enter all the credit-card information for such small transactions. For example, single database-type applications could charge the user 10 pence for every successful match in the lookup process, a pay-per-click approach. The Internet (both fixed and wireless) presents a convenient platform for micro-payments and the IN infrastructure (with billing systems in place) is strategically positioned to provide support for such transactions.

4.3 Desirable Characteristics of Electronic Payment Systems

Having identified the limitations of traditional payment systems, the characteristics that are desirable for Electronic Payment Systems (EPS) are next presented. A complete discussion of these can be found in [Maho01][Kala97][Furc96][Okam93][Essi92].

4.3.1 Security and Data Transmission

One of the most important requirements is to maintain the security of the system. Protection against various forms of fraud, the generation of non-existent funds or the malicious use of lost or stolen cards, are central issues in making payment systems viable. Different infrastructures behind payment schemes require different kinds of protection mechanism. Most systems need a user authentication mechanism or access control system. This is often implemented using a personal identification number (PIN).

The security aspect extends to the data transmission [BIS96]. This is extremely important as, in most cases (particularly for Internet transactions), a transaction is transmitted for remote processing. Such transactions can be intercepted, which can lead to unauthorised use of the payment system [Baum92][Basle99]. At a high level, there are two approaches which could be used for dealing with this problem.

The first involves the isolation of the transmission infrastructure. This concept requires the setting up of isolated networks to be used for financial transactions processing. The scheme is secure but expensive, as it requires setting up a complete network infrastructure, as well as protecting it from intruders (which can prove to be the most costly part).

A second approach is the use of encryption in order to secure transactions (data transmitted as part of a transaction record). This scheme allows the use of public communication networks to transmit financial transaction data. A number of techniques can be applied to encrypt data transmitted over public networks. One of the most commonly used is that of public-key encryption and is implemented by the RSA[*] algorithm [Rive83][RSA98].

4.3.2 Authentication

The second important characteristic deals with authentication and proof of identities [Furc96]. In electronic payment schemes, it is often necessary to restrict user access, for example, to the

[*] RSA is named after its inventors, R. Rivest, A. Shamir, L. Adleman.

owner of an electronic wallet* and to verify the identity of a particular user, to ensure that the person to whom a payment is made is, in fact, who they identify themselves as. Different schemes of identification exist, the most popular of which are PIN numbers, passwords and personal hand-written signatures. Hand-written signatures, although not really an electronic authentication scheme, are included as they are used to secure credit card transactions as well as being the original model for authentication schemes. There is a problem however with personal signatures – that of verification. It takes an expert to tell a forged signature from an authentic one and therefore it does not offer protection to the credit card user from the misuse of a stolen card; however, people have come to accept this authentication mechanism. Similarly, **digital signatures** allow parties transmitting information over an insecure channel to sign the transmission digitally so that these signatures can be used in the same way as hand-written signatures are used in paper documents [Curr01].

Digital signatures are used for authentication of the sender, as well as maintaining the integrity (here meaning that the data has not been altered by a third party) of the data throughout the transmission [Kala97]. This also means that the originator cannot falsely deny having signed the data. In addition, a digital signature enables the computer to "counter-sign" the message, assuring the recipient that the message has not been forged in transit.

Digital signatures ensure authentication by combining the data to be transmitted with a private key. The user's private key is combined with the document and performs a computation on the composite (key + document) in order to generate a unique number called the **digital signature**. Digital signatures must exhibit the following characteristics [ABA01][Furc96]:

- Verifiable – Anybody should be able to validate a signature.
- Unforgeable – It should be impossible for anybody but the issuer to attach the issuer's signature to a document.
- Non-reusable – It should be impossible to "lift" a signature off one document and attach it to another.
- Unalterable – It should be impossible for anybody to change the document after it has been signed.
- Non-deniable – The person signing the signature cannot deny having done so.

The process involved in digitally signing a message requires the sender to apply his private key to the data to be transmitted. To increase the speed of the process, the private key is applied to a

* "Electronic wallet" here refers to a collection of data that is stored on the user's machine and contains information regarding bank details.

shorter form of the data, called a "message digest" rather than to the entire set of data. The resulting digital signature can be stored or transmitted along with the data. The signature can be verified by any party using the public key of the signer. This feature is very useful, for example, when distributing signed copies of virus-free software. Any recipient can verify that the program remains virus-free. If the signature verifies properly, then the verifier has confidence that the data was not modified after being signed and that the owner of the public key was the signer.

For example, when an electronic document, such as an order form with a credit card number, is run through the digital signature process, the output is a unique "fingerprint" of the document. This "fingerprint" is attached to the original message and further encrypted with the signer's private key. If a user is communicating with her bank, she sends the result of the second encryption to her bank. The bank then decrypts the document using her public key and checks to see if the enclosed message was tampered with by a third party. To verify the signature, the bank performs a computation involving the original document, the purported digital signature and the customer's public key. If the results of the computation generate a matching "fingerprint" of the document, the digital signature is verified as genuine; otherwise, the signature may be fraudulent or the message may have been altered.

4.3.3 Transaction Cost and Use of Additional Hardware

Transaction cost can be defined as both the time required for a transaction and the financial expense associated with processing overheads, hardware costs and other financial expenses [Essi92]. Transaction cost determines the amount of money the user of a system is effectively charged for a purchase on top of the actual sale price. For instance, high financial transaction costs make it uneconomical to use a system for small financial transactions (e.g., the writing of a cheque for the amount of five pence) and long processing times of transactions can make it inconvenient to use on a particular system.

Transaction cost and the use of additional hardware are linked*. In most cases, an EPS requires that every payment is cleared online with a central database located at the authorisation organisation (e.g. a bank or a credit-card company). If a decision about the validity of a payment can be made locally without the need for online clearance by a central authority, then both the transaction cost and processing time decrease. Such systems however require additional tamperproof hardware [Bran93]. Prepaid smart card payment systems usually do not perform online verification, as the hardware is assumed to protect the information stored on the cards.

* For example, in [VISA] there are minimum specifications for VISA card readers.

Requiring online verification on the Internet is less of a problem, as the communication infrastructure is already in place. An offline verification system would be advantageous only if no online connection to the central authority can be established. Thus, the online verification requirements of an EPS could produce an unwanted overhead.

Transaction cost can be divided into three categories – high, medium, and low [Essi92]. An EPS is said to have a high transaction cost if any part of the transaction has to be processed manually. Automated transactions that cause a substantial overhead are not quite as expensive as those involving manual transactions and they are also faster. For example, SWIFT [SWIFT] is an example of this level of transaction cost. These systems are cheaper and faster to use, but still involve substantially higher costs than the use of cash. Finally, a system with low transaction cost eliminates the need for online clearance and (by definition) reduces the hardware and communication expenses. An example of this level of transaction cost are pre-paid phone card systems. There is no online clearance (from a financial authority) as they are prepaid.

4.3.4 Traceability of Payments

Traceability of payments involves tracking a transaction without compromising or revealing the details of a user. Common electronic payment systems, such as credit cards, generate a record for every payment made. This is needed to ensure that all payments can be verified. It is now possible to devise payment systems that do not allow payments to be traced without compromising the system's security standards. This permits the implementation of systems that are cash-like in that they ensure some limited anonymity of payments. At the other extreme, an anonymous credit card system was proposed by [Okam89].

In general, payment systems can be grouped into four categories. These are Unconditional, Conditional, Untraceable, and User-controlled:

- Unconditionally Traceable

  Payments in a system are unconditionally traceable if a transaction generates a record that identifies buyer, seller, amount, date and time, and optionally some additional information. This allows the bank, or another party obtaining the bank's records, to trace all payments made within the system. This is the way in which today's credit-card payment system operates. There are however a number of other alternatives which provide some form of anonymity.
- Conditionally Traceable

  A conditionally traceable system is one in which payments are generally anonymous but allow for the identification of transaction details by obtaining what is referred to as a "reference transaction". According to [Furc96] "conditional traceability provides for a somewhat higher level of privacy and anonymity than unconditional traceability, as it requires some action to 'de-anonymise' the transactions, and this will not always be done (i.e. these systems will not be used to obtain marketing data or for personalised loyalty schemes)."

- Untraceable Payment Systems

  These allow the payer to remain completely anonymous. These systems have to mimic the properties of cash as far as possible. In [Chau82], [Chau83] and [Chau92] the author demonstrates that, by using blind signatures*, such systems can be implemented.
- User-Controlled Traceability

  In such systems, every payment generates its own receipt together with the payment in encrypted form. Only the user making the payment owns the key to the encrypted receipt (and can selectively make that known). Commercially implemented systems which provide user-controlled traceability include DigiCash's ecash system. It could be argued that these systems from the payer's perspective are the most powerful. This is because in terms of privacy the payer decides on who can access his identity.

4.3.5 Acceptability and Transferability

This is the property of a payment system to be accepted universally, that is, its acceptance is not limited to one bank or organisation issuing the system [Maho01]. The acceptability property is easy to add to any system. Upon receiving a payment of some form that was issued by another bank, the bank processing the payment clears it with the issuing bank. Acceptability is therefore achieved at the expense of higher communication overhead and therefore higher transaction cost. It is more difficult to achieve acceptability in pre-paid services.

Transferability is concerned with the ability of users of the system to transfer funds to each other without the need to contact the bank for clearance of the transaction. Transferability is difficult to implement without compromising the system's security [Furc96].

4.3.6 Implementation Issues

The final characteristic of an EPS is the trade-off between providing a software-only solution for its implementation versus tamperproof hardware. Systems use a software-only solution when they can be implemented in such a way that all data and communication on the user side is accessible by the user. In such a system, the user must not be able to obtain any benefits from

* Blind signatures are a variation of digital signatures, where the sender multiplies the digital signature by a random number [Chau92].

tampering with data or communication. Systems employing tamperproof hardware use additional hardware that is assumed to be designed in such a way that it protects the information it contains from the user of the device. For example, not even the holder of a smart card can access or modify directly the information stored on the smart card. Generally, according to [Furc96], a software-only solution that does not require additional hardware is considered superior to a solution that requires additional specific hardware to be employed.

4.4 THE IEPS IN-BASED BILLING SYSTEM

This section discusses how an existing IN infrastructure can enable support for online secure transactions. It focuses on the design and operation of the Intelligent Electronic Payment System (IEPS). It also identifies inter-operability issues with existing IN CS-1 infrastructure.

4.4.1 Overview of the IEPS Model

Figure 4.1 depicts the parties involved in the IEPS. These are the Customer (user), the Internet Shop (IS), and the Network Operator.

In figure 4.1, the Service Provider is a public or private company that develops and provides IN services commercially over the common IN-structured network and underlying basic (bearer) services. A Network Operator who utilises an IN CS-1/2 compliant infrastructure can become a service provider of the IEPS. The software required for this is defined as the Network Operator's Agent Software, NO Agent.

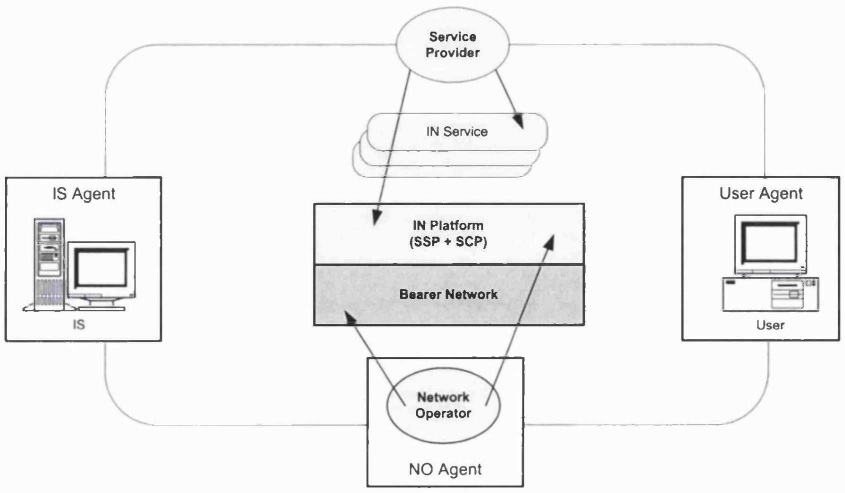


Figure 4.1: *Parties involved in IEPS framework*

The Service Subscriber is an organisation that obtains an IN service from a service provider on a contractual basis and has to pay the charges to that service provider. In the context of this protocol, the service subscriber is the Internet Shop (IS). The software that is running on the IS's equipment is defined as the IS Agent.

Similarly, the person who has access to and makes use of a service, i.e. represents the called or calling party depending on the type of IN service, will not necessarily be the service subscriber. This person must belong to the subscribed users of the service provider. The software that is running on the user's machine is defined as the UA.

As previously discussed, the proposed IEPS protocol makes use of the fact that large sections of the Internet user community rely on home access to the Internet using modems. As a result, access to Internet Service Providers (ISPs) for most homes involves the use of dial-up services via connections through the PSTN. ISPs for the most part rely on leased line services that are part of a core public telecommunications infrastructure.

The proposed system provides the subscriber with a **secure link** through which transactions can be made. The Network Operator can be viewed as both a credit-card authority and a telecom operator. This fits nicely with the expectation of users for a "one-stop shopping" solution. The following section analyses the network elements of the IEPS.

4.4.2 Elements of the IEPS

This section provides a brief overview of the proposed IEPS. Figure 4.2 depicts physical interconnections between the parties involved in the IEPS.

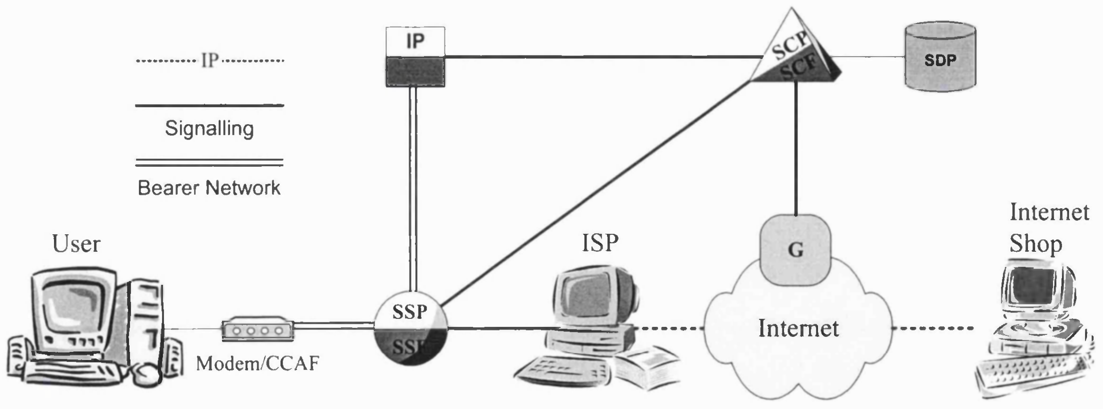


Figure 4.2: *IEPS network elements*

There are three distinct classes of connection. Firstly, there is the bearer connection between the User and the Intelligent Peripheral (IP) through the local exchange and the Service Switching Point (SSP). Secondly, there are the signalling connections between the various functional entities of the IN CS-1/2 infrastructure. An important point here is that the interconnection between the Gateway (G) and the Service Control Point (SCP) is also treated as a **standardised**

signalling connection. Finally, there are two connections which are based on IP: one is from the ISP to the Internet and the other is from the Internet Shop (IS) to the Internet. To clarify a point, the IS, in most cases, will have to go through an ISP but this is not depicted, since it does not interfere with the operation of the protocol.

4.5 THE IEPS PROTOCOL

This section discusses the protocol for the IEPS. Before discussing the technical details of the protocol, a description of the operation of the protocol is given, in a concise manner, avoiding many of the implementation details.

The protocol is divided into three phases. Firstly, the user connects to the ISP; then the user exchanges IP packets with the Internet Shop and the Gateway; finally, the Gateway completes the transaction using the SCP.

The User connects to the Internet using a PC and, in the large majority of cases, a modem, by connecting to an ISP. To the IN model, the modem represents a Call Control Agent Function (CCAF). The Switching Exchange, where the SSP usually resides, is a switching centre of the Network Operator. It routes the call of the Internet Connection from the customer (user) to the ISP. The IP is the physical entity responsible for the implementation of the SRF and is controlled by the SCP. The SCP is responsible for payment-related processing. It includes database access, information validation and verification as well as any transaction-processing related procedures. Attached to the SCP is an SDP, which is a database containing information about customers who are registered as service users. Information stored includes access passwords, credit limits and customer preferences. This information is made available to the SCP for payment-related processing. Finally, the Gateway provides the connectivity as well as the translation functions for the inter-operability between the IP and IN worlds. The Gateway is discussed in detail in section 4.9.

The following sections provide a detailed description of the operation of the protocol.

4.5.1 Phase 1: User Connects to ISP

Figure 4.3 depicts the initial phase of the protocol, which takes place while the user is connecting to the ISP.

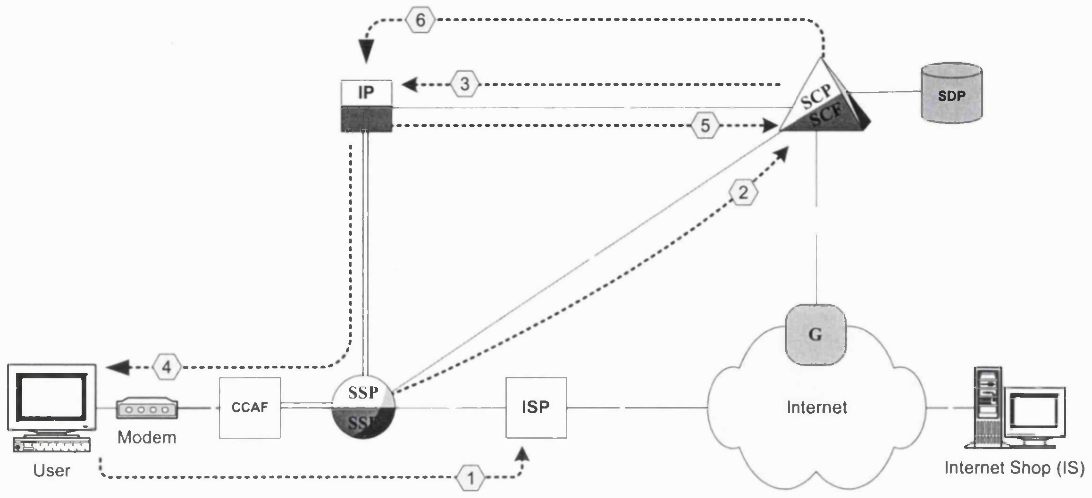


Figure 4.3: *Registration phase of IEPS*

This phase of the protocol is initiated when the user requests a dial-up connection from the PC to the ISP (stage 1). One convenient feature of the protocol arises from the access numbers the vast majority of ISPs assign to their Points of Presence (PoP). Usually, they provide local-call numbers (or increasingly freephone), which will trigger an IN transaction to interpret the number (stage 2). At this point, the service logic for the protocol will be associated with the standard number translation logic within the SCF. As a result, the SCF will also request from the SRF/IP (stage 3) to ask the user to enter a password or PIN associated with the physical connection (telephone line) the user is connecting from (stage 4). The SRF will collect the user's PIN and return it to the SCF for further processing (stage 5).

A point that needs to be addressed is the security of the PIN. Specifically, the PIN is not encrypted; however it is not possible for an intruder to pick up the information transmitted either from the SCF to the Specialised Resource Function (SRF) or from the SRF to the SCF, unless the line is physically tapped. Even if an intruder does collect a user's PIN the only way it can be used is if the intruder connects to the SCP from the telephone number associated with that PIN. It would be irrational for someone to physically tap a telephone line to collect a PIN and then break into the house in order to use a PC for e-commerce, and even more illogical to do so for micro-payments!

Following stage 5, the SCF must now match the PIN received with the PIN stored in the Service Data Point (SDP) and the telephone number of that user. On a successful match, the SCF generates a `ConnectionNumber` and passes it to the SRF using the `AuthorisationCode` information flow. If the match fails, the following options are available:

- The user is asked by the SRF to re-enter the PIN, or
- The user is not issued with a unique `ConnectionNumber`, which is required to make any purchases (further discussed later), or
- The SCF asks the SSF to terminate the connection.

These options can easily be re-configured in the SCF. An important parameter to the protocol is the `ConnectionNumber`. This is a unique number generated by the SCF, which ties the user's PIN and user details with the active connection. It is required by the protocol in the final phase (billing), where the information is cross-checked for validity. To avoid the likelihood of the same `ConnectionNumber` being assigned to two distinct users, the connection number is made a function of three parameters as follows:

$$ConnectionNumber = f(userPIN, telephoneNumber, randomNumber)$$

At this point, the user has connected to the ISP and, most importantly, the UA holds a valid `ConnectionNumber`. The next section discusses the exchange of IP packets.

4.5.2 Phase 2: User Exchanges IP Packets

Figure 4.4 shows the packets that are exchanged after a user has visited a web page and selected to make a purchase.

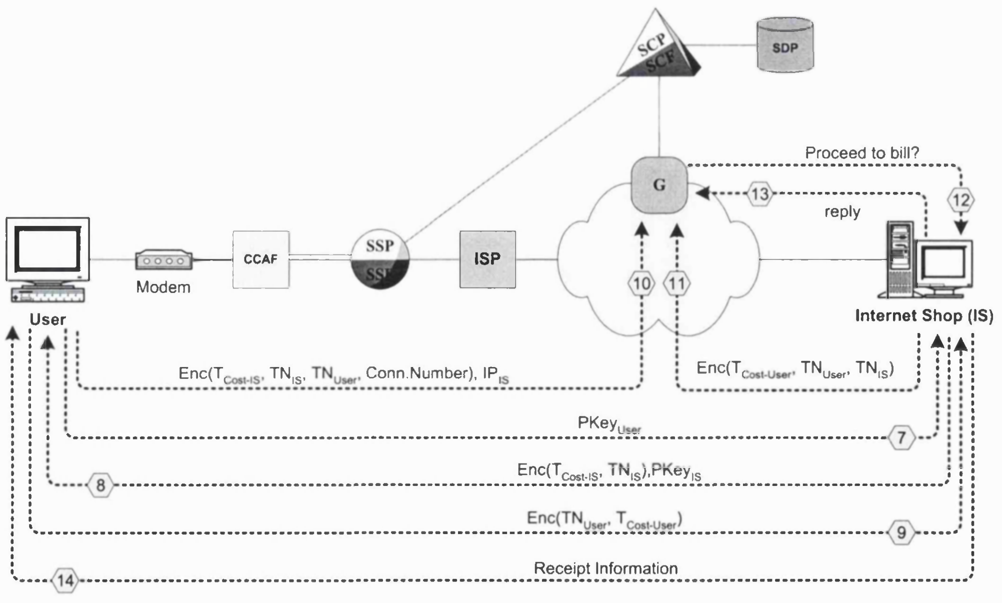

SCP
SCF
SDP
Proceed to bill?
G
13
reply
12
SSP
SSF
ISP
CCAF
Modem
User
10
11
Internet Shop (IS)
Enc($T_{Cost-IS}$, TN_{IS}, TN_{User}, Conn.Number), IP_{IS}
Enc($T_{Cost-User}$, TN_{User}, TN_{IS})
$PKey_{User}$
7
Enc($T_{Cost-IS}$, TN_{IS}),$PKey_{IS}$
8
Enc(TN_{User}, $T_{Cost-User}$)
9
Receipt Information
14

Figure 4.4: *The IP packets of the IEPS*

The IS sends a request to the user for his public key. The User Agent (UA) replies by sending the user's public key.

In stage 8, the IS Agent sends the following packet to the User Agent.

$$\mathrm{Enc}(T_{Cost-IS}, TN_{IS}),\ PKEY_{IS}$$

The packet contains the following information:

Encrypted using the User's public key obtained from stage 5.	Name	$T_{Cost-IS}$
	Description	Transaction cost as calculated by the IS Agent.
	Name	TN_{IS}
	Description	Transaction number generated by the IS Agent.
No encryption	Name	$PKEY_{IS}$
	Description	The public key of the IS.

***Table 4.1:** Packet structure for message 8*

The User Agent must accept the packet. This can be achieved in the following ways:

- The UA can wait while it is polling the incoming data for a special message, after which it saves the incoming data for decryption, or
- The UA can use Java Servlets [JAVA] to accept the incoming packet and write it to disk.

Whichever way is chosen, the data to be sent must be encrypted as it is sent over an insecure network such as the Internet. The encryption that is chosen is public key encryption. Authentication is achieved using digital signatures.

The IS Agent creates a record in a database, with the following record structure:

Field Name	Type
TransNumber	Numeric
TransCost	Numeric
TransDate	Date

***Table 4.2:** Record structure for IS agent*

In stage 9, the UA sends to the IS Agent the packet:

$$\mathrm{Enc}(T_{Cost-User},\ TN_{User})$$

Encrypted using IS's public key obtained from stage 6	Name	$T_{Cost-User}$
	Description	The Transaction cost reported by the IS Agent at stage 6.
	Name	TN_{User}
	Description	Transaction number generated by the UA

***Table 4.3:** Packet structure for message 9*

The UA uses the IS's public key to encrypt the $\mathrm{TN}_{\mathrm{User}}$ and the $\mathrm{TN}_{\mathrm{Cost}}$ as initially reported by the IS Agent. It then uses the $\mathrm{IP}_{\mathrm{IS}}$, received at stage 5, to send the information to the IS Agent.

The IS Agent accepts the packet and sends it to the Gateway, so that the necessary validation and verification can be performed.

In stage 10, the User Agent sends to the Gateway the following packet:

$$\mathtt{Enc}(\mathrm{T}_{\mathrm{Cost\text{-}IS}}, \mathrm{TN}_{\mathrm{IS}}, \mathrm{TN}_{\mathrm{User}}, \mathtt{Conn.Number}),\ \mathrm{IP}_{\mathrm{IS}}$$

Encrypted using the Network Operator's Public Key	Name	$\mathrm{T}_{\mathrm{Cost\text{-}IS}}$
	Description	Transaction Cost calculated by the IS Agent.
	Name	$\mathrm{TN}_{\mathrm{IS}}$
	Description	Transaction Number generated by the IS Agent.
	Name	$\mathrm{TN}_{\mathrm{User}}$
	Description	Transaction Number generated by the UA.
	Name	`Conn.Number`
	Description	The unique connection number assigned to the User by the NO Agent.
Not Encrypted	Name	$\mathrm{IP}_{\mathrm{IS}}$
	Description	The IP of the IS.

***Table 4.4:** Packet structure for message 10*

The User Agent also sends the $\mathrm{IP}_{\mathrm{IS}}$, in an unencrypted form as it is required by the gateway to contact the IS Agent.

In stage 11, the IS agent sends to the gateway the following packet:

$$\mathtt{Enc}(\mathrm{T}_{\mathrm{Cost\text{-}User}}, \mathrm{TN}_{\mathrm{User}}, \mathrm{TN}_{\mathrm{IS}})$$

Encrypted using the Network Operator's Public Key	Name	$\mathrm{T}_{\mathrm{Cost\text{-}User}}$
	Description	Transaction Cost returned to the IS by the User agent at stage 7.
	Name	$\mathrm{TN}_{\mathrm{User}}$
	Description	Transaction Number returned by the IS Agent at stage 7.
	Name	$\mathrm{TN}_{\mathrm{IS}}$
	Description	Transaction Number generated by the IS Agent

***Table 4.5:** Packet structure for message 11*

In stage 12, the gateway sends a final confirmation to the IS Agent in the form of an "ok to bill message" which contains the `TransactionNumber` and `Cost`.

In stage 13, the IS Agent processes the transaction number and the cost. If there is a pending transaction in its database that matches that transaction number and cost, it returns a true response, otherwise false.

4.5.3 Phase 3: Charging using the IN Gateway

This packet (stage 14) is sent independently of the Charge request to the SCP from the gateway. The IS Agent will confirm to the User the transaction number, receipt number (generated by the IS) and method of dispatch or delivery. This part of the protocol does not need to be defined, as different Internet Shops may choose different ways to send the Receipt Information, depending on the type of service/product that was purchased. However, the way in which the gateway can be implemented in discussed in section 4.8.

4.6 SECURITY WITHIN THE IEPS SYSTEM

The implications of security in a payment platform are of paramount importance. The IEPS Protocol is in an advantageous position because it utilises an existing billing system that, through time, has come to be accepted by end users as relatively secure. The basis for this is that few people individually check telephone bills in case of over-charging. However, since it is utilising an insecure network for Phase 2 of the protocol, security is still an essential ingredient in enabling electronic transactions.

Public-key technology is widely accepted as a qualified technology to meet the necessary security requirements for electronic business and it has become the preferred means for providing these capabilities [DOC94]. The benefits of public-key cryptography in relation to secure transactions are that it uses encryption to keep the information confidential and, through digital signatures, it provides for authentication, data integrity, and non-repudiation [Kali93a][Hale98][Baum94]. These techniques are combined to effectively “sign and seal” any electronic transaction and the signing can be done in such a way that the user who signed the information cannot later successfully deny signing that information.

The elements of a public-key infrastructure (PKI) are presented here. Further the section identifies how these relate to the IEPS System. The aim is not to give a detailed analysis of the workings of public key encryption, but rather to identify how it relates to the IEPS System. A detailed explanation of PKI can be found in [Aust00][Kali93a][Kali93b][RSA98].

4.6.1 Elements of a Public Key Infrastructure

The public-key infrastructure (PKI) is used to manage keys and certificates on behalf of users and applications. An important requirement of the PKI is to do this in a way that is transparent to end users. If it is not easy to use, people will not take advantage of its features. In addition to user transparency, the following are some of the key elements for a PKI:

- Certification Authority

  The certification authority (CA) is the trust centre of a PKI as it manages public key certificates for their whole life cycle. The CA is responsible for issuing certificates by binding the identity of the user to a public key with a digital signature and for scheduling the expiry date for these certificates.
- Registration Authority

  The Registration Authority (RA) provides the interface between the user and the CA. It captures and authenticates the identity of the users and submits the certificate request to the CA. The quality of this authentication process determines the level of trust that can be placed in the certificates.
- Certificate Distribution System

  Certificates can be distributed in a number of ways depending on the structure of the PKI environment, for example, by the users themselves, or through a directory service.

Other features include support for key backup and recovery. More essential is the support for non-repudiation of digital signatures. The automatic update of key pairs and certificates and the management of key histories are also desirable.

4.6.2 PKI and the IEPS System

As mentioned above, public key encryption requires two keys. The two keys are mathematically related so that data encrypted with one key can only be decrypted using the other.

Unlike secret key encryption, which uses a single key shared by two (or more) parties, public key encryption uses a pair of keys for each party. One of the two keys is public and the other is private. The public key can be made known to other parties; the private key must be kept confidential and must be known only to its owner. The parties involved in the IEPS will have their own public keys. These are denoted as follows:

- Network operator's public key denoted as $\texttt{Pkey}_{NO}$
- Internet shop's public key denoted as $\texttt{Pkey}_{IS}$
- Consumer's public key denoted as $\texttt{PKey}_{CO}$.

4.7 Evaluation of the IEPS Protocol as an EPS

Section 4.3 presented key desirable characteristics of electronic payment systems. This section evaluates the IEPS against those characteristics.

4.7.1 Software-only versus Tamperproof Hardware

The IEPS provides a software-only solution. This is true except for the IN/Internet Gateway that is an essential component to the IEPS. However, once the gateway is implemented it can be used to provide a wide number of integrated services.

4.7.2 System Security and Data Transmission

In terms of system security, the IEPS is classified as one that utilises an isolated infrastructure for the registration phase, as well as an unsecured transmission network for the second phase. Clearly, the registration phase of the protocol is in isolation from the Internet. However, unlike most systems utilising a network that is in isolation, it does not require the setting up of a complete network infrastructure (except from the SCP–Gateway link). Therefore, it makes use of existing links. Of course, these links must be maintained in order to protect them from intruders. For the second phase, encryption and security are clearly of paramount importance.

For the first phase, the system achieves authentication by means of a PIN, which is bound to the specific physical link of the isolated network. In order for the system to enhance the identification of Internet Shops in the second phase, the network operator could provide users with a customised browser that limits access to Internet Shops that are in agreement with the operator. This would provide the user with a more secure feel for the system than if a standard "open" browser is used. In any case, PKI is an essential component for ensuring the security of the second phase.

4.7.3 Transaction Cost

The system is classified as having a low transaction cost. This is because the online clearance is obtained by the Gateway after it requests an authorisation from the SCP. The SCP makes a database enquiry in the SDP to check, for example, the credit limit of the customer before it authorises the transaction. The protocol could also be enhanced to operate as a form of prepaid service, where the subscriber buys credit from the operator in advance and therefore further limits the need for online clearance.

4.7.4 Traceability of Payments

The protocol can be adapted to provide conditional traceability or user-controlled traceability. Obviously from the user's perspective to the Network Operator, the first phase of the protocol is classified as being unconditionally traceable. This is because the PIN number is associated with the physical connection. However, from the Internet Shop's perspective the system could be classified as offering a user-controlled traceability. If the user is buying a service, then the system is user-controlled. However, if the user is buying goods then the Internet Shop needs to know the delivery address of the user. Currently the protocol offers unconditional traceability in this respect (i.e. payer and payee are always identified). This can be changed and instead of the user providing the information to the Internet Shop, the information could be sent to the gateway and submitted to the Network Operator, who in turn arranges for the delivery of the goods.

4.7.5 Acceptability and Transferability

The acceptability of the system is limited only to Internet Shops that have signed an agreement with the Network Operator. However, different Network Operators can reach agreements amongst themselves therefore allowing a wider choice to the subscriber. Of course, this would require a higher communication overhead between the involved network operators and thus slightly increase the transaction cost of the system. At present, the system does not offer transferability of funds and there should be no need to do so.*

4.7.6 Comparison with Currently Available Systems

This system, unlike most currently available systems, does not require the user to open an account with the organisation providing the service. It plainly uses the account that is already in place (i.e. the phone bill). Unlike other systems, it does require the providing organisation (the network operator) to form alliances and reach an agreement with the shops. This could be a problem but, depending on the size of the organisation, it can be solved relatively easily. Because the organisation providing this system would almost certainly be the network operator, it should not be a major problem to reach agreements with Internet Shops. The bargaining power of the network operator would easily convince a smaller company to try their system.

A major advantage of this system in comparison with existing ones is that it does not use credit cards and hence completely avoids the existing public controversy about whether submitting credit card information over the Internet is safe or not. It could easily be marketed as a system that does not require credit cards because those systems are not secure.

4.8 IN CS-1 Information Flows for IEPS Registration

This section describes the capabilities of IN CS-1 (discussed in section 2.4.3.1) that can be used, unchanged, for the implementation of the IEPS. As previously identified, IN CS-1 capabilities are utilised in two phases: the registration phase and the billing phase. The IN CS-1 flows that

* Transferability is a desirable characteristic for electronic payment systems that are targeted at the business-to-business sector, rather than the business-to-consumer.

are needed for the registration phase are discussed in this section. The second phase requires a gateway for translation between IN and IP and this is discussed in section 4.9.

Figure 4.5 presents the information flows (IFs, refer to section 2.4.3.1) that are utilised by the protocol. It is important to make the distinction here that the IFs are physical connections. For example, at the point where the PIN number is collected from the user, the SSF is connected to the SRF (as indicated by the dashed grey line).

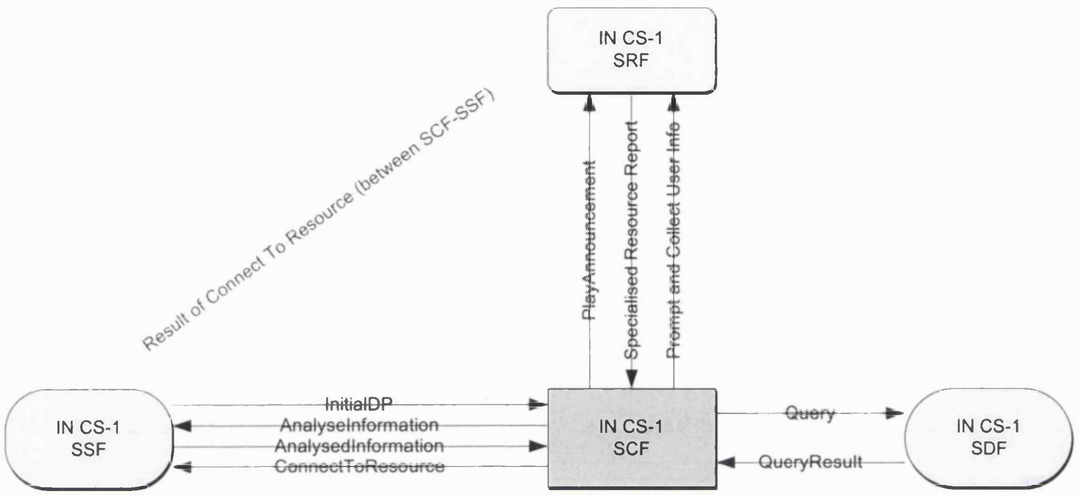


Figure 4.5: *IN CS-1 IFs for the registration phase of the IEPS protocol*

4.8.1 The SCF–SSF Interface

In IN CS-1 the SCF–SSF relationship is established either as a result of the SSF sending a request for instruction to the SCF, or at the request of the SCF for initiation of a call for some non-call related reason [Q.1214]. As previously discussed in section 2.4.1, information flows (IF) are used for the communication between IN functional entities.

One such IF is the **InitialDP** IF. This IF is generated by the SSF, when a trigger is detected at any DP in the BCSM, to request instructions from the SCF. One of the information elements of this IF is *Dialled digits,* which contains the actual digits received by the SSF from the calling party. It is used by the SCF to perform the number translation. When the number is translated, the call is re-directed to the ISP.

One of the IFs between the SCF and the SSF is the **Analyse Information** IF. This IF requests the SSF to perform the originating basic call-processing actions to analyse destination information, which is either collected from a calling party or provided by the SCF (e.g. for number translation). This includes actions to validate the destination information according to a specified dialling plan, and if valid, to determine call setup information (e.g. called party address, nature of address, and route index to a list of one or more outgoing trunk groups).

Although the **Connect to Resource** IF is depicted in figure 4.5 between the SCF and SSF, it is discussed in the next section because it is essentially a request to instruct the SSF to connect the user to the SRF.

4.8.2 The SCF–SRF Interface

This interface provides capabilities to authenticate the user using a Personal Identification Number (PIN).

In order for the SRF to be accessible to the user, the SCF must request the SSF to perform this connection. This is done by using the **Connect to Resource** IF, at the SCF–SSF interface. The information elements of the **Connect to Resource** IF include the *Call ID,* which identifies a specific instance of a relationship between an SCF and SSF.

After the SSF is connected to the SRF, the user is prompted with a welcome message. This is done using the **Play Announcement** IF. One of the information elements of the **Play Announcement** IF contains the *Inbandinfo* structure, which allows the SCF to specify an elementary message or text, the number of repetitions as well as the duration and the interval of the announcement. **Play announcement** can be used to welcome the user with an introductory message.

Following that, the **Prompt and Collect user information** IF is used to collect the information from the user. The information collected from the user is stored in the *Collected Info* information element that can contain either *Digits* or IA5 information (for collection of text from the user). The structure of *Digits* enables the definition of the minimum and maximum number of digits to be collected, timeouts and the way errors should be treated. In the case where an error (e.g. timeout) does occur, it can be handled in a number of ways including playing to the user a "help" message, repeating the prompting message, or sending the information collected to the SCF for further processing. An additional element of the *Digits* record is *VoiceInfo*, which indicates that digits may be collected using voice recognition.

Once the data is collected from the user, the SCF needs to match the entered PIN with the PIN of the user stored in the SDF. For this the SCF–SDF Interface is used.

4.8.3 The SCF–SDF Interface

The **Query IF** is used by the SCF to collect information from the SDF. The **Query IF** allows the collection of data from the SDF and is widely used in applications such as number translation and freephone numbers. Its information elements include *DatabaseID, RequestedInfoType* and

Information key. In the IEPS, the *Information Key* allows data to be retrieved from the database based on the calling line ID. More specifically, any data that is contained within the **InitialDP** can be used as an *Information key*. The response to the **Query IF** is the **Query Result IF**. The returned information element, *Result*, may contain either data or simply a true or false value.

4.8.4 Resuming Processing at the SSF

An important point is that throughout the time that the processes of translating the dialled number, connecting the user with the SRF, then collecting the user's PIN and matching the information entered with the information in the SDP are being carried out, the call is on hold at the SSF. Once these procedures are completed, the SSF is connected to the terminating exchange and the user has access to an IP connection – provided by the ISP.

Having discussed the IFs that are available for implementing the registration phase of the IEPS protocol, the next section moves on to introduce two potential architectures for the implementation of the gateway function, which allows the inter-connectivity between IN functional entities (FE) and the IP end-points of the IEPS.

4.9 The Gateway between IN and IP

An important functional entity of the protocol is the gateway. It provides seamless inter-operability across two fundamentally different networks. On the one side, the gateway must be treated as a standardised IN functional entity and, on the other, the gateway must be able to process IP packets according to the design of the protocol. Therefore, the gateway is essential in providing the interconnection between a circuit and a packet-switched network. At the same time there are some fundamental non-functional requirements imposed on the gateway. Essentially these are the capabilities of the underlying IN infrastructure; in this case IN CS-1.

The gateway device must be in a position to communicate interactively with the proposed application. This communication should minimise any potential changes that may be needed on the IN-side because a new system, or protocol, is being introduced into an already extremely well defined and standardised environment. It would not be feasible to expect already standardised systems to adapt to newly-developed systems. To achieve this, two potential functional entities that could be used to allow this interaction are examined: the Intelligent Peripheral and the SCF. To these, the gateway would appear as another standardised element of the existing IN infrastructure.

This section describes two possible options for implementing the gateway between the IN and IP worlds. The section focuses on the specific way in which it can be implemented for the IEPS protocol. The two proposed implementation options are to treat the gateway as an Intelligent Peripheral (SRF) or as an SSF. The difference between these cases is in the communication primitives which are used between the elements, which in turn extend or limit its capabilities.

4.9.1 The Gateway as an SRF

Under this approach, the SCP views the gateway as an Intelligent Peripheral. As a result, the gateway must adhere to the standard composition of an IN CS-1 SRF, whose main components include the Functional Entity Access Manager (FEAM) and the SRF Resource Manager (RM) [Q.1214].

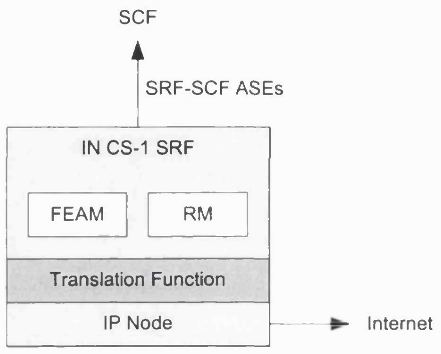


Figure 4.6: *The gateway as an SRF*

Figure 4.6 illustrates the SRF approach. The top part consists of a standardised IN CS-1 SRF. The translation function is responsible for converting IP packets to SRF–SCF IFs. While this approach can provide the necessary functionality, the second, where the gateway is treated as an SSF, is more powerful as it allows the triggering of SCP-based IN services.

4.9.2 The Gateway as an SSF

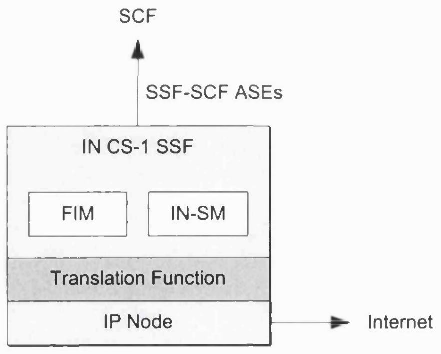


Figure 4.7: *The gateway as an SSF*

Figure 4.7 depicts the second approach, where the gateway is treated as an SSF from the point of view of the SCP whilst the IP side could be an application or even a router function. If the router QoS mechanism can be described as a state machine then this can potentially be used for triggering services in the same way that the BCSM can in the call control function. The CCF then effectively becomes an IP Control Function (IPCF). The SSF/Gateway now allows the triggering of SCP-based IN services and is much more powerful than the intelligent peripheral case where the IN standards would not allow the intelligent peripheral to be the instigator of services.

The question remains as to whether it is a good thing to apply the IN concept to routers within the network. It could be argued that one of the reasons why IP networks are so successful is because there is little operator control of the core of the IP network.

The IP router case draws out the fundamental difference between the telecommunications paradigm and the Internet. A connectionless Internet Service does not have a basic call state model because there is no network layer connection. This makes it difficult to work with the equivalent of the detection point mechanism of the SSF/CCF. On the other hand, the concept of a basic call routing function is not seen to be impossible if one considers the possibility of connection-oriented IP mechanisms such as cell tagging.

4.10 Chapter Summary and Research Contributions

This chapter presented a protocol that was developed to demonstrate one of the ways in which existing IN CS-1/2 infrastructure can be utilised to support new and innovative IP-based services. The proposal has the following innovative features:

- It is advantageous over existing electronic payment systems as it does not require the use of credit cards.
- Existing IN capabilities are sufficient to implement the system.
- Two distinct options for implementing the gateway were put forward.

The proposed system is based on the principle that the large majority of consumers are reluctant to engage in electronic commerce activities due to lack of trust [OECD97a]. This lack of trust is a result of the insecurities of the transmission protocols used in the Internet, as well as the large publicity that Internet break-ins receive versus the millions of transactions that are completed successfully.

The IEPS is innovative and secure. It allows end users to engage in electronic commerce without the controversy surrounding credit cards. It utilises existing billing platforms that, through time, have been accepted by users as secure and have gained their trust. The system design utilises models from the IN world to create a robust system. This is discussed in the next chapter, which deals with the implementation of the IEPS.

The work involved designing the protocol for both the IP domain and the IN domain. For this, the work put forward two possible designs in which a gateway function can be implemented in order to allow inter-operability across two dissimilar network architectures. It was also noted that the existing IN capability sets provide sufficient functionality to be utilised in new service implementations.

The IEPS utilises existing architectures, protocols and billing systems in a way that allows network operators to further enhance their position by engaging in electronic commerce activities as an authentication authority. This is clearly a desirable situation for the operator because of the revenue streams it offers. It is also desirable for the end user, because the payment is carried out without the need for credit cards, but rather on an existing billing system that is trusted by the majority of end users.

CHAPTER 5

IMPLEMENTATION & SIMULATION OF THE IEPS SYSTEM

The utilisation of existing IN infrastructure as a means of authentication and billing was described in the previous chapter, through the description of the IEPS. This chapter describes the implementation of the IEPS.

5.1 INTRODUCTION

This chapter presents the design of the protocol and the implementation and simulation of the system. The implementation presented in this chapter covers Phase 2 of the protocol (section 4.5.2) and the interface between the gateway and the SCP (phase 3, section 4.5.3).

The reasons for implementing the system were threefold: firstly to identify potential limitations of the protocol; secondly, to examine the interactions of the functional entities in order to appreciate where the complexities of such a system lie; and, finally, to gain a better understanding of Java in a concurrent client–server environment

Section 5.2 describes the implementation of the protocol and provides a walk-through by describing the packets that are exchanged by the parties involved. Section 5.3 presents the Java classes and the state transition diagrams that implement the system and section 5.4 provides output from the simulation of the system. Finally, section 5.5 provides a chapter summary and outlines research contributions.

5.2 IMPLEMENTATION OF THE IEPS PROTOCOL

The IEPS consists of three parties: the User, the Internet Shop and the Gateway. For each of the three, a separate server is used. These are, respectively, the `uServer`, the `isServer` and the `gServer`. Additionally, in order for the protocol to be fully simulated, an `scpServer` (simulating the SCP and the G–SCF interface) and a `servletServer` (simulating servlets) are also implemented.

The system is distributed as follows: The uServer runs on the user's machine as part of the software that may be provided by the Network Operator. The gServer runs on the gateway and the isServer may either run on the machine that hosts the Internet Shop's web page or on a dedicated system.

Figure 5.1 shows the architecture of the implementation. It consists of five servers and an applet. One of the servers is used to simulate the servlet connection. The name in italics below each server identifies the workstation the server is running on. The servletSimulator and isServer are running on the same workstation.

A point to note here are the multiple instances of uServer, which represent multiple users, and the multiple instances of isServer, representing multiple Internet shops. The implementation can cope with multiple users and Internet shops.

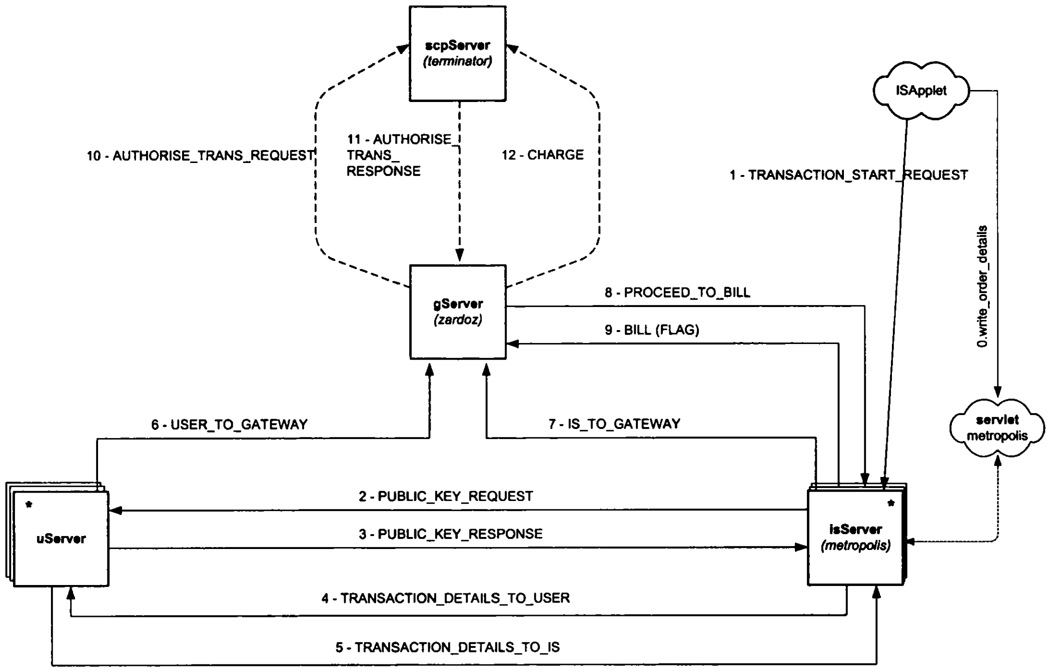


Figure 5.1: *Deployment of IEPS*

As mentioned earlier, the initiating action for the protocol is the user submitting an order on a web page. This is the trigger for the elements to begin exchanging IP packets. The isServer needs to be made aware of the items the user has selected. This requires the applet to communicate the information to the isServer, which can be achieved using a Java Servlet [Voss97a][Voss97b][Cowa01]. Servlets require the presence of a web server that is capable of providing the required underlying functionality. The use of such a platform for the simulation would not have provided additional results, as the focus was on the core part of the protocol rather than the edges (i.e. web-browser to internet shop). As a result, a server was used to achieve this communication.

In what follows, the structure of the packets that make up the IEPS is presented.

5.2.0 WRITE_ORDER_DETAILS[*]

When the user clicks the "proceed" button on the applet, the applet sends a `write_order_details` packet to the `servletSimulator`. In the case where a servlet is used with an application server, such as Tomcat Jakarta, an http request would initiate the servlet on the Internet shop. The `servletSimulator` receives the packet, extracts the individual tokens from the string and writes the information to disk. The packet that is received by the `servletSimulator` has the following structure.

Field	Type	Bytes	Obtained from
SourceIP	java.net.InetAddress	4	User's IP
OrderTotal	Float	4	Order total reported by applet
ItemsOrdered	String[]	Variable	List with ordered items and quantity

Table 5.0: *WRITE_ORDER_DETAILS packet structure*

```
BufferedReader in = newBufferedReader(newInputStreamReader(data.getInputStream()));
theStr = in.readLine();
StringTokenizer st = new StringTokenizer(theStr, ",");

System.out.println("Servlet received:");
while (st.hasMoreTokens()) {
theToken = st.nextToken();
        if (count == 1) {
           // this is the IP of the user.
           theFile = new File("IS-SERVLET-" + theToken + ".dat");
           os = new FileOutputStream(theFile);
           out = new PrintWriter(os);
        }
}
```

The code extract above shows the process of extracting the individual tokens by using the `StringTokenizer` class. The conditional if-statement checks for the first token, which contains the IP, and creates a file called `"IS-SERVLET-a.b.c.d.dat"`, where `a.b.c.d` represents the IP address of the user. The user's IP address represents a unique way of identifying the user and it is therefore treated as a **primary key**[†] throughout the protocol.

[*] Note that the section and table numbers are intentionally numbered from 0, as they correspond to the packets that are exchanged, as depicted in figure 5.1.

[†] For simulation purposes this is satisfactory; however a real implementation needs to use a user id, possibly with a session identifier. This is because the user's IP address may not be static and also IP addresses are easy to spoof.

5.2.1 TRANSACTION_START_REQUEST (TSReq)

At the same time, the applet sends a `TSReq` packet to the `isServer`. The packet structure is shown in table 5.1.

Field	Type	Bytes	Obtained from
MessageID	Int	4	Protocol Message identifier
SourceAddress	Java.net.InetAddress	4	User's IP
DestAddress	Java.net.InetAddress	4	Is.ip file

Table 5.1: *TRANSACTION_START_REQUEST packet structure*

The `SourceAddress` is obtained by the applet after it makes an `InetAddress.getLocalHost()` call. The `DestAddress` is read from a text file. This is the "is.ip" file. Two similar files exist: the "gateway.ip" and the "scp.ip" contain the IP addresses of the `gServer` and the `scpServer` respectively.*

5.2.2 PUBLIC_KEY_REQUEST (PKReq)

The table below shows the packet structure.

Field	Type	Bytes	Obtained from
MessageID	Int	4	Protocol Message identifier
SourceAddress	Java.net.InetAddress	4	Internet Shop's IP
DestAddress	Java.net.InetAddress	4	TSReq Source Address

Table 5.2: *PUBLIC_KEY_REQUEST packet structure*

When the `isServer` receives the `TSReq` packet, it sends a `PKReq` to the IP of the party that initiated the `TSReq`. This is simply the `SourceAddress` of the `TSReq` packet.

5.2.3 PUBLIC_KEY_RESPONSE (PKRes)

Table 5.4 shows the packet structure.

Field	Type	Bytes	Obtained from
MessageID	Int	4	Protocol Message identifier
SourceAddress	Java.net.InetAddress	4	User's IP
DestAddress	Java.net.InetAddress	4	PKReq Source Address
Pkey	Int	4	User's Public Key

Table 5.3: *PUBLIC_KEY_RESPONSE packet structure*

At this point the `uServer` has received a `PKReq` to which it replies by sending its public key. The `DestAddress` of the packet will be the `SourceAddress` of the `PKReq` packet. Now the `isServer` knows the public key of the user. The next step is to submit to the user a packet that contains all the items the user has ordered, in encrypted form, using the user's public key.

* In a real implementation, this would be configured on a database to allow easy management.

5.2.4 TRANSACTION_DETAILS_TO_USER (TDUser)

The packet structure is shown below.

Field	Type	Bytes	Obtained from
MessageID	Int	4	Protocol Message identifier
SourceAddress	Java.net.InetAddress	4	Internet Shop's IP
DestAddress	Java.net.InetAddress	4	PKRes Source Address
TrnsCost	Float	4	Applet saved info
TrnsNumber	Int	4	Assigned by IS

Table 5.4: *TRANSACTION_DETAILS_TO_USER packet structure*

Once the `isServer` has received the `PKRes`, it has to send to the `isServer` of the corresponding user a `TDUser` packet. The `isServer` opens the `"IS-SERVLET-a.b.c.d.dat"` file with the IP of the user, which is the `SourceAddress` of the `PKRes` packet.

```
// Delete the Public Key Request datafile.
dbManagement.PKRequestDelete(inPacket.getSourceAddress());

// first get the transaction information from the applet datafile.
theCost = dbManagement.retrieveAppletInfo (inPacket.getSourceAddress());

// Send a TransactionDetails packet.
outPacket.TDUser(java.net.InetAddress.getLocalHost().getHostAddress(),
                  inPacket.getSourceAddress(),
                  theCost,
                  dbManagement.getTransactionNumber(),
                  isPublicKey);

// increment the transaction number
dbMngmnt.writeTransactionNumber(dbManagement.getTransactionNumber()+1);

mySocket userSocket = new mySocket("USR", inPacket.getSourceAddress());
userSocket.sendPacket(outPacket.getPacket());
userSocket.killConnection();
```

The `isServer` reads the transaction number and the transaction cost reported to it by the applet, using the `dbManagement.retrieveAppletInfo` call. This is shown in the code extract above. Also shown above is that the `isServer` sends its public key to the user so that the user can encrypt the next packet.

5.2.5 TRANSACTION_DETAILS_TO_IS (TDIs)

Once the user receives the `TDUser` packet from the IS, it forwards the information to the `gServer`. The packet structure is shown below.

Field	Type	Bytes	Obtained from
MessageID	Int	4	Protocol Message identifier
SourceAddress	Java.net.InetAddress	4	User's IP
DestAddress	Java.net.InetAddress	4	TDUser Source Address
TrnsCost	Float	4	TDUser
TrnsNumber	Int	4	TDUser

Table 5.5: *TRANSACTION_DETAILS_TO_IS packet structure*

At this point, the IS and User have exchanged information and each holds the information reported to them by the other party. They must now both send the information to the gateway, using the `IS_TO_GATEWAY` and the `USER_TO_GATEWAY` packets respectively.

5.2.6 USER_TO_GATEWAY (UtoG)

Field	Type	Bytes	Obtained from
MessageID	Int	4	Protocol Message identifier
SourceAddress	Java.net.InetAddress	4	User's IP
DestAddress	Java.net.InetAddress	4	Gateway.ip file
TrnsCost	Float	4	TDUser message
TrnsNumber	Int	4	TDUser message
ConnectionNumber	Int	4	SCP (not simulated)

Table 5.6: *USER_TO_GATEWAY packet structure*

5.2.7 IS_TO_GATEWAY (IStoG)

Field	Type	Bytes	Obtained from
MessageID	Int	4	Protocol Message identifier
SourceAddress	Java.net.InetAddress	4	Internet Shop's IP
DestAddress	Java.net.InetAddress	4	Gateway.ip file
TrnsCost	Float	4	TDis message
TrnsNumber	Int	4	TDis message

Table 5.7: *IS_TO_GATEWAY packet structure*

In order for the `gServer` to send the `ATReq` message, both the `UtoG` and `IStoG` messages must have been received by the gateway and the information in them must match.

Different delays are introduced by the parties involved at various points in the protocol. For instance, there may be delays in the network, or delays by the shop in processing the order, or even by the user due to lost transmissions. As a result, these two messages can arrive at the gateway in any order (and may be amongst messages received from different Internet Shops).

The `gServer` must therefore have a mechanism for identifying matching pairs. Here, the primary key is chosen to be the transaction number. As a result, this imposes a requirement that each Internet Shop is allocated a special range of transaction numbers or has a uniquely identifiable flag within the transaction number.

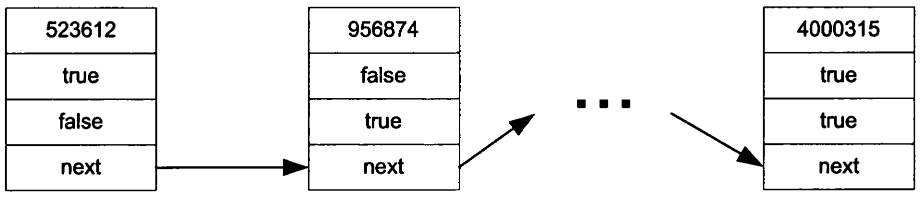


Figure 5.2: *Data as a linked list*

The local data could be implemented using a linked list, as indicated in figure 5.2. Starting from the top, the first field represents the transaction number. The next two fields are Boolean flags that indicate whether the `IStoG` and `UtoG` messages have been received. When the communications threads receive `IStoG` and `UtoG` requests, these are passed to the control process, which in turn creates a new entry in the data store and notifies the control process that a new packet has arrived.

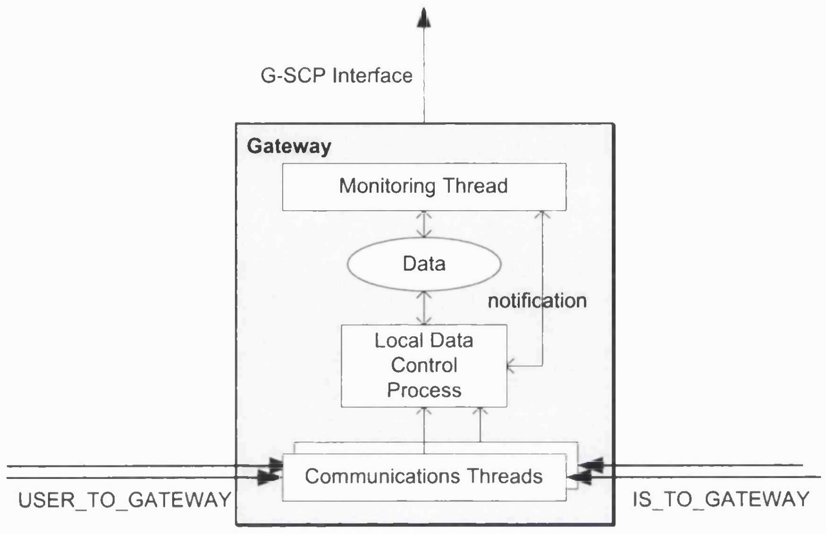


Figure 5.3: *Monitoring for matching pairs using shared local data*

The monitoring thread (figure 5.3) scans the local data for matching packet pairs and when two packets are identified as referring to the same transaction, further processing of the transaction can take place.

The purpose of the simulation was to focus on the robustness and operational issues of the protocol rather than to implement a "real-world" system. If a linked-list approach was adopted, a monitoring thread such as the one depicted in figure 5.3 would be needed. What is more, additional functionality would have to be implemented and simulated to ensure that the monitoring thread and the notification mechanisms operated correctly. Furthermore, the additional threads and the presence of the control processes would impose a requirement for an internal communication mechanism for the notifications. As a result, the approach adopted for the implementation uses a temporary file on disk. The `gServer` creates the file whenever it receives either an `IStoG` or a `UtoG` message.

Because each type of incoming packet is handled by a different thread (of the `gServer`) the method that creates the file must be synchronised. This is done in order to avoid the case where both threads are trying to access the same file concurrently. The code extract below shows the operation of handling either an `IStoG` or a `UtoG` message.

```
switch (inPacket.getID())
{
  case 6:
    // User to gateway transaction
    rvalue = dbManagement.createGWtransaction(inPacket.getTNis(),
                                              inPacket.getTCOSTis());
    // Need to save the transaction number and the IP of the internet shop,
    // as it is needed later to match a transaction number to the IP of
    // the Internet shop to send the final Proceed to bill message.
    dbManagement.saveTNumber(inPacket.getTNis(),inPacket.getSourceAddress());

    if (rvalue == 1) {
        outPacket.AuthoriseTransactionRequest(inPacket.getConnectionNumber(),
                                              inPacket.getTCOSTis(),
                                              inPacket.getTNis());
        mySocket scpSocket = new mySocket("SCP", settings.scpIP());
        scpSocket.sendPacket(outPacket.getPacket());
        scpSocket.killConnection();
    }
    break;
    ...
}
```

Another point which needs to be addressed is that the gateway must save the IP of the IS in order to subsequently send the `PROCEED_TO_BILL` message. This is achieved by making a `dbmanagement.saveTNumber` call. The first parameter is the transaction number and the second is the IP of the Internet Shop.

At this stage the `gServer` needs to get authorisation from the SCP. This is achieved by using the following three packets.

5.2.8 AUTHORISE_TRANSACTION_REQUEST (ATReq)

An `ATReq` message is sent by the `gServer` to the SCP. For simulation purposes, once the `scpServer` receives an `ATReq` message, it simply replies with the `ATReply` message.

Field	Type	Bytes	Obtained from
MessageID	Int	4	Protocol Message identifier
SourceAddress	Java.net.InetAddress	4	The gateway's IP
DestAddress	Java.net.InetAddress	4	scp.ip file
TrnsCost	Float	4	IStoG or UtoG Packet
TrnsNumber	Int	4	IStoG or UtoGPacket

Table 5.8: *AUTHORISE_TRANSACTION_REQUEST packet structure*

5.2.9 AUTHORISE_TRANSACTION_REPLY (ATReply)

The `scpServer` sends an `ATReply` to an incoming `ATReq` request.

Field	Type	Bytes	Obtained from
MessageID	Int	4	Message identifier
AuthoriseFlag	Boolean	4	Boolean flag
TrnsNumber	Int	4	The transaction number

Table 5.9: *AUTHORISE_TRANSACTION_REPLY packet structure*

5.2.10 PROCEED_TO_BILL (PTB)

The `gServer` has confirmation from the `scpServer` and at this stage performs a last double-check with the `isServer`. The `gServer` knows the IP of the corresponding IS to send the `PTB` message by reading the file created previously and by calling the `dbmanagement.saveTNumber` method.

Field	Type	Bytes	Obtained from
MessageID	Int	4	Protocol Message identifier
SourceAddress	Java.net.InetAddress	4	Gateway's IP
DestAddress	Java.net.InetAddress	4	SCP's IP from scp.ip file
TrnsNumberiS	Int	4	UtoG or IStoG message

Table 5.10: *PROCEED_TO_BILL packet structure*

5.2.11 BILL

The `isServer` performs various checks, some of which are related to stock-levels, before sending a `BILL` message. The flag of the message is set accordingly.

Field	Type	Bytes	Obtained from
MessageID	Int	4	Protocol Message identifier
SourceAddress	Java.net.InetAddress	4	The Internet Shop's IP
DestAddress	Java.net.InetAddress	4	Gateway.ip file
TrnsNumber	Int	4	PTB message
Flag	Boolean	4	When set to true, indicates that the gateway can proceed to the charging phase.

Table 5.11: *BILL packet structure*

5.2.12 CHARGE

The `gServer` receives the `BILL` message, checks the flag and sends the final CHARGE message to the `scpServer`.

Field	**Type**	**Bytes**	**Obtained from**
MessageID	Int	4	Message identifier.
ConnectionNumber	Int	4	The connection number.
TrnsCost	Float	4	The transaction cost
TrnsNumber	Int	4	The transaction number

Table 5.12: *CHARGE packet structure*

Having discussed the various packets of the protocol, the next section focuses on presenting the Java classes and the state transition diagrams that make up the system. These were developed using the Unified Modelling Language (UML) [UML99].

5.3 THE IEPS CLASSES

Having given a walkthrough of the protocol and an outline of the implementation decisions, the next sections identify three of the major base classes of the system. These were designed using UML, a modelling language that has a broad spectrum of usage. It can be used for business modelling, software modelling in all phases of development and for all types of systems, and general modelling of any construction that has both a static structure and a dynamic behaviour [UML99].

The initial aim for the development of UML was to bring together the various design methodologies, such as Booch [Booc99], OMT [Rumb91], and Yourdon [Cons79]. As the name suggests, UML incorporates ideas from these methods, thus "unifying" the disparate attempts that existed at the time. Some of the goals of UML include [UML99]:

- To model systems (and not just software) using object-oriented concepts,
- To establish an explicit coupling to conceptual and executable artefacts and
- To address the issues of scale inherent in complex, mission-critical systems.

One of the advantages of UML is its acceptance. To establish UML, the developers and Rational Software Inc. [Rational] realised that the language had to be made available to everyone at no charge. Therefore, the language is non-proprietary and open to all. The UML specification can be found in [UML99].

5.3.1 The *dbManagement* Class

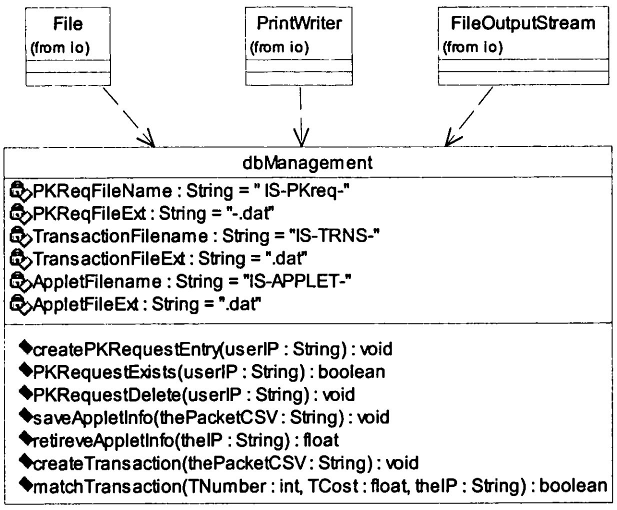

File
(from Io)
PrintWriter
(from io)
FileOutputStream
(from Io)
dbManagement
PKReqFileName : String = " IS-PKreq-"
PKReqFileExt : String = "-.dat"
TransactionFilename : String = "IS-TRNS-"
TransactionFileExt : String = ".dat"
AppletFilename : String = "IS-APPLET-"
AppletFileExt : String = ".dat"
createPKRequestEntry(userIP : String) : void
PKRequestExists(userIP : String) : boolean
PKRequestDelete(userIP : String) : void
saveAppletInfo(thePacketCSV : String) : void
retireveAppletInfo(theIP : String) : float
createTransaction(thePacketCSV : String) : void
matchTransaction(TNumber : int, TCost : float, theIP : String) : boolean

***Figure 5.4:** Class diagram for dbManagement*

The `dbManagement` class provides the database functionality required by the system. Throughout the operation of the protocol, the servers need to save a number of parameters as previously discussed.

5.3.2 The *myPacket* Class

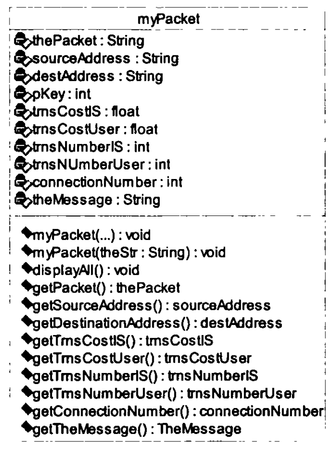

myPacket
thePacket : String
sourceAddress : String
destAddress : String
pKey : int
trnsCostIS : float
trnsCostUser : float
trnsNumberIS : int
trnsNUmberUser : int
connectionNumber : int
theMessage : String
myPacket(...) : void
myPacket(theStr : String) : void
displayAll() : void
getPacket() : thePacket
getSourceAddress() : sourceAddress
getDestinationAddress() : destAddress
getTrnsCostIS() : trnsCostIS
getTrnsCostUser() : trnsCostUser
getTrnsNumberIS() : trnsNumberIS
getTrnsNumberUser() : trnsNumberUser
getConnectionNumber() : connectionNumber
getTheMessage() : TheMessage

Figure 5.5: *Class diagram for myPacket*

The `myPacket` class provides the encapsulation for containing the IEPS packets. It has two **overloaded constructors.** It also contains a number of selector methods, which allow the users of this class to gain access to the private attributes of the class.

5.3.3 The *mySocket* Class

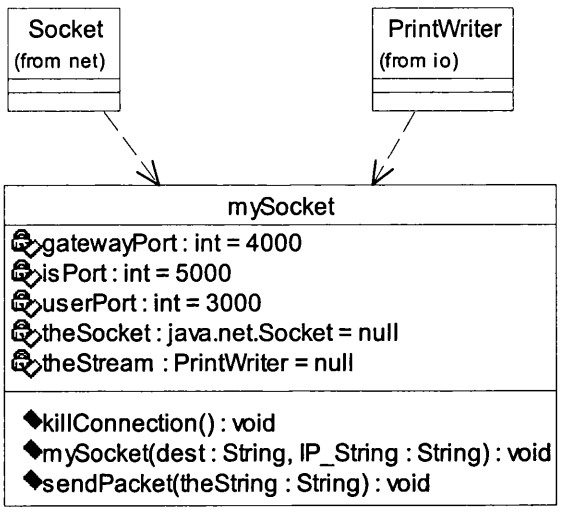

Socket
(from net)
PrintWriter
(from io)
mySocket
gatewayPort : int = 4000
isPort : int = 5000
userPort : int = 3000
theSocket : java.net.Socket = null
theStream : PrintWriter = null
killConnection() : void
mySocket(dest : String, IP_String : String) : void
sendPacket(theString : String) : void

Figure 5.6: *Class diagram for mySocket*

This class provides the basic socket functionality for the system. It comprises three methods. The constructor, `mySocket`, takes two arguments: The first is a string that denotes where the user of the class wants to connect. This is used to determine the port to be used for the connection. The second argument, the `IP_String`, denotes the IP address of the target connection.

In Java, when information needs to be transmitted using sockets, a socket needs to be created. Information is then printed onto the stream of the socket and the stream is closed, as follows:

```
theSocket = new Socket (dest_address, port);
theStream = new PrintWriter(theSocket.getOutputStream(), true);
theStream.println("this will be sent");
theStream.close();
theSocket.close();
```

The first call creates the socket, the second creates an output stream to that socket, the third sends the string to the stream, and the final two close the stream and the socket respectively. To avoid the above repetition, the `mySocket` class encapsulates the stream and the socket in the class, and the above call can now be made using the following code.

```
mySocket scpSocket = new mySocket("SCP", settings.scpIP());
scpSocket.sendPacket("this will be sent");
scpSocket.killConnection();
```

The next section discusses the implementation of the parent class, `TCPServer`, which all servers inherit.

5.3.4 The *TCPServer* Class

The `TCPServer` class creates a `ServerSocket` and accepts connection requests from clients. This is done in a separate thread. Once a connection is made, the server clones itself so that it may handle the new client connection in a new thread.

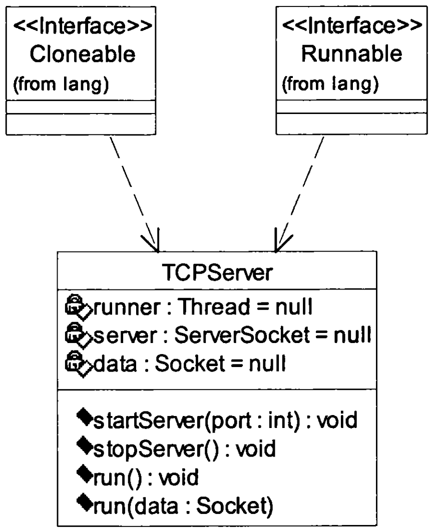


Figure 5.7: *Class Diagram for TCPServer*

The `TCPServer` implements the Runnable interface (this is because new threads will be created, which will be executed by this class). The class is Cloneable, so that a copy of this class can be created for each connection. As a result, since the copy of the class is also Runnable, another copy for each client connection can also be created. The `startServer` and `stopServer` methods are synchronised.

5.3.5 The *UServer* Class

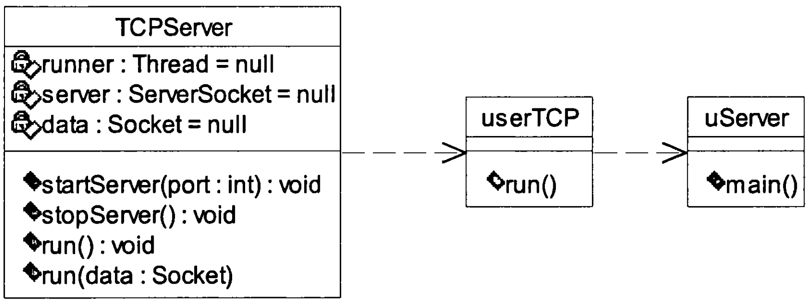
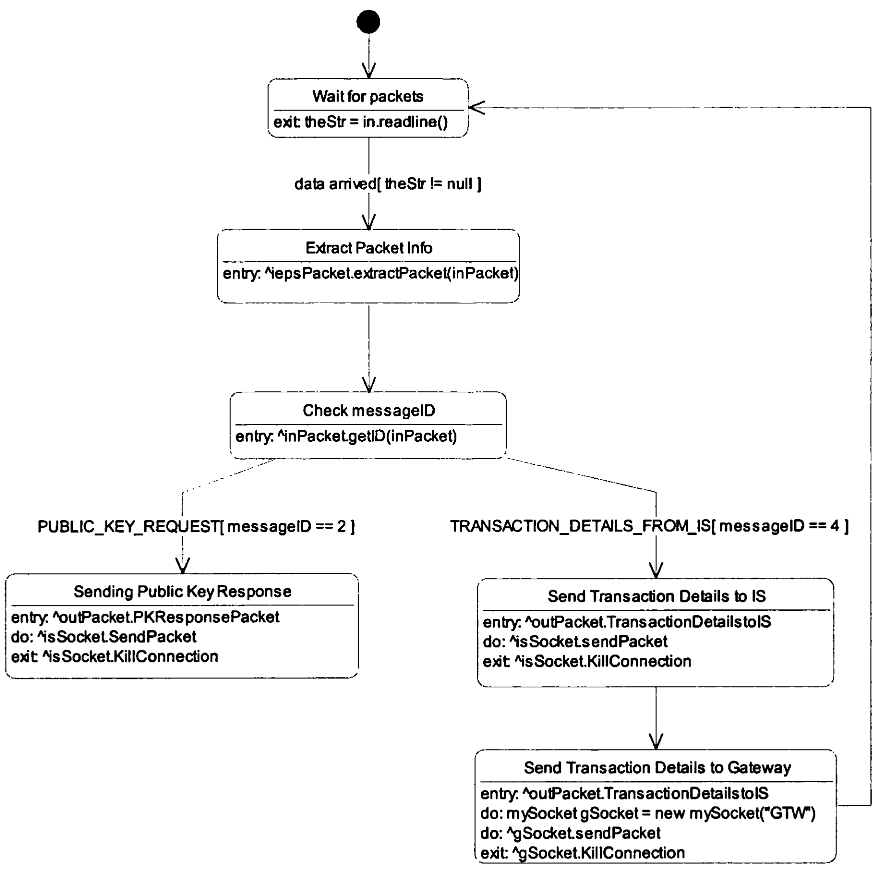


Figure 5.8: *Class diagram for UServer*



Wait for packets
exit: theStr = in.readline()
data arrived[theStr != null]
Extract Packet Info
entry: ^iepsPacket.extractPacket(inPacket)
Check messageID
entry: ^inPacket.getID(inPacket)
PUBLIC_KEY_REQUEST[messageID == 2]
TRANSACTION_DETAILS_FROM_IS[messageID == 4]
Sending Public Key Response
entry: ^outPacket.PKResponsePacket
do: ^isSocket.SendPacket
exit: ^isSocket.KillConnection
Send Transaction Details to IS
entry: ^outPacket.TransactionDetailstoIS
do: ^isSocket.sendPacket
exit: ^isSocket.KillConnection
Send Transaction Details to Gateway
entry: ^outPacket.TransactionDetailstoIS
do: mySocket gSocket = new mySocket("GTW")
do: ^gSocket.sendPacket
exit: ^gSocket.KillConnection

Figure 5.9: *State transition diagram for UServer class*

In figure 5.9 the state transition diagram for the UServer class is shown. The initial state is the state depicted at the top of the transition diagram. The exit condition from the state is specified in the lower part of the state, i.e. theStr=in.readline(). When this occurs and the condition set by the transition (i.e. theStr != null) the class moves to the `ExtractPacketInfo` state. In this state, the entry condition is verified to ensure that the received packet has not been corrected in transit.

When the class moves to the `CheckMessageID` state, a valid packet has been received, and this state therefore contains the actions that extract the packet identifier. Depending on the type of packet that was received, the class moves either to the sending `PublicKeyResponse` or send `TransactionDetailstoIS` states. Following this transition, the class eventually returns to the idle state, where it waits for further incoming packets.

5.3.6 The *GServer* Class

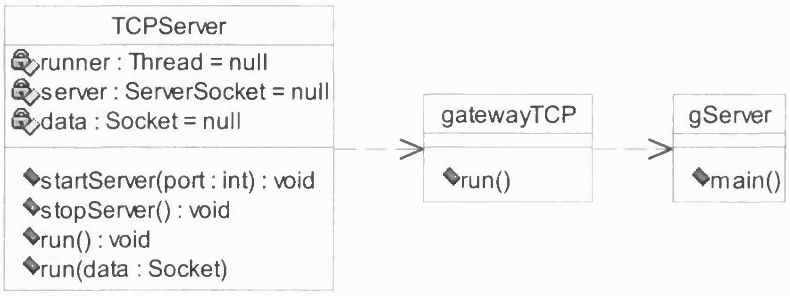


Figure 5.10: *Class diagram for GServer*

The state transition diagram for the GServer class (figure 5.11) is similar to that of the UServer class (figure 5.9).

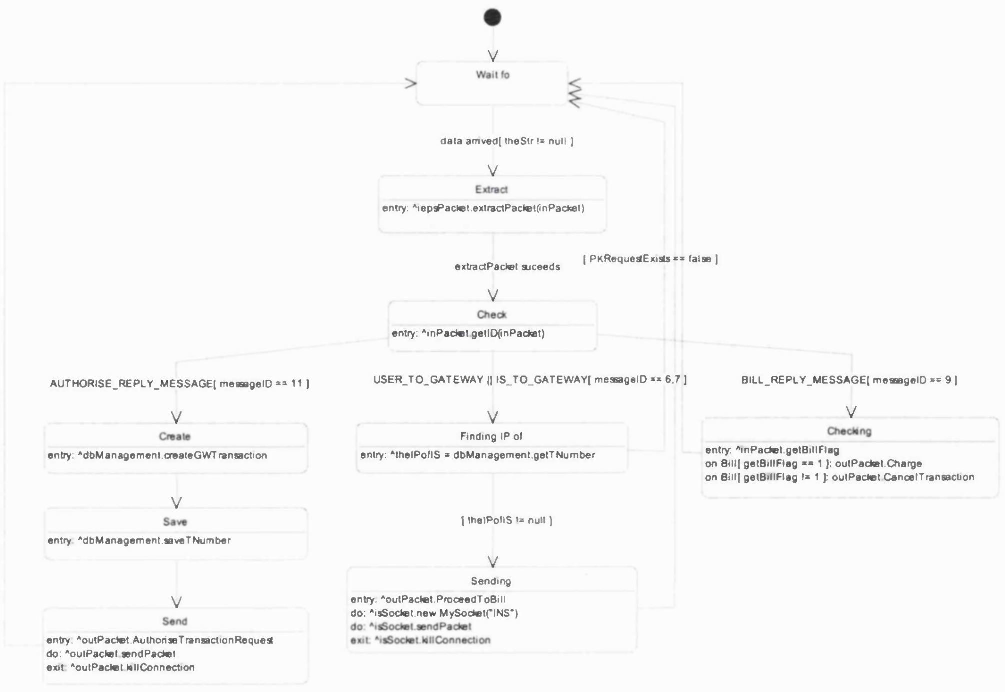

Wait fo
data arrived[theStr != null]
Extract
entry: ^iepsPacket.extractPacket(inPacket)
extractPacket suceeds
[PKRequestExists == false]
Check
entry: ^inPacket.getID(inPacket)
AUTHORISE_REPLY_MESSAGE[messageID == 11]
USER_TO_GATEWAY || IS_TO_GATEWAY[messageID == 6,7]
BILL_REPLY_MESSAGE[messageID == 9]
Create
entry: ^dbManagement.createGWTransaction
Finding IP of
entry: ^theIPofIS = dbManagement.getTNumber
Checking
entry: ^inPacket.getBillFlag
on Bill[getBillFlag == 1]: outPacket.Charge
on Bill[getBillFlag != 1]: outPacket.CancelTransaction
Save
entry: ^dbManagement.saveTNumber
[theIPofIS != null]
Sending
entry: ^outPacket.ProceedToBill
do: ^isSocket.new MySocket("INS")
do: ^isSocket.sendPacket
exit: ^isSocket.killConnection
Send
entry: ^outPacket.AuthoriseTransactionRequest
do: ^outPacket.sendPacket
exit: ^outPacket.killConnection

Figure 5.11: *State transition diagram for GServer class*

The class waits for an incoming packet, when it is received, it is checked for validity and the class moves to the appropriate internal state, depending on which packet was received.

5.3.7 The *ISServer* Class

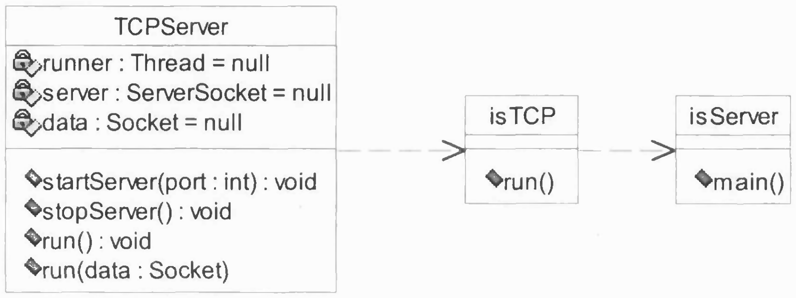


Figure 5.12: *Class diagram for ISServer*

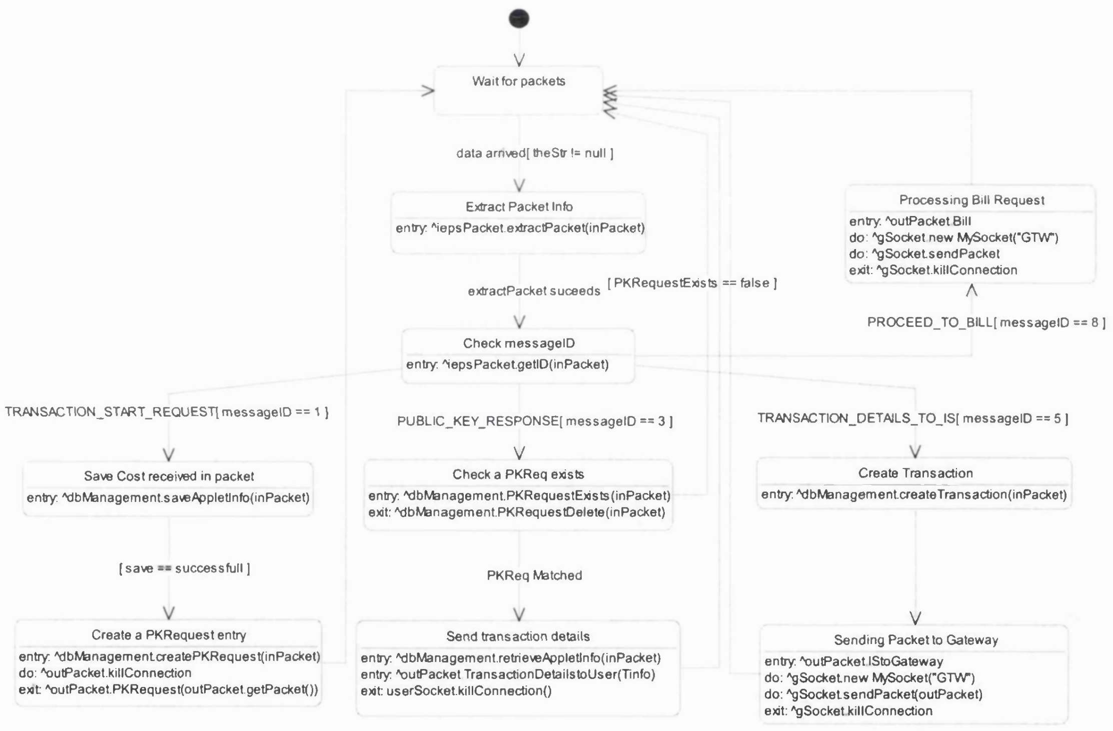

Wait for packets
data arrived[theStr != null]
Extract Packet Info
entry: ^iepsPacket.extractPacket(inPacket)
extractPacket suceeds
[PKRequestExists == false]
Check messageID
entry: ^iepsPacket.getID(inPacket)
TRANSACTION_START_REQUEST[messageID == 1]
PUBLIC_KEY_RESPONSE[messageID == 3]
TRANSACTION_DETAILS_TO_IS[messageID == 5]
PROCEED_TO_BILL[messageID == 8]
Processing Bill Request
entry: ^outPacket.Bill
do: ^gSocket.new MySocket("GTW")
do: ^gSocket.sendPacket
exit: ^gSocket.killConnection
Save Cost received in packet
entry: ^dbManagement.saveAppletInfo(inPacket)
Check a PKReq exists
entry: ^dbManagement.PKRequestExists(inPacket)
exit: ^dbManagement.PKRequestDelete(inPacket)
Create Transaction
entry: ^dbManagement.createTransaction(inPacket)
[save == successfull]
PKReq Matched
Create a PKRequest entry
entry: ^dbManagement.createPKRequest(inPacket)
do: ^outPacket.killConnection
exit: ^outPacket.PKRequest(outPacket.getPacket())
Send transaction details
entry: ^dbManagement.retrieveAppletInfo(inPacket)
entry: ^outPacket.TransactionDetailstoUser(Tinfo)
exit: userSocket.killConnection()
Sending Packet to Gateway
entry: ^outPacket.IStoGateway
do: ^gSocket.new MySocket("GTW")
do: ^gSocket.sendPacket(outPacket)
exit: ^gSocket.killConnection

Figure 5.13: *State transition diagram for ISServer class*

In figure 5.13, note the transition from the checking PKReq details state back to the idle state (wait for packets), and the associated failure event, i.e. PKRequestExists=false. This indicates that a public key request was received prior to the IS sending a request for a public key.

5.3.8 The *SCPServer* Class

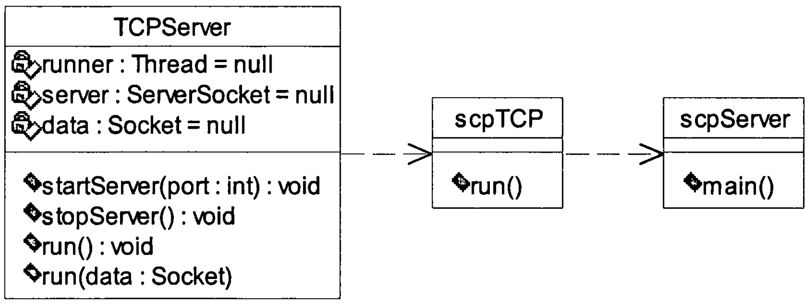


Figure 5.14: *Class diagram for SCPServer*

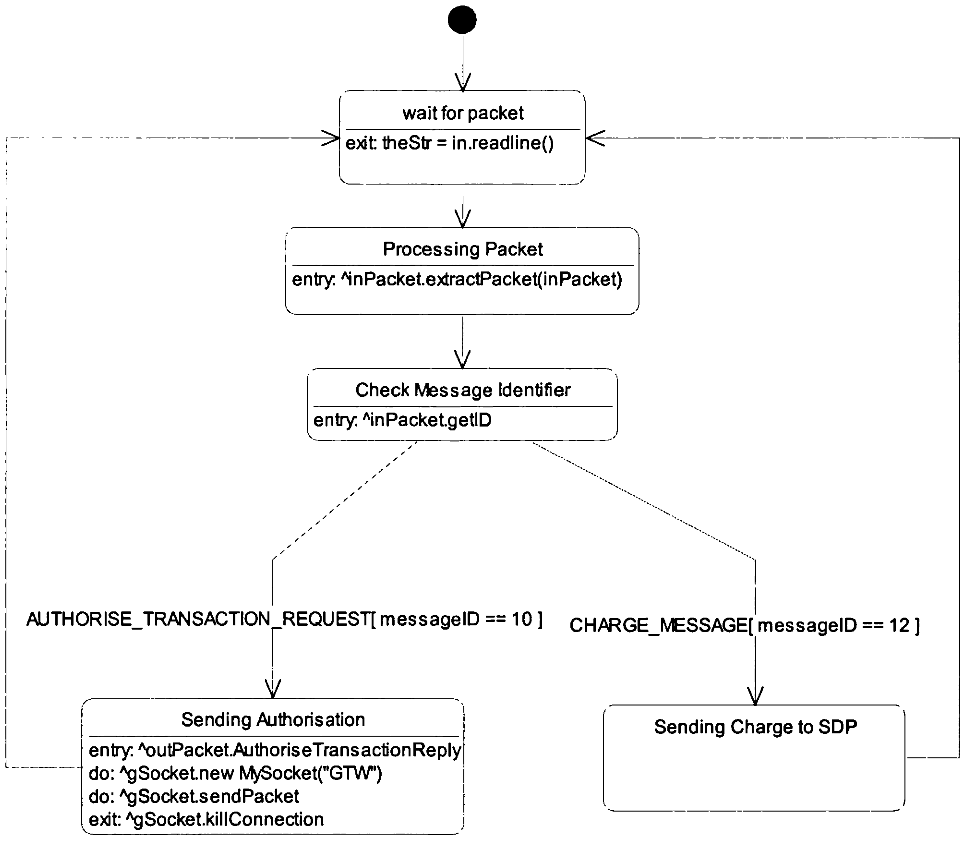

wait for packet
exit: theStr = in.readline()
Processing Packet
entry: ^inPacket.extractPacket(inPacket)
Check Message Identifier
entry: ^inPacket.getID
AUTHORISE_TRANSACTION_REQUEST[messageID == 10]
CHARGE_MESSAGE[messageID == 12]
Sending Authorisation
entry: ^outPacket.AuthoriseTransactionReply
do: ^gSocket.new MySocket("GTW")
do: ^gSocket.sendPacket
exit: ^gSocket.killConnection
Sending Charge to SDP

Figure 5.15: *State transition diagram for SCPServer class*

For simulation purposes the state transition diagram for the SCPServer class (figure 5.15) is simplified. In a real implementation INAP [Q.1208] states would have to be incorporated within the state diagram.

5.4 Results of the IEPS Simulation

In this section, the results of the simulation are presented in the form of output from each of the servers. The simulation can be followed by referring to figure 5.1 and cross-referencing the `MessageID` of each packet.

5.4.1 Output from the isServer

The simulation starts when the user submits the order through the applet. The applet sends the `TSRes` message to the `isServer`, which is the first packet below. The `isServer` then sends the `PKReq` and receives the `PKRes`. It then sends a `TDUser` message and receives a `TDIs` message.

```
Started IS-Server, Port 5000
Message received on Sat Aug 29 20:17:07 GMT+00:00 1998
-------------------------------------------------------------
*** TRANSACTION START REQUEST ***
-------------------------------------------------------------
MESSAGE ID     = 1
Source IP      = 212.228.179.77
Dest.  IP      = 128.40.38.119
Tr. Cost       = 916.5
The Packet     = 1,212.228.179.77,128.40.38.119,916.5
-------------------------------------------------------------

Message received on Sat Aug 29 20:17:10 GMT+00:00 1998
-------------------------------------------------------------
*** PUBLIC KEY RESPONSE ***
-------------------------------------------------------------
MESSAGE ID     = 3
Source IP      = 212.228.179.77
Dest.  IP      = 128.40.38.119
Enc. Key       = 9999
The Packet     = 3,212.228.179.77,128.40.38.119,9999
-------------------------------------------------------------

Message received on Sat Aug 29 20:17:15 GMT+00:00 1998
-------------------------------------------------------------
*** TRNS DETAILS TO IS ***
-------------------------------------------------------------
MESSAGE ID     = 5
Source IP      = 212.228.179.77
Dest.  IP      = 128.40.38.119
Tr. Cost       = 916.5
Tr. Number     = 44
The Packet     = 5,212.228.179.77,128.40.38.119,916.5,44
-------------------------------------------------------------

Message received on Sat Aug 29 20:17:21 GMT+00:00 1998
-------------------------------------------------------------
*** PROCEED TO BILL (from Gateway) ***
-------------------------------------------------------------
MESSAGE ID     = 8
Source IP      = 128.40.38.129
Dest.  IP      = 128.40.38.119
Tr. Number     = 44
The Packet     = 8,128.40.38.129,128.40.38.119,44
-------------------------------------------------------------
```

Figure 5.16: *Simulation output from the IS Server*

5.4.2 Output from the gServer

```
Started Gateway-Server 4000
---------------------------
Message received on Sat Aug 29 20:17:16 GMT+00:00 1998
------------------------------------------------------------
*** USER TO GATEWAY ***
------------------------------------------------------------
MESSAGE ID     = 6
Source IP      = 212.228.179.77
Dest.  IP      = 128.40.38.129
Tr. Cost       = 916.5
Tr. Number     = 44
Conn.Number    = 333
The Packet     = 6,212.228.179.77,128.40.38.129,916.5,44,333
------------------------------------------------------------
Message received on Sat Aug 29 20:17:18 GMT+00:00 1998
------------------------------------------------------------
*** IS TO GATEWAY ***
------------------------------------------------------------
MESSAGE ID     = 7
Source IP      = 128.40.38.119
Dest.  IP      = zardoz
Tr. Cost       = 916.5
Tr. Number     = 44
The Packet     = 7,128.40.38.119,zardoz,916.5,44
------------------------------------------------------------
Message received on Sat Aug 29 20:17:20 GMT+00:00 1998
------------------------------------------------------------
*** AUTHORISE TRANSACTION RESPONSE ***
------------------------------------------------------------
MESSAGE ID     = 11
Tr. Number     = 44
Authorised Flag= 1
The Packet     = 11,1,44
------------------------------------------------------------
Message received on Sat Aug 29 20:17:21 GMT+00:00 1998
------------------------------------------------------------
*** BILL ***
------------------------------------------------------------
MESSAGE ID     = 9
Source IP      = 128.40.38.119
Dest.  IP      = zardoz
Tr. Number     = 44
Bill Flag      = 1
The Packet     = 9,128.40.38.119,zardoz,44,1
------------------------------------------------------------
```

***Figure 5.17:** Simulation output from the Gateway Server*

The `gServer` receives the `IStoG` and the `UtoG` message and, after a successful match, it sends an `ATReq`, to which it receives the `ATRes` message.

5.4.3 Output from the userServer

```
Message received on Sat Aug 29 20:16:34 GMT+00:00 1998
-------------------------------------------------------
*** PUBLIC KEY REQUEST ***
-------------------------------------------------------
MESSAGE ID     = 2
Source IP      = 128.40.38.119
Dest.  IP      = 212.228.179.77
The Packet     = 2,128.40.38.119,212.228.179.77
-------------------------------------------------------
```

***Figure 5.18:** Simulation output from the User Server*
(continues on next page)

```
Message received on Sat Aug 29 20:16:38 GMT+00:00 1998
-------------------------------------------------------
*** TRNS DETAILS TO USER ***
-------------------------------------------------------
MESSAGE ID     = 4
Source IP      = 128.40.38.119
Dest.  IP      = 212.228.179.77
Tr. Cost       = 916.5
Tr. Number     = 44
The Packet     = 4,128.40.38.119,212.228.179.77,916.5,44
-------------------------------------------------------
```

Figure 5.18: Simulation output from the User Server (continued.)

Depicted in figure 5.18 are the two messages the `uServer` deals with. Firstly, the `PKReq` message from the `isServer` and, secondly, the `TDUser` message.

5.4.4 Output from the scpServer

Shown below is the `scpServer`receiving an `ATReq`.

```
Started SCP-Server 4001
-----------------------

Message received on Sat Aug 29 20:17:19 GMT+00:00 1998
-----------------------------------------------------
*** AUTHORISE TRANSACTION REQUEST ***
-----------------------------------------------------
MESSAGE ID     = 10
Tr. Cost       = 916.5
Tr. Number     = 44
The Packet     = 10,-1,916.5,44
-----------------------------------------------------
```

Figure 5.19: *Simulation output from the SCP Server*

5.4.5 Output from the IS Servlet Server

```
Started IS-Servlet/CGI Simulator 123
-----------------------------------
Servlet received:
count = 1  token = 212.228.179.77
count = 2  token = 916.5
count = 3  token = Joe
count = 4  token = Bloggs
count = 5  token = Sir Matt Busby Way
count = 6  token = Trafford
count = 7  token = Manchester
count = 8  token = n/a
count = 9  token = M4 4DX
count = 10  token = 4
count = 11  token = This is item:8
count = 12  token = 2
count = 13  token = 251.45
count = 14  token = This is item:30
count = 15  token = 3
count = 16  token = 454.72
count = 17  token = This is item:80
count = 18  token = 1
count = 19  token = 210.32
```

Figure 5.20: *Output from the Servlet simulator*

Figure 5.20 shows the `isServletSimulator`. The output is the information entered by the user when completing the applet. All the information is written onto a file, as discussed in section 5.2.0.

5.4.6 The Applet Output

Shown here is the output from the applet. It provides information to the user pertaining to the progress and status of the order.

```
C:\My Stuff\IEPS>appletviewer testnew.html
Symantec Java! JustInTime Compiler Version 3.00.029(i) for JDK 1.1.x
Copyright (C) 1996-98 Symantec Corporation

*** New order processing BEGIN ***
User address = 212.228.179.77
IS address = metropolis.ee.ucl.ac.uk/128.40.38.119
Order total was:916.5
System is now processing order.. beginning to send packets.
*** New order processing END
```

***Figure 5.21:** Simulation output from the applet*

5.5 Chapter Summary and Research Contributions

This chapter dealt with the implementation and the simulation of the IEPS. To achieve this, the following tasks were undertaken:

- Analysis of the communication mechanisms of Java
- Gaining a clear understanding of the communication capabilities of the IN
- Identification of potential limitations and complexities within the designed protocol.

One of the important requirements of the protocol was that it needed to operate in an already well-established and, more importantly, standardised environment. This meant that the protocol must be designed in such a way that it operated without requiring any change to the registration and billing phases. Such changes would weaken its position considerably, indeed to the extent that they could render the system unusable. As a result, existing information flows that provide sufficient functionality to perform the authentication, authorisation and billing were identified from the IN CS-1. Therefore, through this chapter it was shown that it is possible within the "new role of IN" to utilise existing infrastructure in a way that was never conceived before.

The simulation has also identified some of the potential limitations of the existing system. For example, there is a requirement for the protocol to support session identifiers and user identifiers, rather than IP addresses. Session identifiers, for instance, would enable the system to cope with lost connections from the user side. An additional desirable feature that needs to be looked at is the behaviour of the servers as the number of incoming requests increases.

Java was chosen as the implementation language for the simulation. If such a system is to be implemented commercially, the way that the Internet shop handles incoming requests must be optimised and functionality to support load-balancing, robustness, and scalability must be incorporated.

One of the most important components of the system is the gateway that provides the translation between the IP and IN worlds. The previous chapter showed that the gateway can be implemented in two different ways. The simulation of the system in this chapter provided a better understanding of the complexities involved in the design of such a network element. For instance, the gateway needs to maintain the robustness of the IN world at all costs, so that it does not degrade the integrity of the IN network. Further work is needed in this area in order to ensure that any gateway function that interfaces to the IN world does not hinder the robustness, integrity and reliability of the underlying network.

In implementing the system, the servers maintained state to track the progress of individual transactions. State transitions within these servers are caused explicitly by external events (for example when the servers receive a valid incoming request). One limitation of the existing implementation is that it does not maintain internal state. This means that an individual server does not keep track of "active" sessions.

Chapter 6 continues to examine state utilisation in IP-based architectures, by presenting ways to implement and maintain state. Furthermore, the chapter identifies the extent to which state is utilised in existing IP protocols.

CHAPTER 6

USE OF STATE IN ARCHITECTURES & PROTOCOLS IN THE IP DOMAIN

So far the work presented has focused on the general view of "state". This is particularly true when IP architectures were presented. This chapter examines in detail the concept and use of "state". IP-based protocols, architectures and frameworks are presented with the aim of examining the control elements – and therefore the control plane – within such systems.

6.1 INTRODUCTION

Firstly, section 6.2 provides an overview of concurrency concepts and describes how these are implemented in Java. Section 6.3 presents traditional client–server architectures and examines the use of state within them, while section 6.4 examines distributed object technologies. Section 6.5 provides a classification of the IP architectures according to their use of state. Finally, section 6.6 provides a summary of the work presented in this chapter.

6.2 CONTROLLING STATE-DEPENDENT BEHAVIOUR

State-dependent actions are implemented in software using policies that may be optimistic or pessimistic. This section examines the ways in which state-dependent actions can be configured using appropriate policies and highlights the complexities involved in deciding on the adoption of a specific policy. A complete discussion on concurrent systems and state-dependent policies can be found in [Rosc97][Baet99][Thom00][Pala00].

The discussion first focuses on the types of trigger that cause state-transitions. Actions performed by mutable objects* generally have two kinds of triggering conditions, external and internal. An **external trigger** is caused when the object receives a message from another object requesting that an action is performed [Lea99]. An **internal trigger** is caused when the object is in an appropriate state to perform the action [Lea99]. Each of these triggers has associated pre-

* A mutable object is one that can change behaviour depending on its internal state [Alex93].

and post-conditions. If a trigger is to be allowed and a transition to occur, these conditions must be satisfied.

The way in which a system deals with triggers is described by its policy on state-dependent behaviour. Policies may be optimistic or pessimistic. A protocol is called **optimistic** if it optimistically assumes that failures are rare events, so optimizing failure-free performance is more important than achieving good recovery performance. In contrast, a **pessimistic** protocol always pessimistically prepares for failures, so it is willing to pay higher failure-free overhead in order to recover faster should a failure occur [Huan95].

Table 6.1 classifies policies into these two categories. According to [Lea99], in general pessimistic policies lead to simpler and more reliable designs in most concurrent settings.

Pessimistic Policies	
Inaction	A request is ignored if it cannot be fulfilled
Balking	Failure indications are returned to the client if the action cannot be performed
Guarded suspension	Execution is suspended until preconditions become true
Optimistic Policies	
Provisional action	An action is performed but its effects are not committed until success is assured
Rollback/recovery	The by-products of partially completed actions are undone. Rollback refers to reverting back to the initial state and recovery refers to attaining an comparable valid state
Retry	Failed actions are attempted repeatedly after recovering from previous attempts

Table 6.1: *Pessimistic and optimistic policies for state-dependent actions*

The following section discusses the issues affecting the choice of policy.

6.2.1 Deciding on an Adoption Policy for State-Dependent Behaviour

In deciding which of the policies listed in table 6.1 a system is to adopt, a number of considerations need to be looked at. This section provides a concise discussion regarding these considerations.

The following issues are important when regarding the adoption of a specific policy:

- The internal and external computability: internal computability refers to the ability of the host object to detect state-based pre-conditions; external computability refers to the ability of clients or other objects to know if the host object is in a state that allows the action.
- The ability for a client to ensure that an object that was in an appropriate state remains so when a subsequent message is issued.
- The cost of computing the above preconditions [Lea99]. Determining the preconditions can be computationally more expensive than just trying the action and then coping with any

failures. The computation of the preconditions must also be balanced against the probability of failure and the cost of recovery.

Further considerations are about resource contention. This pertains to whether the action requires exclusive access to a resource [Lea99]. Associated with this is the acceptability of indefinite suspension, which refers to whether the activity that invoked the method is allowed to suspend while waiting for preconditions to become true [Lea99].

The ability to "undo" internal by-products of failure is important because, otherwise, the host may enter inconsistent states, after which nothing can be guaranteed about the future behaviour of the object or other objects that depend on it. This capability is termed **recoverability** [Lea99]. In association with recoverability, clients need to take special action upon failure and there needs to be provisions for doing so.

The next section discusses ways in which state can be represented in a system implementation.

6.2.2 Representation of State

When a system is designed and a policy on state-based behaviour (i.e. optimistic or pessimistic) is adopted, there needs to be a formal way of representing state information for the objects that make up the system. The representation of the state must be explicit, in sufficient detail to prevent actions from occurring when they are not wanted and to ensure evasive action when they fail [Lea99]. This section presents three approaches for representing state: interfaces, logical definition within variables, and history and execution states.

6.2.2.1 Interfaces

Interfaces [Somm01] provide tools for defining abstract state and implementing the required state representations. This means that since interfaces provide a particular abstraction of an object's behaviour, and interfaces cannot reference code or instance variables, they force the designer to describe monitor invariant properties and functionality in ways that help avoid the need for other designers to have to read implementation source code to discover intent.

6.2.2.2 Logical State

Logical states are usually defined in terms of predicates [Kali80] that distinguish particular ranges, values or other computable properties of instance variables. These predicates can be coded either as free-standing internal Boolean methods or as Boolean conditions written inside the methods that rely on them [Aror98].

Logical state can also be represented explicitly* in a variable, with each distinct state labelled as an integer or any other discrete data type. The instance variable representing state is then re-evaluated upon each update so that it is always accurate.

A different approach, rather than coding state as a value, is for the state to be coded as a reference to a state-object. For each state, a class describing the behaviour of the object when it is in that state is written. In the main class, a reference instance variable is created that is always bound to the appropriate state-object.

If state is represented as a state-object, then state-specific behaviour is localised as well as partitioned for different states [Gamm95]. All behaviour associated with a particular state is put into one object and, because all state-specific code lives in a State subclass, new states and transitions can be added easily by defining new subclasses. However, there is a drawback to this approach. This has to do with the distributed behaviour of different states across several State subclasses – there is an increase in the number of classes and it is less compact than a single class. Regardless of this, E. Gamma in [Gamm95] makes the point that "encapsulating each state transition and action in a class elevates the idea of an execution state to full object status. This imposes structure on the code and makes its intent clearer."

A further advantage of this approach according to [Cham93] is that "state transitions are explicit." When internal variables are used to define state, the state transitions have no explicit representation but rather "only show up as assignments to some variables" [Gamm95].

The implementation of state-objects could be achieved using **lookup tables** [Carg92]. However, according to [Gamm95] "the State pattern models state-specific **behaviour**, whereas the table-driven approach focuses on defining state **transitions**."

6.2.2.3 History and Execution States

It is desirable in some situations to maintain a complete history log that records all messages received and sent, along with all corresponding internal actions that are initiated or completed [Lea99]. As this can lead to inefficient representations of state as well as possibly uneconomical use of resources, a compromise method is to define execution state variables that are particular forms of meta-variables [Marc00].

* The disadvantage of this approach is that similar conditional statements would be scattered throughout the implementation and, as a result, the addition of a new state could require the changing of several operations, which complicates maintenance.

Execution state variables can represent the fact that a given message was received, whether the corresponding action was initiated, whether the action has terminated, and whether a reply to the message was issued.

The introductory sections presented the various policies for state-dependent actions and discussed the ways in which "state" can be represented. The following section describes the implementation of two specific approaches, one for a pessimistic policy and one for an optimistic policy.

6.2.3 Implementation of Guarded Suspension

Guarded methods are those that block if the object is not in a state in which the associated actions can be executed. "In concurrent programming, a guarded method may be thought of as a customisable extension of synchronised methods, with the 'guard' for a plain synchronised method being that the object is in the Ready execution state" [Lea99]. Correspondingly, in sequential programming, guards may be considered to be special forms of conditionals: an if-statement can check whether a condition holds upon entry to a method [Magee99].

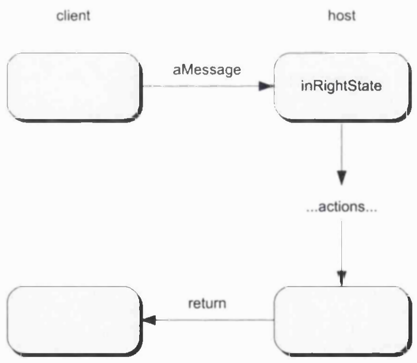


Figure 6.1: *Guarded suspension*

6.2.3.1 Waits and Busy-Waits

In Java, the standard coding method for expressing guarded waits is by using a simple loop, while invoking the Object.wait method. The code extract below identifies how this can be achieved:

```
public class GuardedClass {
     protected Boolean cond_ = false;
     protected synchronised void awaitCond() {
             while (!cond) {try {wait();}
                     catch (InterruptedException ex) {}
             }
     }
     public synchronised void guardedAction() {
             awaitCond(); //actions
     }
}
```

Busy-waits, on the other hand, are implemented using:

```
Protected void spinWaitUntilCond() {
     While (!cond_)
             Thread.currentThread().yield();
}
```

According to [Lea99], the implementation of busy-waits has the following drawbacks:

- They can waste an unbounded amount of CPU time spinning without success [Thom95]. In contrast, waits recheck conditions only when another thread sends notification that the object's state has changed, thus possibly affecting the guard condition.
- The yield in the spin-loop is not guaranteed to be effective in allowing other threads to execute so that they can change the condition.*

In addition, both implementations suffer from fairness.

A comprehensive description of the issues surrounding the advantages and disadvantages of various implementation mechanisms of guarded suspension can be found in [Boge01][Coli91].

6.2.3.2 Interrupts

A guarded wait can be viewed as if it were an undirected call to objects running in other threads asking them to take any action that makes the condition true. A notification serves as a signal that the desired condition may have been attained.

In Java, this analogy is made stronger by the fact that a wait can also be broken by an InterruptedException caused by some object invoking Thread.interrupt. Interruptions can serve as notifications indicating that the required state changes can never occur, for example due to the termination of certain threads.

6.2.3.3 Notifications

Wait-based constructions make up the bulk of the safety side of guard translation. The first step to ensure liveness† is to insert code that wakes up waiting threads when the conditions they are waiting for change value. Every time the value of any variable or object mentioned in a guard changes in a way that might affect the true value of the condition, waiting tasks should be

* This occurs for example in the case where the busy-wait is running at a high priority, therefore not allowing other processes to change the condition that would take it out of the loop.

† A liveness specification is a set of state sequences that meets the following condition: for each finite state sequence α, there exists a state sequence β such that $\alpha\beta$ is in that set [Aror98].

woken up so they can recheck guard conditions. The simplest way to do this is to insert notifyAll in methods that cause state changes.

6.2.4 Tracking State

One of the problems associated with using the NotifyAll method has to do with the possibility that some of these notifications cannot possibly affect the guard conditions of any waiting thread. These are ineffective notifications and can be eliminated by using logical state analysis. So, rather than generating notifications for all changes in instance variables, notifications are issued only upon transitions out of the logical states in which threads can wait. The disadvantage of this approach is that any changes in the implementation of the class may require different partitioning of logical state, which alters both the guard and the notification conditions for the base methods, which in turn leads to a total rewrite of the class.

Another way to monitor state is by tracking **state variables**. State variables represent the entire logical state of an object, usually in a single instance variable. According to [Lea99] the most extensible way to implement state-variable designs is to isolate state re-evaluation in a single method that is called after each update method.

6.2.5 Optimistic Policies

In pessimistic designs objects refuse to engage in actions unless they are known to be in states that allow the action to succeed [Stro85]. In optimistic, try-and-see designs, objects proceed with actions without necessarily checking to see if all preconditions are met. However, they also possess strategies and mechanisms for detecting failures and, when necessary, undoing the effects of any of the actions that led to failure.

Optimistic control techniques share three basic features: a way of detecting failure, for example by assessing the logical state; a way of dealing with failure; and a way of dealing with the consequences of actions leading to failure.

Dealing with the consequences that lead to the failure can be achieved in a forward or backward direction. One approach to dealing with the consequences is by provisionally performing the operations in a dry-run way. When this is successful and the possibility of failure has been ruled out, the operations are re-performed only this time their actions are not running in a dry-run mode. A second approach is to use rollback and recovery for every action.

It can be argued that optimistic policies are computationally expensive. This is because of the large number of variables that need to be maintained to control operations in a dry-run manner.

Also to implement rollback and recovery, every internal system message would need its counter message to undo the effect of the original message, although in some cases it is possible to implement optimistic systems without the need of additional messages [Venk97]. An additional method to implement rollback and recovery is through message logging [Alvi98]; the cost of recovery using message logging is discussed in [Rao00].

However, regardless of the computational expense, if a system's functional requirements include real-time characteristics or some safety-critical issues, then the computational complexity is a minor concern. A performance-oriented comparison of optimistic and pessimistic policies can be found in [Song95].

The following section builds on the theory presented in the previous sections and moves on to examine IP-based protocols in order to highlight their reliance on state-dependent actions as well as internal and external triggers.

6.3 USE OF STATE IN CLIENT–SERVER ARCHITECTURES

A client–server architecture is a network architecture in which each host or process on the network is either a client or a server. Some of the characteristics of client–server architecture include [Orfa99]:

- Service: Client–server is primarily a relationship between processes running on separate machines. The server process is a provider of services and the client is a consumer of services.
- Shared resources: A server can service many clients at the same time and regulate their access to shared resources.
- Asymmetrical protocols: There is a many-to-one relationship between clients and server. Clients always initiate the dialogue by requesting a service. Servers passively await requests from the clients.
- Transparency of location: The server is a process that can reside on the same machine as the client or on a different machine across a network.
- Message-based exchanges: Clients and servers are loosely coupled systems that interact through a message-passing mechanism. The message is the delivery mechanism for the service requests and replies.

Irrespective of the transport layer protocol that is used or the type of server, there exists a general categorisation of client–server architectures that is included in this section for completeness. These are the "fat clients" and "fat servers."

Client–server architectures often require the client to act as a server at some point during the execution of a service. For instance, in the system described in section 6.2, the web server is a server from the applet's point of view but it is also a client when the requests are forwarded to the NMS command processor.

For this reason, client–server applications can also be differentiated by how the distributed application is split between the client and the server [Orfa99]. A fat server model places more functionality on the server, whereas a fat client model places more functionality on the client. Fat clients are the more traditional form of client–server architectures, where the bulk of the application runs on the client side. For example in a database server, the client knows how the data is organised and stored on the server side.

It may be apparent from this initial discussion of client–server architectures that in most cases there is no clear-cut line that separates a host from being a "pure client" or a "pure server." This grey-scale representation holds true in a discussion of state-based versus stateless architectures.

Traditional IP-based protocols and architectures are overwhelmingly based on the client–server model. Edge components (clients) require services from the core of the network (servers). Some of these architectures are defined as stateless, claiming that "no state information" is maintained thereby making it "easy to cope with failures" of servers [Schu98].* The following sections present IP-based client–server architectures, beginning by looking at the RADIUS protocol, and analyse the use of state in them.

6.3.1 Remote Authentication and Dial-In User Service

The Remote Authentication and Dial-In User Service (RADIUS) is an IETF protocol "for carrying authentication, authorisation and configuration information between a Network Access Server that desires to authenticate its links and a shared authentication server." [RFC2138].

A RADIUS server is usually deployed (as the name suggests) to authenticate remote access to resources. For instance, organisations that allow employees to gain access to corporate intranets

* There are "opposing" views as to whether state should be maintained in a system. In the author's view, state is useful because it provides the system designer with a powerful tool to control the behaviour of the system. State-dependent information can also be used in systems that are more general, in order to collect information pertaining to the status of the system, as will be shown in chapter 8. Furthermore a conclusive view on state is presented in chapter 9.

utilise RADIUS components to achieve this. This section focuses on the presence of state behind a RADIUS server, rather than the operation of the protocol.

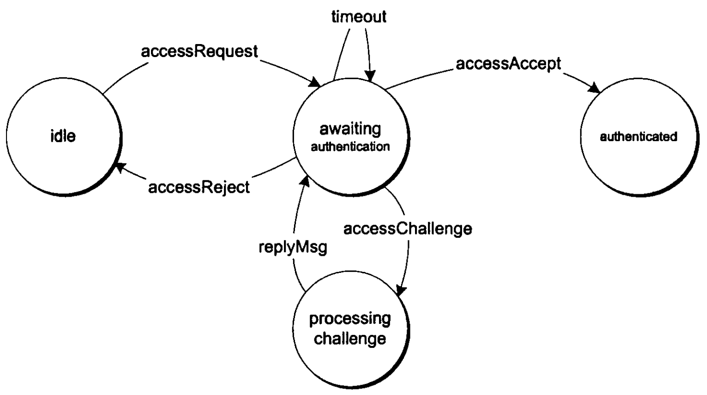


Figure 6.2: *State transition diagram for RADIUS client*

From [RFC2138], the state model presented in figure 6.2 can be derived for a RADIUS client. The state transition diagram comprises four states and six transitions. More importantly, the timeout transition implies that the RADIUS client needs to maintain information about the time elapsed since the accessRequest message was sent to the server.

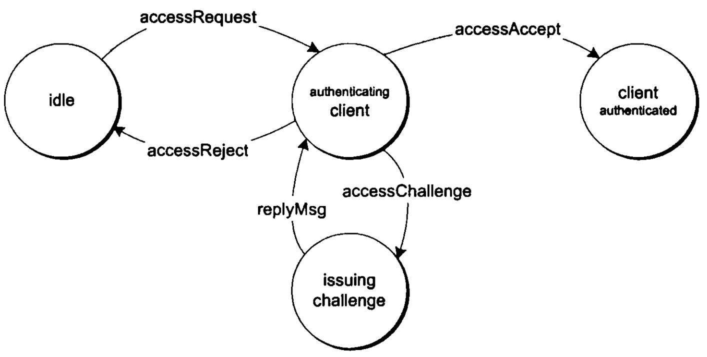


Figure 6.3: *State transition diagram for RADIUS server*

This is not the case for the RADIUS server, as depicted in figure 6.3. It can be seen that there are no timers associated with the transitions*. Essentially, it is the responsibility of the client to initiate any re-transmissions.

In [RFC2138], the authors also note that "the stateless nature of this protocol simplifies the use of UDP" and that "UDP simplifies the server implementation." The initial approach† towards

* This is true because the state transitions are generated from external messages rather than internal timers.

† An enhanced version of RADIUS is currently under development by the DIAMETER project [Calh01] of the IETF; the DIAMETER API is presented in [Kemp01]. The specification [Calh01] and the API

RADIUS was one that assumed a single process with a single request. The request is received, processed and returned – therefore the complexities associated with multiple threads do not apply and, as a result, there is no need for state management.

6.3.2 Authentication, Authorisation and Accounting

The Authentication, Authorisation and Accounting (AAA) Architecture of the IETF, aims at "providing a generic framework that allows complex authorisations to be realised through a network of interconnected AAA servers" [Laat00].

This section examines the work presented in [Laat00], [Voll00a], and [Voll00b] by looking at the protocols and the architectures from the perspective of state utilisation and the extent of its use.

[Voll00a] presents the requirements for Authorisation of Internet Resources and Services. The generic framework identifies the following conceptual entities that may be participants in an authorisation:

- A user who wants access to a service or resource
- A user home organisation that has an agreement with the user and checks whether the user is allowed to obtain the requested service or resource
- A service provider's AAA server, which authorises a service based on an agreement with the user home organisation without specific knowledge about the individual user.

Figure 6.4 depicts these conceptual entities as well as the service agreements (thick dashed lines) between the parties involved.* The figure also identifies the sequences of accessing the resources for a single-domain scenario. In sequence 1, the agent sequence, the user communicates with the resources (service equipment) through the AAA server. In sequence 2, the pull sequence, the user communicates directly with the resources. In sequence 3, the push sequence, the user must first contact the AAA server but direct access to the resource equipment is permitted following the authentication and authorisation by the server. A detailed explanation of these procedures can be found in [Voll00a] and [Laat00].

[Kemp01] explicitly define two state machines: there is a peer state machine, section 8.0 of [Calh01], and a session state machine, section 11.1 of [Calh01]. Furthermore, the client session manager is expected to maintain both the peer state machine and the session state machine [Kemp01].

* These service agreements can take the form of formal contracts or service level agreements. However, an important element in any agreement is trust. The authors in [Laat00] say that "trust is necessary to allow each entity to 'know' that the policy it is authorising is correct. This is a business issue as well as a protocol issue." The issue of trust was also highlighted in section 4.2.1.

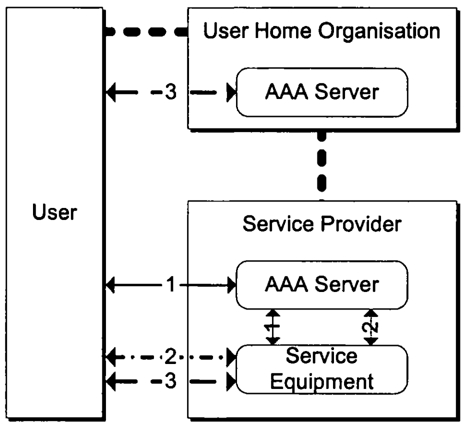


Figure 6.4: Basic authorisation entities, service agreements, and access methods

The emerging open standards through APIs (such as the ones discussed in section 3.4.2.3) enable the provisioning of service components, with the final service offering being a combination of service components from distributed service providers. For this reason, the AAA authorisation framework also considers the scenario where services are combined across administrative domains. Figure 6.5 shows the agreements present in a distributed service hierarchy.

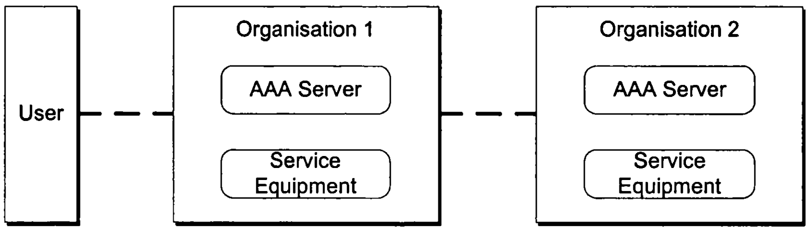


Figure 6.5: Distributed services and agreements

The agreements in the distributed services scenario imply that "the request from the User will be authenticated and authorised by the first organisation, then forwarded to the second organisation" [Voll00a].

As the authorisation requests may be chained (e.g. by forwarding from organisation 1 to organisation 2 etc.) there is a requirement for resource management. Furthermore, in many applications, the authorisation results in establishing an ongoing service, i.e. a session. "Each of the servers involved in the authorisation may also want to keep track of the state of the session, and be able to effect changes to the session if required" [Voll00a]. The framework proposes the use of a resource manager that is responsible for tracking the session as well as being able to initiate changes to the session and inform other resource managers when changes occur.

The Resource Manager (RM) is defined as "the anchor point in the AAA server from which a session can be controlled, monitored, and coordinated even if that session is consuming network resources or services across multiple Service Provider administrative domains" [Voll00a]. Numerous requirements for the RM are identified in the specification, however one that is of importance for the discussion is that an RM cooperates with "policy servers" or policy decision points [Stev99, section 7.3.3]. The RM "maintains internal state information, possibly complex cross-administrative domain information, supported by dialogues with its peer Resource Managers" [Voll00a].

Some of the issues that are identified by [Voll00a] include the capability of service equipment to notify its resource manager when a session terminates or changes state; the RM must inform other RMs that keep state for this session. The RM must also set a time limit for each session, which must be refreshed by having the resource manager query for authorative status or by having the authorative source send periodic keep alive messages that are forwarded to all RMs in the authorisation chain. Finally, any RM in the chain must have the ability to terminate a session. This requires the RM to have knowledge of at least the adjacent AAA servers in the authorisation chain.

In order to provide the above functionality, the RM must maintain session state information in order to make decisions about new sessions based on the state of existing ones and to allow monitoring of sessions by all interested AAA Servers. Furthermore, **session identifiers** are required to identify sessions; these must be unique within each AAA Server and, according to [Voll00a], it is desirable that the session identifier for a specific session be the same across all AAA servers.

The requirements for maintaining session information in AAA servers increase the complexity of the server, especially if session state must be maintained across administrative boundaries.

The RM must be able to track the state of sessions and allocate resources to a session that is associated with an AAA server. Furthermore, it may track use of resources allocated by peer resource managers to a session. All session-specific AAA state information required by the AAA server is also maintained by the RM. Session information includes pointers to peer RMs in other administrative domains that possess additional AAA state information that refers to the same session.

6.3.3 Common Open Policy Service Protocol

The Common Open Policy Service protocol (COPS) [RFC2748] is a simple query and response protocol that is used to exchange policy information between a policy server, i.e. a policy decision point, and its clients, i.e. policy enforcement points.

Figure 6.6 depicts a simple configuration for a framework for policy-based admission control, defined in [RFC2753]. The two main architectural elements for policy control are the policy enforcement point (PEP) and the policy decision point (PDP). The PEP is a component at a network node and the PDP is a remote entity that may reside at a policy server.

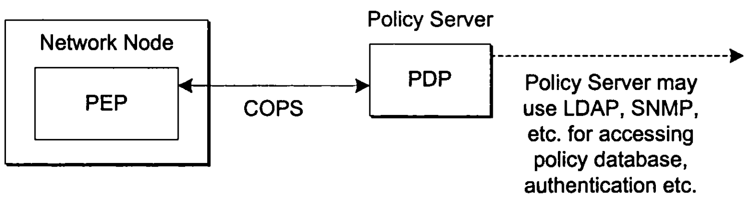

Network Node
PEP
COPS
Policy Server
PDP
Policy Server may use LDAP, SNMP, etc. for accessing policy database, authentication etc.

***Figure 6.6:** The primary policy control architecture components*

The PDP may make use of additional mechanisms and protocols to achieve additional functionality such as user authentication, accounting and policy information storage.

The interaction between the components begins with the PEP receiving a notification that requires a policy decision. The PEP then generates a request for a policy decision and sends it to the PDP. The PDP returns the policy decision and the PEP enforces it by appropriately accepting or denying the request.

The COPS protocol maintains state information in the following ways.

- The request/response state is shared between the client and server. This means that requests from the client PEP are remembered by the remote PDP until they are explicitly deleted by the PEP. Furthermore, responses by the remote PDP can be generated asynchronously at any time for a currently installed request state.
- The state from various events may be associated. This means that the server may respond to new queries differently because of previously installed request/decision states that are related.
- The protocol is stateful in that it allows the server to push configuration information to the client and allows the server to remove such state from the client when it is no longer applicable.

As the protocol maintains state information, it must be able to deal with broken connections as well as provide synchronisation methods. When a TCP connection is lost, the PDP is expected to clean up any outstanding request related to request/decision exchanges with the PEP. Once a connection is re-established, the "PEP is expected to notify the PDP of any events that have passed local admission control. Additionally, the remote PDP may request that all the PEP's internal state be re-synchronized (all previously installed requests are to be reissued) by sending a Synchronize State message" [RFC2748].

6.3.4 Web Servers

Arguably, the capabilities of the HTTP protocol with regard to complex manipulation of state-dependent actions are limited. In fact, the HTTP protocol is stateless and a web server forgets all information about a particular client after it has responded to a specific HTTP GET request.

Often there is a requirement to maintain information for a session in a web browser. For instance, when a CGI form is filled, and the user moves to the next page, the information submitted by the user on the first page needs to be transferred onto the next page. This can be achieved using hidden fields within the CGI form. Hidden fields are invisible fields that store the information a user enters and resubmit that information in subsequent forms without any need for the user to re-enter it or even be aware that the information is being passed around. Arguably, hidden fields act as variables that maintain "state" between form submissions; here, the term "state" is used very lightly, since holding the value of a variable across HTTP transactions in the same session is regarded by the author as a very weak form of "state."

An alternative way to maintain session information is by using cookies. A cookie [Java131][Laur98] is a small piece of data that is stored in the client on behalf of a server. Typically, servers use a cookie to store the user identifier or basic configuration information. The cookie is sent back to the server in subsequent page requests from this client.

Session persistence generally means that a client has reserved some form of session state on a server and that state is maintained even if connections are destroyed and re-established. Cookies enable HTTP sessions to be persistent and the state information to be maintained even when the session has ended.

Arguably, cookies are a new approach towards managing state. Until now, all the architectures presented have maintained state throughout the execution lifecycle of the specific process requiring state. Cookies however, enable state to be maintained even after a process has terminated. The term persistence is re-defined in section 6.4 where persistence for CORBA and Enterprise Java Beans (EJB) is defined.

6.3.5 The Session Initiation Protocol Revisited

As discussed in section 3.3.2.1.1, the operation of the SIP protocol caters for two modes of operation: proxy server and redirect.

When SIP is used in a proxy mode, the proxy server may be either stateless or stateful. A proxy server operating in stateful mode tracks incoming requests that generate outgoing requests and the outgoing requests. Furthermore, a stateful proxy acts as a virtual user agent client–server (UAC, UAS) by implementing the server state machine, when receiving requests, and the client state machine for outgoing requests.*

A stateful proxy needs to maintain internal tables to store previously processed requests (including acknowledgements and responses), in order to decide how to deal with further incoming messages. When an incoming request is received, the server must check a number of fields in the incoming packet (for instance, the To, From and Call-ID fields) against existing requests to determine how to deal with it. Furthermore, acknowledgements and responses must also be examined against the table that contains previously processed messages to determine how to respond.

A stateless proxy on the other hand, forgets all information once an outgoing request is generated. As a result, a stateless proxy does not behave as a virtual UAC/UAS but forwards incoming requests downstream and all responses received upstream. Furthermore, "proxies that accept TCP connections must be stateful otherwise if the proxy were to lose a request, the TCP client would never retransmit it" [RFC2543].

* The exception to this is when receiving a 2*xx* response to an INVITE [RFC2543].

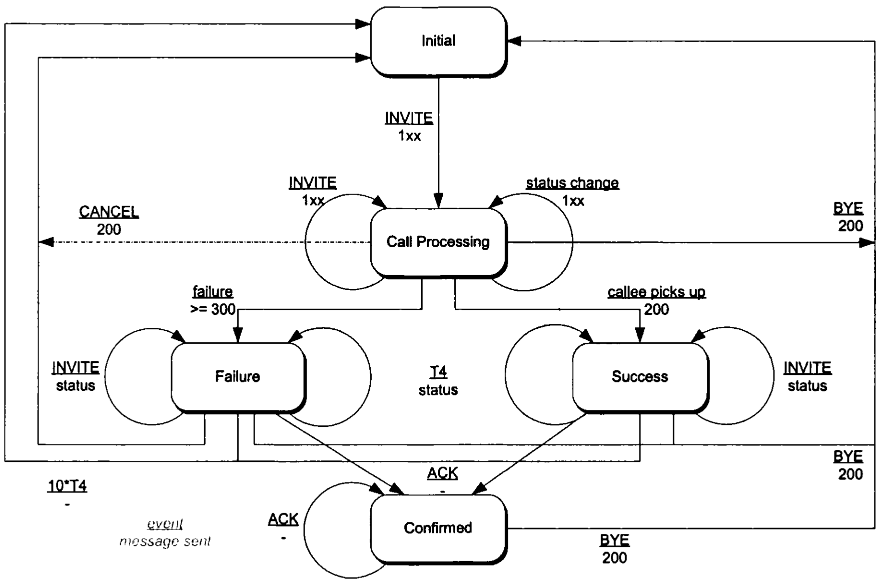


Figure 6.7: SIP server state transition diagram

The state within a SIP proxy server is explicitly defined in the case of stateful SIP proxy servers. However, even in the case where a stateful SIP proxy server is not used, its behaviour can still be characterised by a state transition diagram. This does not mean that internal state is maintained. The events and movement from one state to the next are based on an ordered set of client–server transactions, as depicted in figure 6.7 [Schu98].

6.3.6 Integrated Services Architecture

The Resource ReserVation Protocol (RSVP) provides resource reservations for multicast or unicast data flows [RFC2205]. In terms of state maintenance, RSVP opts for the soft state approach. The state is maintained within the routers and periodically refreshed by incoming packets.

6.3.7 Use of State in Client–Server Architectures: A Summary

This section provides a summary and identifies whether the systems discussed maintain state and support sessions. It also identifies whether transitions are triggered exclusively by external events or whether internal triggers are automatically generated.

The RADIUS protocol is simple as it comprises a small number of transitions and states, and is based on simple message-passing. State transitions are triggered by both external and internal events, such as requests. In the case of the RADIUS client, there are also internal triggers arising from the timer that is maintained to detect timeouts and issue re-transmissions. However, it could be argued that the simplicity of the protocol arises from the fact that the initial approach taken for its design assumed a single process with a single request. This removed any concurrency issues such as the ones discussed in section 6.2.

The next architecture that was presented was that of the AAA. This architecture is arguably simple until the complexities of open service provisioning through APIs and third-party service providers are introduced. These require the AAA architecture to be able to provide resource management (for the reasons identified in section 6.3.2).

The introduction of the resource manager considerably complicates the implementation of the AAA architecture. The RM is a complex functional entity with compounded requirements. It is responsible for maintaining state information across administrative boundaries, tracking sessions by keeping a time limit for each of them and for communicating with peers in order to terminate sessions across domains. Furthermore, state and session information must be replicated and kept up-to-date along the chain of AAA servers; this means that resources need to inform the RM when an event occurs that may affect the state.

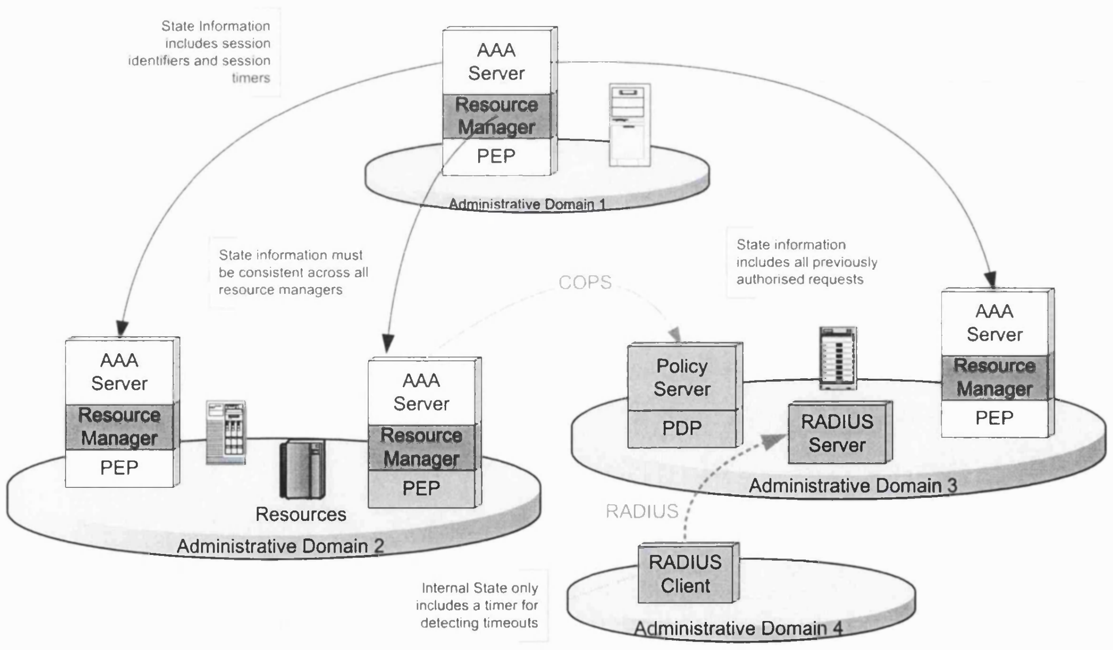


Figure 6.8: *Communication between resource managers across administrative domains*

A further specific requirement that was presented is that "any resource manager in the chain must have the ability to terminate a session" [Voll00a]. For instance, in figure 6.8, the resource manager in administrative domain 2 must be able to terminate the session of a user who lies within domain 1 and is accessing resources in domain 3 through domain 2.

The capabilities for an entity such as the resource manager point towards an architecture that resembles more the control plane of the PSTN rather than an IP network. The RM is responsible for controlling peer sessions across domains and terminating sessions that are in the path. To provide this, state is maintained at the heart of the IP network in the case of RSVP. This approach is similar to that provided by SSPs and SCPs in the IN architecture. The author has argued in [Solo00a] that such state information could be used to describe a BCSM-like state machine.

Figure 6.8 also depicts a PEP, a PDP and a policy server communicating using the COPS protocol. The protocol relies on storing state information for all previously processed requests. This is because previous requests may alter the behaviour of the RADIUS server for new requests.

The following section moves on to discuss the presence of state in distributed object technologies.

6.4 USE OF STATE IN DISTRIBUTED OBJECT TECHNOLOGIES & ARCHITECTURES

The focus of the work presented in this section is to identify the use and presence of state in Distributed Object Technologies (DOT). A comprehensive introduction to distributed computing can be found in [Orfa99]. All DOTs include the characteristics and mechanisms that are given below [Veni00]:

- Remote Method Invocation (RMI) is an evolution of the Remote Procedure Call [RMI] method for object-based distributed scenarios. While classical RPC [RFC1831] foresees the invocation of a function that is naturally separate from the data it handles, in an object-based distributed environment, RMI invokes operations (methods) on specific object instances.
- Implementation-independent Interface Definition Language (IDL) defines the interfaces between objects. An object is characterised by a contractual interface that describes the methods and attributes that are available to clients [CORBA95][Quantitative97]. In a distributed environment, objects need to communicate even though they may have been implemented in different programming languages and the contractual interface defined in IDL enables this communication.
- Location transparency allows objects that participate in interactions to do so without conveying location information or being aware of the actual transport mechanism that is used for this communication [Orfa99].

The next sections discuss the distributed object architectures of CORBA and DCOM and Enterprise Java Beans.

6.4.1 Common Object Request Broker Architecture

The Common Object Request Broker Architecture (CORBA) is the product of the Object Management Group (OMG) consortium. CORBA is an architecture based on the concept of a common software or object bus allowing for distributed object inter-operability and providing a wide set of services to interacting objects [CORBA95]. The OMG has also defined the Object Management Group Architecture (OMA) with the goal of "providing a high level specification of the functionality needed for object oriented distributed processing" [OMA97].

CORBA provides a description of the interfaces and services that an OMA-compliant Object Request Broker (ORB) must implement in order to conform with the OMG standards [CORBA95]. In addition, it defines a software infrastructure to facilitate the development of reusable and portable applications in a distributed environment [CORBA95].

One of the services provided by the CORBA Services Specifications is that of Persistent State Service (PSS) [PSS99]. The PSS "presents persistent information as storage objects stored in storage homes. Storage homes are themselves datastores; a datastore is an entity that manages data" [PSS99].

A full description of the capabilities provided by the PSS can be found in [PSS99]. The focus here is on a small subset that is of interest to illustrate the use of state models within CORBA.

6.4.2 Enterprise Java Beans

This section discusses the Enterprise Java Beans (EJB) architecture and focuses on the specific issues regarding state management, presence of state and persistence. A complete and comprehensive coverage of the architecture can be found in [EJB99]. However, for the purpose of completeness a short introduction to the constituents of the architecture is provided before the issues regarding state are examined.

EJB is an architecture for component-based distributed computing [EJB99]. It defines a state-management protocol that, according to [EJB99], is simple but "provides an enterprise Bean developer great flexibility in managing a Bean's state."

An Enterprise Bean is a body of code with fields and methods to implement modules of business logic. Client programs interact with one or more Beans and an Enterprise Bean can be implemented to interact with other Beans. An Enterprise Bean is a building block that can be used alone or with other Beans to build a complete and robust thin-client multi-tiered application.

There are two types of Enterprise Beans: session Beans, which implement business tasks, and entity Beans, which implement business entities.

Table 6.2 provides a high level view of the comparison between session and entity Beans.

Session Beans	Entity Beans
Contains conversation state	Represents data in a database
Handles database access for the client	Shares access for multiple users
Persists for the life of the client	Persists as long as data in the database
Can be transaction-aware	Is transaction-based
Does not survive server crashes	Survives server crashes

Table 6.2: *Session and entity beans*

Figure 6.9 presents the state diagram for a stateful session Bean. Note the presence of time-dependent transitions, such as timeouts. These imply internally generated messages.

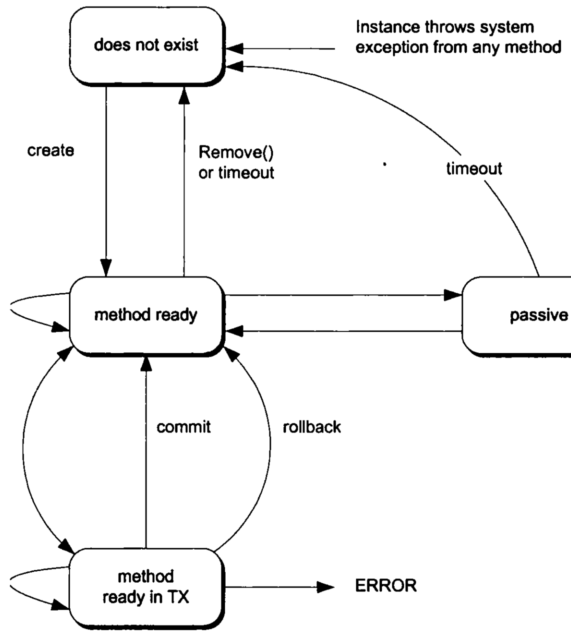


Figure 6.9: *Stateful session bean state diagram based on EJB*

The presence of state for session and entity beans is related more to "transactional" state rather than "control" state. Transactional state information is used to decide the validity of a transaction on a database; control state information is used to decide, for instance, the availability of resources as in the other architectures discussed.

6.5 A State-Based Classification of IP-Based Protocols

The work presented up to this point showed that the majority of the systems examined rely on state to some extent. This section provides a classification of the systems using metrics such as whether:

- state is critical to the operation of the system, i.e. the system cannot function if the state information is lost.
- the system is stateless or state plays a supplementary role to the operation of the system. Systems that maintain no state information appear in this category, as do systems where state is maintained but is not critical for the operation of the system.
- maintaining session information is a requirement. This classification includes systems that need state information to be maintained in the form of a session, such as a system that maintains a history of all processed requests during a single lifecycle.
- the use of persistent sessions is mandatory. Systems that maintain state information which can be retrieved across subsequent lifecycles fall into this category.

With the above metrics in mind, the Venn diagram depicted in figure 6.10 can be constructed.

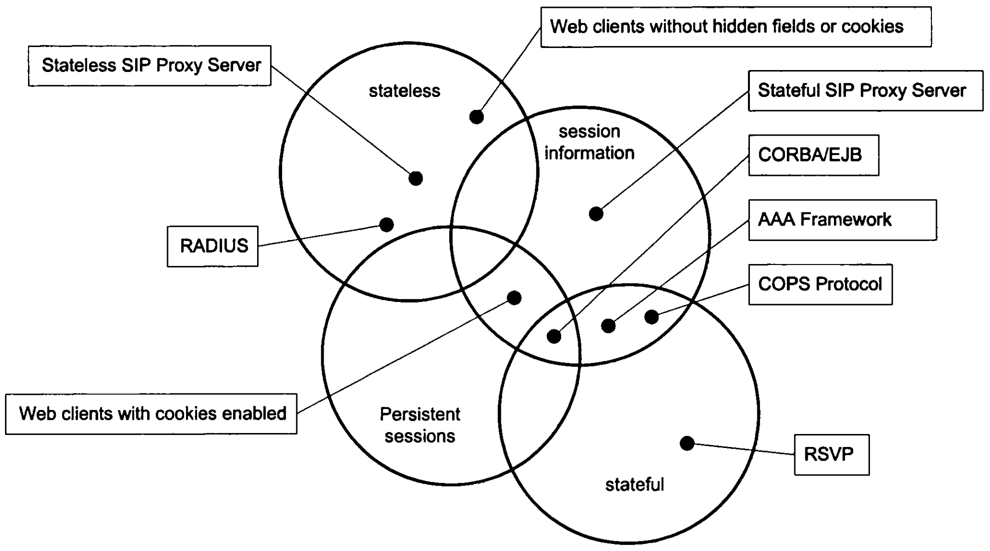

Web clients without hidden fields or cookies
Stateless SIP Proxy Server
stateless
Stateful SIP Proxy Server
session information
CORBA/EJB
AAA Framework
RADIUS
COPS Protocol
Web clients with cookies enabled
Persistent sessions
RSVP
stateful

Figure 6.10: *Classification of state-dependent architectures*

Figure 6.10 represents the interactions of the various architectures. It shows that the CORBA and EJB architectures rely extensively on maintaining state information for their execution. Of course, this is not surprising as the state information that is maintained for these DOTs is required to provide transactional capabilities.

The RADIUS protocol can be characterised as having a low dependence on state. This means that although external messages result in state transitions, there is not heavy reliance or communication between the states. Furthermore, the fact that RADIUS can operate over UDP implies that the protocol is connectionless. By design the protocol does not exhibit a high degree of state-dependent behaviour, otherwise it would have been designed over TCP.

Importantly, RADIUS does not support multiple requests. This implies a single-instance view of the protocol that partly accounts for its simplicity. If RADIUS could support multiple requests concurrently, it would need to provide mechanisms such as threading, queue management and resource management (as the AAA framework does). To provide these, a manager would be needed. Furthermore, the manager would have to be aware of the status of these sub-systems. This can be phrased in a statement that says that the complexities and overheads that arise in maintaining state in any form may be partly attributed to the underlying complexities imposed by the concurrent behaviour of such a system.

In a concurrent or real-time system, a limited pool of resources may need to be allocated to processes requesting these resources. Such a system needs to control access to these resources and to ensure that they are allocated to the requesting processes in a manner that is determined by a scheduling algorithm. In order for the control process to maintain a view on the status of the resources, there must be a way for it to keep track of the status of the resources that are available. Such systems explicitly impose the requirement for state-dependent actions and the need to maintain internal state.

The PSTN is one such system with a limited pool of resources and a large number of processes (individual users) making requests. The IN architecture and the SS7 protocol play a major role in maintaining the robustness of the PSTN. Deeply embedded within both the IN architecture and the SS7 protocols are state models that form an integral part of the control plane.

The AAA architecture that was presented in section 6.3.2 identified the need for the resource manager and its requirements. Some of the requirements that were presented include:

- Sending of notifications from service equipment to the RM,
- Maintaining time limits for sessions,
- Refreshing the time limits and
- Terminating sessions anywhere in the authorisation chain.

Such capabilities reflect closely the functionality provided traditionally by the IN control architecture rather than the control plane of the IP domain. The functionality provided by AAA servers must exhibit the robustness usually associated with the control plane of the PSTN.

The capability of an AAA RM to terminate sessions anywhere in the authorisation chain as identified by [Voll00a] requires that each RM maintain state information such as active sessions and timers for each session. Figure 6.11 shows a graphical representation of one way that an AAA server could maintain a list of the active sessions and the participants in that session.

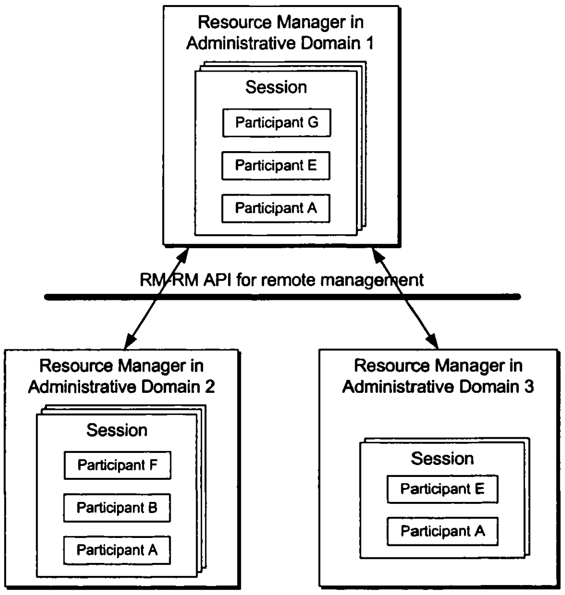

Resource Manager in Administrative Domain 1
Session
Participant G
Participant E
Participant A
RM/RM API for remote management
Resource Manager in Administrative Domain 2
Session
Participant F
Participant B
Participant A
Resource Manager in Administrative Domain 3
Session
Participant E
Participant A

Figure 6.11: *Distributed control with RMs residing across administrative boundaries*

The association of participants with a unique session is similar to the mechanism of the IN CS-2 CPH that was discussed in section 2.4.6.

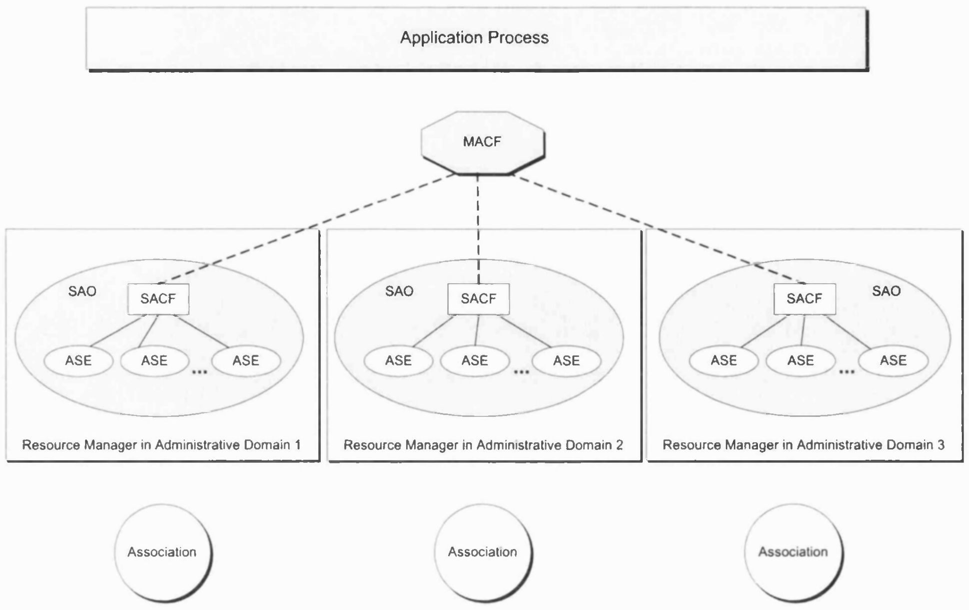

Application Process
MACF
SAO
SACF
ASE
ASE
ASE
Resource Manager in Administrative Domain 1
SAO
SACF
ASE
ASE
ASE
Resource Manager in Administrative Domain 2
SACF
SAO
ASE
ASE
ASE
Resource Manager in Administrative Domain 3
Association
Association
Association

Figure 6.12: *Application entity structure adopted from [Q.1208]*

Furthermore, the controlling RM can be viewed as a multiple association control function (MACF) [Q.1208]. The associations represent the communication pipes through which one AAA RM communicates with another. In the case of AAA, since an RM needs to be aware of all participants in an authorisation chain, this relationship can be implemented using a Single Association Object (SAO).

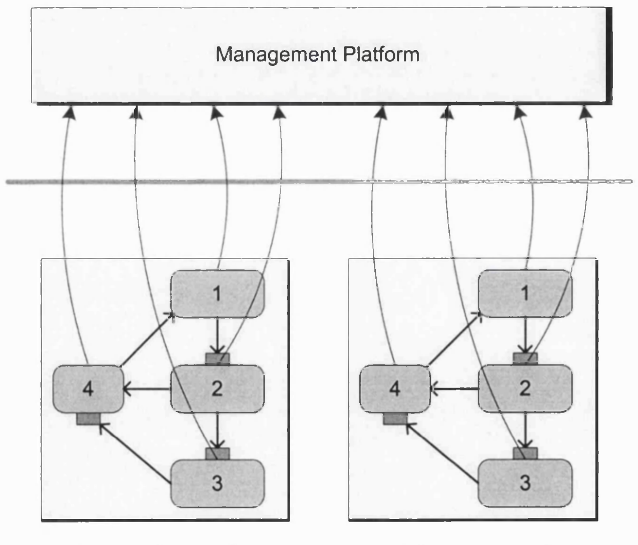


Figure 6.13: *The management layer interface*

There is a further point that needs to be examined – the effect on the system if a management platform needs to have a view on the state of multiple RADIUS servers. This problem is represented graphically in figure 6.13, which shows a management platform communicating with various RADIUS servers. Potentially such a platform could collect information regarding the state of the underlying resources (in this case RADIUS servers), either by requesting the information or by having the RADIUS servers send status information to the management layer.

If such a mechanism is incorporated into the existing RADIUS architecture, it adds complexity to the implementation of the protocol. This problem is further aggregated in the case where the resources may not be in the same administrative domain. For such a scenario a common API must exist that allows vendor-specific resources to communicate with the management layer, or any overlaid layer.

This issue is further discussed in chapter 8, where the architecture for an application server is presented based on the concepts of both the IN architecture and the IP-domain.

6.6 Chapter Summary and Research Contributions

This chapter presented the control constructs for implementing state behaviour, the representation of state and the policies on state-dependent actions. A number of client–server architectures and protocols were then presented and examined with respect to their reliance on state. This was followed by a discussion on the presence of state in distributed object technologies, such as CORBA and EJB. The last section provided a view on state models by incorporating ideas that were presented in the previous chapters.

Through the work that was presented in this chapter, the presence or absence of state in various IP-based technologies was demonstrated in a Venn diagram. A discussion on state models incorporated both ideas from the work presented in this chapter regarding state-dependent actions and principles from the IN control architecture.

Another important issue that can be drawn through the work presented in this chapter is the issue regarding the classification of whether a system is state-based or stateless; this depends on the level at which the system is looked at. As discussed in section 6.3.1, when a single-instance view of a system is examined, it may lead to the system being characterised as stateless. However, once the system becomes distributed through middleware technologies, the system may then be viewed as maintaining state at the middleware layer.

In the following chapter, the implementation of a network management system is presented. The system was implemented by utilising the approaches to state-management that were presented in this chapter as well as from state models present in the IN-domain.

CHAPTER 7

INVESTIGATING STATE MODELS IN THE IP-DOMAIN THROUGH A WEB-BASED NETWORK MANAGEMENT SYSTEM

The work presented in this chapter aims to investigate the notion of state in distributed IP-based systems. For this, a web-based Network Management system is presented. The system was developed for use in the department of Electronic and Electrical Engineering at UCL.

7.1 INTRODUCTION

The Network Management System (NMS) allows administrators to monitor the status of the various devices on their network. The system makes use of the Simple Network Management Protocol (SNMP) [RFC1098] and standard Management Information Bases (MIBs) [RFC1155] [RFC1212].

In this chapter, section 7.1.1 provides the motivation for this work. Following this, section 7.1.2 gives a brief overview of the operation of the SNMP protocol. Section 7.2 then presents the Network Management System and section 7.3 examines the distributed behaviour of the NMS. Finally, section 7.4 provides a chapter summary.

7.1.1 Motivation and Approach

The motivation for the work presented in this chapter was to gain a clear understanding of the issues surrounding "state" and the notion of "state" in IP-based systems. Chapters 2 and 3 examined telecommunications control architectures, such as those of the IN, and signalling protocols, such as SS7. In chapter 6 state-management and the existence of state in IP-based systems was presented.

Therefore, chapters 2, 3 and 6 provided a clear understanding of the use of state models within IN-based as well as IP-based systems. For instance, chapters 2 and 3 demonstrated that state-dependent actions are deeply embedded within the core of the control plane of the PSTN, whereas chapter 6 showed that IP-based systems can be classified both as "stateless" and "state-dependent", depending on the specific viewpoint. To fully examine the role of the control plane and the role in which network intelligence is provided, it is essential to study the notion of "state" by adopting the state-management techniques discussed in chapter 6.

In order to investigate the communication of distributed systems while focusing on the specific issue of state-based and stateless transactions in IP-based architectures and systems the author took the view that a useful way to examine the behaviour of such systems was to design and implement a relatively complex system that is used in a real-time environment, such as the NMS.

In the implementation of the system, the author also believed that a useful way to understand the state-based and stateless behaviour of such systems at the application layer was by excluding middleware technologies that could have taken care of the communication layer. For instance, middleware technologies such as CORBA [CORBA95], DCOM [DCOM][Will94] and Enterprise Java Beans [EJB01] could have been used.

Moreover, a slightly similar approach was to utilise existing software patterns [Gamm95] (see section 6.2.1) to represent state. However, this again would involve some overheads of driving the state models and may have obstructed a clear view of the underlying system.

The approach that was adopted was to implement the system using message-passing to simulate the effects of internal state using external trigger conditions (section 6.2). These messages effectively make up transactions that are used to drive the distributed system from one state to another according to the internal state transition tables. The internal state transition tables are therefore explicitly defined using language-dependent constructs, such as "case" and "if-statements".

The message-passing approach does not maintain **persistent session state**. This means that requests are received, processed and replied to using a request identifier that is not maintained across sessions. For example, if an incoming request is received and the server processing the request does not respond, the client will not re-issue that request, as it does not maintain a list of the sent requests and, therefore, does not maintain persistent state information.

The reason for this lies in the environment of the system. Since this is a Network Management System, if a request is lost and the NMS server responds with a time lapse, the results of the request may not be reliable.*

Having introduced the motivation and approach of the work that is presented in this chapter, the next section provides a brief background to the SNMP protocol.

7.1.2 Introduction to the Simple Network Management Protocol

The Simple Network Management Protocol [RFC1098][RFC2570] is an application layer protocol for the management of network devices. These network elements are called **managed objects**. Associated with each managed object is a defined set of management-related information. This includes variables, also known as attributes, that can be read or written to by the network manager via the network. Figure 7.1 depicts the components of a Network Management System.

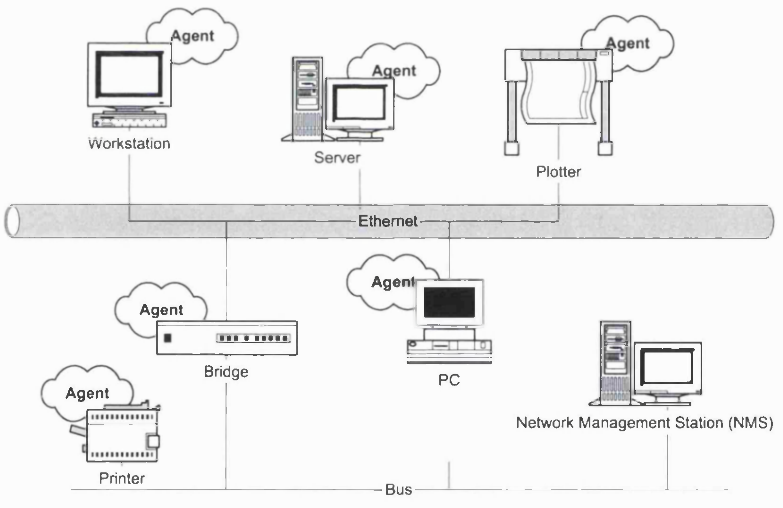


Figure 7.1: *Example of the components of a Network Management System*

The management information associated with a network is kept at the network manager station in a Management Information Base (MIB) [RFC1155][RFC1212]. The MIB specifies the variables that the network elements contain. The variables have unique object identifiers (objectID) and use a hierarchical numbering system. For example, [RFC1213] defines the ipInReceives attribute as:

* This occurs in the case where the client sends a request which is received by the server, which then issues an SNMP request to the SNMP agent. The server obtains the result but, for some reason, does not respond. If the client re-issues the same request with the same request identifier and the data is returned by the server from the cache, it may not reflect the real-time view of the managed device.

```
"ipInReceives OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
 "The total number of input datagrams received from
  interfaces, including those received in error."
        ::= { ip 3 }"
```

This identifies the attribute as a counter, with read-only access (i.e. the network manager cannot change this value), and a mandatory attribute. The hierarchical name for ipInReceives is:

```
.iso.org.dod.internet.mgmt.mib-2.ip.ipInReceives
```

while the numerical value corresponding to the same object identifier is:

```
.1.3.6.1.2.1.4.3
```

A network management station is used to query this information for each device in figure 7.1. Such queries are handled by agents located on each of the managed objects. SNMPv1 [RFC1098] defines five message types whereas SNMPv2 [RFC1441] defines an additional two requests, as well as two new MIBs: the SNMPv2 MIB [RFC1213] and the SNMPv2-M2M MIB (Manager-to-Manager). A full description of the SNMP protocol can be found in [Stal99].

7.2 THE NETWORK MANAGEMENT SYSTEM

The Network Management System (NMS) is composed of the following main subsystems:

- Web Server Manager,
- Applet,
- Network Management System,
- SNMP Agents and
- Database Agents.

At a high level, the aim is to allow an applet to access the managed objects.

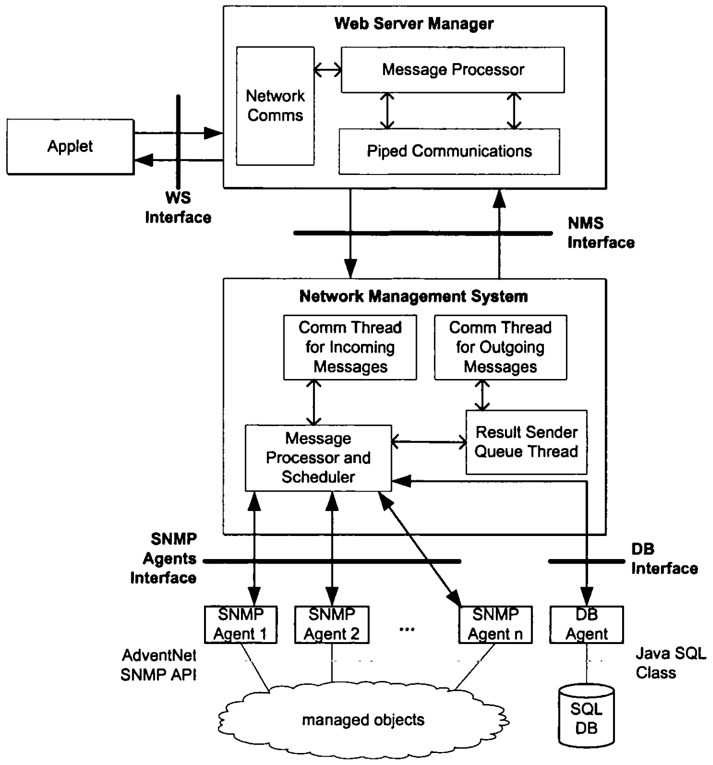


Figure 7.2: Interfaces of the NMS

The web server provides the communication between the applet and the NMS. When the administrator logs onto the system, the applet presents a floor-view* of the requested floor. The floor maps represent a top-down view of all managed objects for that floor. This includes devices from printers to workstations, ports, hubs, routers and PCs.

The NMS controls the interface between the information that the web server requires and the information that the agents can provide. It controls the agents and updates the web server so that the web pages reflect the underlying status of the managed objects. It does this by polling the agents at regular intervals so that the data is updated automatically. Furthermore, the NMS is responsible for controlling the SNMP agents and their polling intervals, as well as setting SNMP traps when such requests are received from the web server.

The database agent provides an interface to the database containing information about all the managed objects. For each object, information in the database includes its location (i.e. room), the IP address (if appropriate), as well as the MIB file that corresponds to that device.

* A floor-view represents a literal top-down view of a specific floor, and the colours of the devices on that floor represent the status of the devices.

The SNMP agent is responsible for collecting the information from the managed object. When an object is constructed, it accesses the SNMP daemon for that device. The object then uses the objectID to query the DB agent, which loads the appropriate MIB for that managed object. For example, if the SNMP agent is responsible for a router, then the router MIB provided by the manufacturer will automatically be loaded. The SNMP agents are threaded objects that are controlled by the NMS system and which can access the specific attribute from the managed object by using the AdventNet SNMP API [Advent].

7.2.1 Communication across the Components

Communication between the web server and the NMS is implemented using pipes, as illustrated in figure 7.3. Communication between the applet and the web server uses sockets, with the web server creating a Server socket.

The applet and the NMS are therefore responsible for controlling separate threads for incoming and outgoing messages; the web server establishes a connection with both the applet and the NMS and hence requires four threads (two for incoming and two for outgoing).

Having given a brief description of the communication between the system's components, the individual interfaces (figure 7.2) are discussed next.

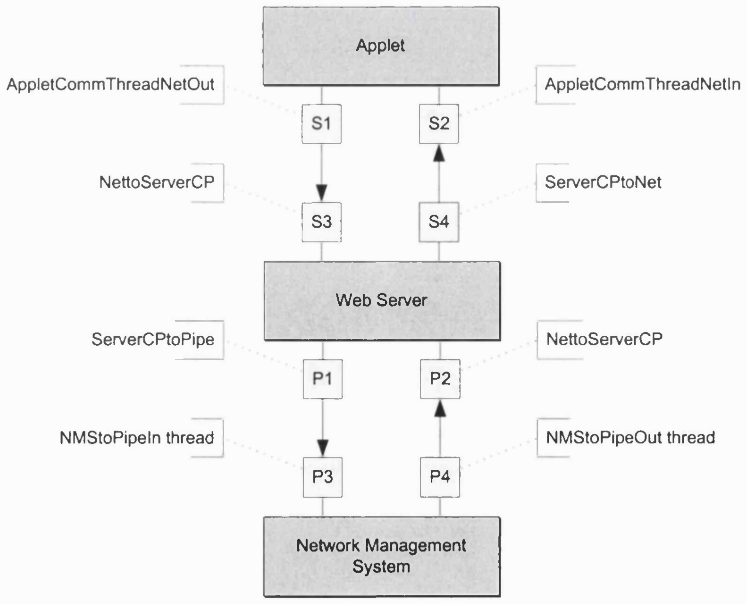
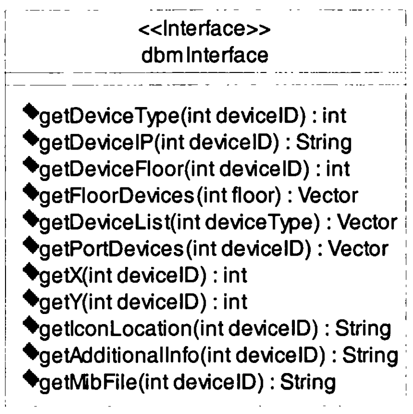


Figure 7.3: *Communication between the applet, web server and NMS*

7.2.2 The DB Interface and the DB Agent

The database interface provides access to the database through the agent. Figure 7.4 shows the class diagram for the DB Agent.

The methods made available by the class allow access to the relational SQL database, which contains all managed objects. Two methods whose function may not be apparent are getX and getY. These provide the x-coordinate and y-coordinate to the applet. Essentially the applet generates an image of the floor based on these coordinates, which are entered by the administrator.

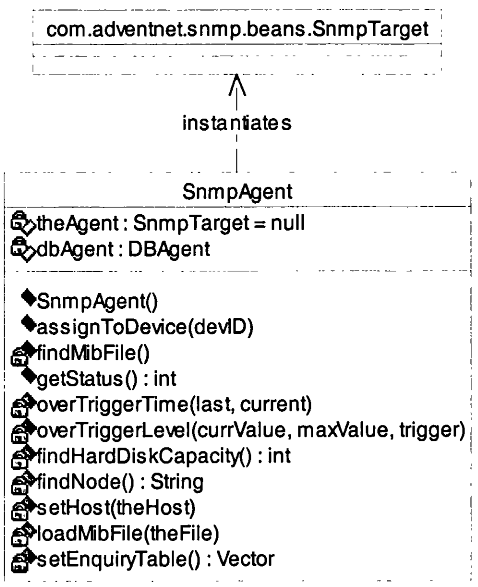


<<Interface>>
dbmInterface
getDeviceType(int deviceID) : int
getDeviceIP(int deviceID) : String
getDeviceFloor(int deviceID) : int
getFloorDevices(int floor) : Vector
getDeviceList(int deviceType) : Vector
getPortDevices(int deviceID) : Vector
getX(int deviceID) : int
getY(int deviceID) : int
getIconLocation(int deviceID) : String
getAdditionalInfo(int deviceID) : String
getMibFile(int deviceID) : String

__Figure 7.4:__ The DB Agent class

7.2.3 The SNMP Agent and the SNMP Interface

The SNMP agent class uses the SnmpTarget class from the AdventNet API [Advent]. The public methods of the class essentially allow the NMS to assign a new agent to a specific device using its devID. The getStatus method informs the NMS of the status of the device.



com.adventnet.snmp.beans.SnmpTarget
instantiates
SnmpAgent
theAgent : SnmpTarget = null
dbAgent : DBAgent
SnmpAgent()
assignToDevice(devID)
findMibFile()
getStatus() : int
overTriggerTime(last, current)
overTriggerLevel(currValue, maxValue, trigger)
findHardDiskCapacity() : int
findNode() : String
setHost(theHost)
loadMibFile(theFile)
setEnquiryTable() : Vector

__Figure 7.5:__ The SNMP Agent class

A detailed explanation of the AdventNet API and MIB Browser tool can be found in [Advent].

7.2.4 The NMS Components

It can be seen from figure 7.6 that five of the classes that make up the NMS extend the java.lang.Thread class. This enables the NMS to deal with multiple requests from the applet; it also monitors the managed objects at the requested polling intervals.

A second point to note is that the outgoing messages from the NMS are sent by the ResultSender class. This works as follows. When the concurrent threads of the NMS need to send messages to the web server, the ResultBuffer.put() method is called by the thread and places the message in a buffer. The ResultSender's run() method invokes the ResultBuffer.get() method at intervals. When a message is returned by the ResultBuffer.get() method, it is sent using the communication pipes discussed in section 7.2.1.

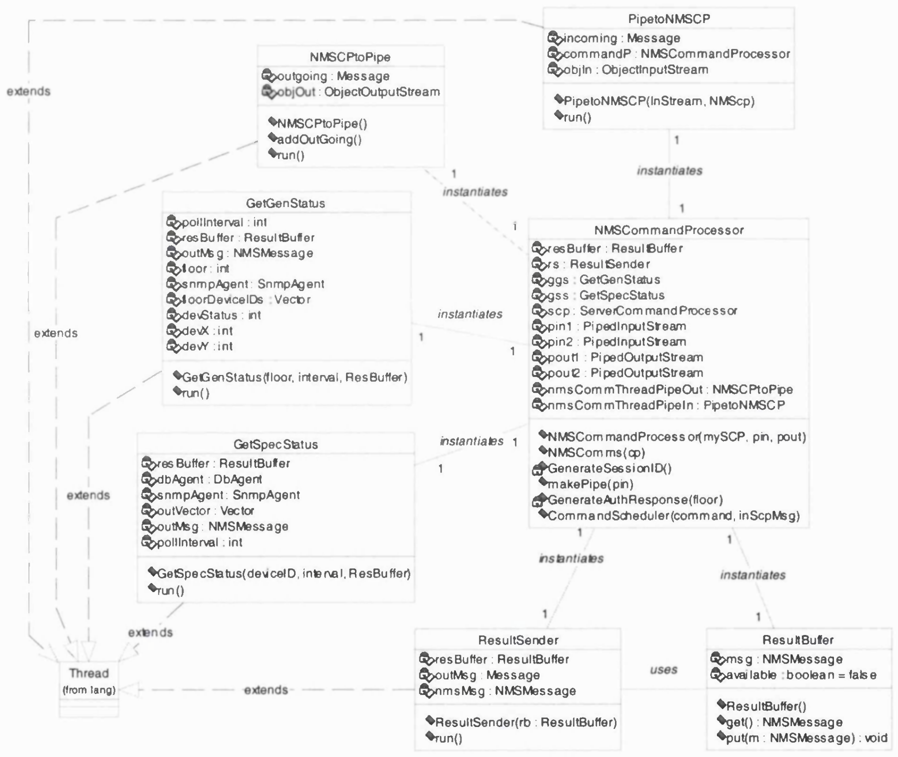


Figure 7.6: The NMS

7.2.5 The Web Server and the Applet

The structure of the web server closely resembles the structure of the NMS, i.e. there is a CommandProcessor (in both the web server and the applet) that is responsible for handling incoming and outgoing requests. Where a response is sent from the NMS and needs to be directed to the applet (without the intervention of the web server) the web server simply passes the incoming message onto the outgoing thread towards the applet.

The earlier sections described the network management system, the communication among the components, the internal details of the NMS components, as well as more "lightweight" parts such as the database agent. The next section looks at the implementation of the system from the point of view of the communication mechanisms adopted as well as state behaviour of the key controlling parts.

7.3 Distributed State-Based Behaviour of the NMS

The objectives behind implementing the NMS system can be summarised as follows:

- To design a robust system by using extensive state models,
- To define well-structured and clear interfaces between the individual components and
- To understand how a classical three-tier architecture operates in the IP domain.

One of the architectural characteristics of the NMS is the fact that the sources of information are distributed. The system queries distributed SNMP agents to obtain the status of various devices scattered across the network. The NMS must be able to collect the information, examine the configuration under which it is running, and return the results in a meaningful manner to the applet that is making the request.

To achieve this, the request must go through a web server (figure 7.2), reach the managed object and return the result. In this system, the initial constraints were imposed by the capabilities (and limitations) of existing protocols and architectures. For instance, the information collected from the devices was limited to what the MIB could provide, as well as by the capabilities of the SNMP protocol. Also, there are the security considerations in the communication between an applet, its host, and the web server. As a result, the information returned by the web server to the applet had to be in a format that the applet could process without requiring any additional resources that would break the limitations imposed by the security model of applets.

The core part of the system had to allow the transfer of information from the applet to the SNMP agent and back. The internal functional modules had to be designed in a way that would enable them to communicate internally and externally. The external communication of the system is defined by the interfaces identified in figure 7.2. The following interfaces were developed for external communication between the systems:

- The WS Interface, between the applet and the web server,
- The NMS Interface, between the web server and the NMS core manager and
- The DB Interface, between the NMS core manager and the SQL database.

The internal interfaces that were developed allowed the Message Processors and Schedulers of each of the components to send internal requests. These requests are powerful and vary from requests that initiate queries to the SNMP agent to management requests that query the status of existing requests. As a result, through this interface, it is possible for the existing core NMS to be deployed in a distributed manner. For example, while on one host there would be the command and message processor, another would handle the management-type requests.

The modularised approach utilises well-defined messages for the internal communication of the system. Therefore, it is possible for the overall architecture of the system to be re-used and implemented in a completely different environment. For instance, the SNMP interface can now be replaced with a GSM interface for obtaining information relating to GSM subscribers.

The internal message-passing mechanism can also be further extended to include ideas from the IN world. For instance, when an external request is received, the internal status of the system is examined before passing it to the appropriate sub-system that deals with the implementation of the request. To achieve this, internal state within the system must be maintained. This internal state would then be examined by the sub-system once a request is received and, if the internal state is satisfactory, the request is executed otherwise it is returned. For the implementation of internal state, a similar approach to the IN model would be utilised using detection points and points in call (section 2.4.4).

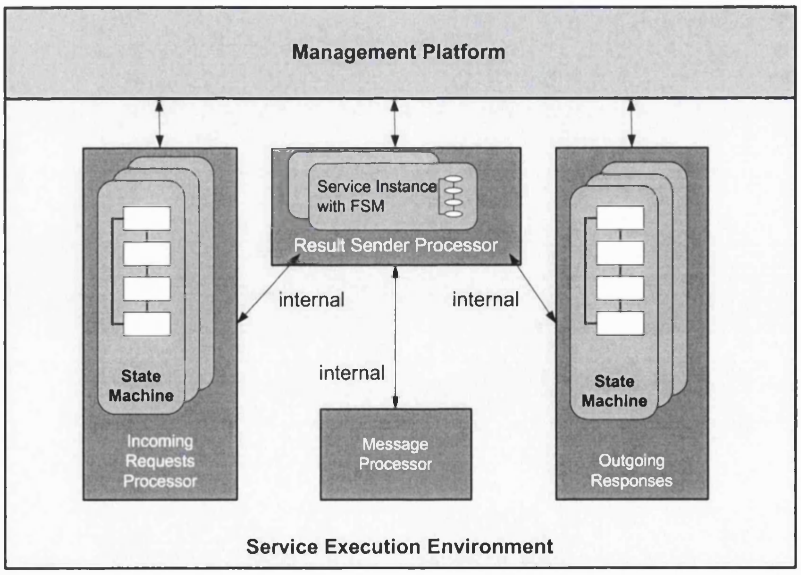


Figure 7.7: *Extensions to the NMS architecture*

Figure 7.7 depicts an extension to the basic architecture of the NMS system. Shown are the processors for the incoming and outgoing requests, the internal message processor, and the results sender. On the top, there is the management platform, which communicates with the service execution environment.

The extensions are introduced by the state models that are present in the incoming, the outgoing and the result sender components. These state models are introduced in order to maintain the internal state of the system, in a similar manner to the Basic Call state model in the IN.

Chapter 8 examines in detail the applicability and extensibility of this architecture as a generic application server that draws from IN principles, but also from the state-dependent discussions of chapter 6.

7.4 Chapter Summary and Research Contributions

This chapter examined the notion of "state" by implementing a distributed IP-based network management system. The approach that was used excluded additional complexity or possible overheads of distributed middleware technologies. The distributed communication utilised message-passing.

One of the initial design decisions that had to be made was to decide how to distribute the system, to examine the communication between the various individual parts, but also to make sure that the system maintained its robustness throughout its execution. As a result, ideas from the IN world, which enabled distributed parts to communicate in a clearly defined manner, were incorporated.

The implementation of the system provided the opportunity to realise in practice the effort that is required in maintaining a system in a valid state that is capable of processing incoming requests but also supports basic fail-safe techniques.

Chapter 8 examines in detail the need for a generic application-server in an open service creation environment. The chapter also discusses the architecture for such as an application-server, based both on telecommunication and computer-science principles.

CHAPTER 8

OPEN PROGRAMMABLE NETWORKS, APPLICATION SERVERS AND NETWORK INTELLIGENCE

This chapter looks at service provisioning in an environment where third parties can access core network components through specially designed interfaces and identifies issues that need to be resolved. One specific problem is the interconnection of service providers. The chapter proposes a solution to this in the form of an application server for third-party service providers.

8.1 INTRODUCTION

The interest in open programmable networks is gaining momentum. This can be seen by the increase in the number of members of groups such as Parlay and also from the increase in the number of conferences and publications in the field.

The traditional telecommunications environment has been one where the network has been under the exclusive and strict control of the incumbent operator and revenue was generated by services that were conceived, designed, implemented and managed by the operator. There is now a move towards providing open access to network components through APIs [Moye01] [Bisw98]. This necessitates the investigation of new functional entities, such as gateways, that enable this interconnection. These issues are the focus of the work presented in section 8.2, where this change in the telecommunications environment is described.

Section 8.3 presents an API Server architecture and describes its functional layers, its interfaces and its state models. The work presented in the section has already been presented by the author at major international conferences.

The API Server, after initial work, was specified in the Specification Description Language (SDL) [Z.100] and simulated using Message Sequence Charts (MSC) [Z.100] [Ekka95]. The aim of this was to investigate whether it was feasible to incorporate the proposed state models of the API Server within the Parlay framework. This work is presented in section 8.4. Section 8.5 provides a chapter summary and research contributions.

8.2 Open Networks

This section provides the reasoning behind allowing core network functionality to support independent software vendors and third-party service providers. It complements section 3.3.2.4, which presents two main approaches for providing APIs that open up the network.

8.2.1 Traditional View on Service Implementation

The traditional approach to telecommunications has been that the incumbent network operator has total and exclusive control over the network. This is illustrated in figure 8.1.

The figure shows a number of network operators and describes in detail the business domain of a single network operator. In this single-instance view, the network resides at the core and consists of control-plane architectures, such as the IN, and signalling networks, such as SS7. The network resources represent the components that are needed by many applications. The external circle represents the traditional view of the business domain for a network operator, one where the focus was the network. All three tiers are owned, maintained and operated exclusively by network operators.

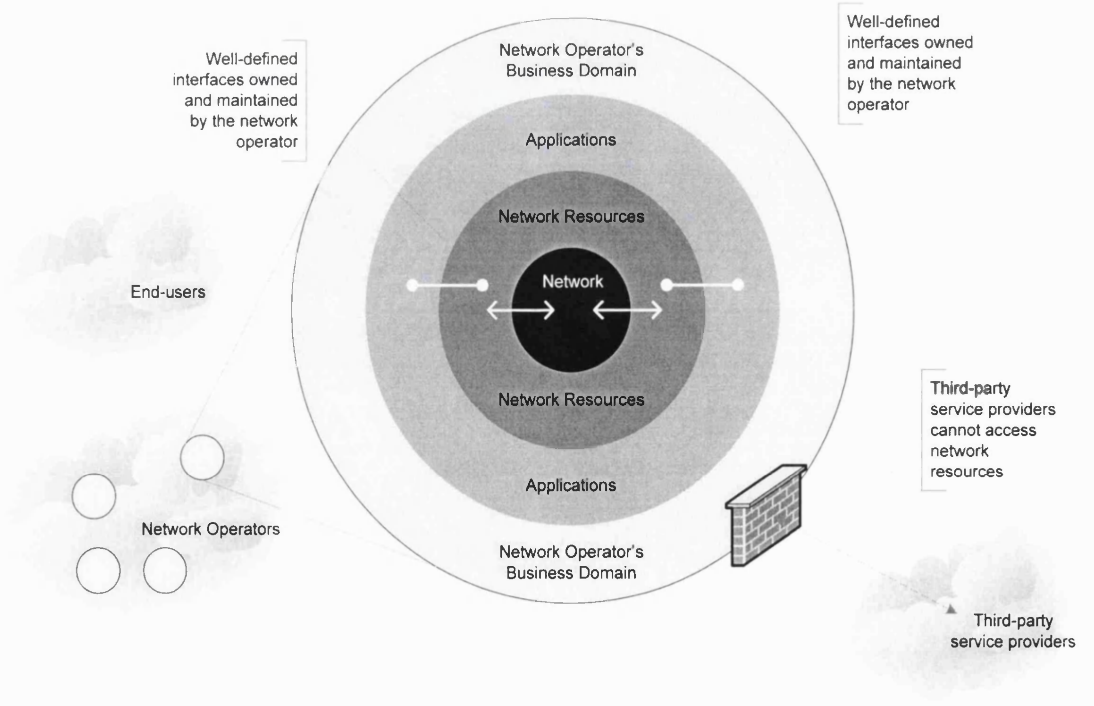

Well-defined interfaces owned and maintained by the network operator
Network Operator's Business Domain
Well-defined interfaces owned and maintained by the network operator
Applications
Network Resources
End-users
Network
Network Resources
Third-party service providers cannot access network resources
Applications
Network Operators
Network Operator's Business Domain
Third-party service providers

Figure 8.1: *Traditional approach to service provisioning*

A number of network operators exist within the network operator cloud and network interconnection enables communication between different subscribers rather than the development of services that span multiple network boundaries. Such services are developed by third-party service providers; the wall in figure 8.1 represents the fact that third-party service providers cannot access network components within the network operator's boundary.

This approach has resulted in network-centric communications for service delivery. The monolithic, network-centric approach does not allow services to access data in the enterprise domain for decision-making [ParlBuss99]. This is because the implementations of IN services run in the network domain, at the core of the network, and as a result, any third-party involvement in service programming is limited to customising only a small set of operational parameters [Laza97]. The network-centric approach is sufficient for mass-market applications [ParlBuss99] where there is a business case for wide appeal, such as the Freephone service, but unfortunately it is not sufficient for smaller applications. Furthermore, in the network-centric approach, the services are relatively easy to manage and can be built in a very robust fashion. There are also fewer security considerations to deal with [ParlBuss99].

However, there are significant disadvantages to the traditional approach. Today's converging telecommunications environment requires a new approach; end users are slowly demanding an interoperable environment regardless of the transport network being used for a service [Tarj97] [Solo00b]. In the traditional approach, the network operator is responsible for the creation, operation and management of all applications and, as a result, it is difficult to achieve the necessary flexibility to deploy many customised versions of services to different customer groups. This leads to a long time-to-market for new applications [Chen00] [ParlBuss99] [Laza97].

The move towards an open network environment can be described as an evolutionary step. The network operators have established a resilient core network infrastructure and it is in their interest to generate further revenue from core network components by enabling open access to elements such as SCPs, HLRs and location servers. It is the author's view that no network operator would consider allowing open access unless it were deemed to be profitable, although regulatory pressure has also acted to direct operators in this way [ECD98/10/EC].

Consequently a new approach that combines the benefits of the network-centric approach with the flexibility of the edge of the network approach is desirable. Of course, in enabling such open access, the integrity of the network must not be hindered [Alex98].

8.2.2 Open Network Access through APIs

The approach taken by advocates of Application Programming Interfaces (API) such as Parlay is to open up access to the various protected functional entities (FE) of operators to third-party service providers using carefully defined class libraries.

The concept of APIs is in essence very simple and requires very little group agreement or standardisation. This is in stark contrast to the traditional telecommunications approach, which is to standardise almost every aspect of the hardware, software and protocol development. Regardless of this, the case in support of opening up core network components has been accepted by incumbents, such as BT and AT&T. This is supported by the fact that BT is a founder member of the Parlay Group. Although APIs significantly simplify and open up access to the telecommunications equipment of operators to third-party service providers, some significant problems still remain:

Firstly, the operator needs to ensure that the access offered to third-party service providers is used in a manner that does not in any way hinder the integrity of the network [Alex98]. To emphasise this, the work presented in [Ward95] identifies issues relating to degradation of network integrity from the simple interconnection of networks, which does not enable open access to key network components. If interconnecting networks may have an impact on network integrity, then open access to network components must use architectures that can guarantee the robustness and operation of the network.

This leads to the second issue of maintaining the integrity of the interface. This means that the interface is used in a manner that conforms to its description. For example, if a service offering requires that certain calls are made in a manner described by a sequence diagram, the operator must ensure that requests that do not conform to the sequence diagram are dropped. Furthermore, in order to preserve the integrity of the network the third-party service providers in the enterprise domain must use the interface in a manner that is compliant both with the capabilities of the interface and the service level agreement with the operator. More importantly, the network operator must ensure that third-party service providers make use of the API in a foreseeable manner and, even more so, within the proper sequencing invocations which are acceptable to the API.

A third issue deals with billing for the services [Solo00b]. Apart from the business issues regarding the billing of services, there is the question of how the billing information is conveyed to the operator's platform. Moreover, mechanisms may need to be provided that allow the service providers to inform other service providers if a certain service is not available.

A key element to API technologies is the gateway. This is a server architecture that provides the interface between client requests and the network services. Whilst it is in the interests of simplicity and ease of implementation for the client–server interface to be as simple as possible, special consideration must be given to the interface to maintain security, integrity, scalability and general manageability of what is essentially a fragile access to precious network resources.

Equally important is the problem of maintaining session state [Solo00b]. Services that make use of network functionality are likely to be much more complex than simple client–server-type transactions. Within a session, services are requested by a client through code written using the operator-supplied APIs. The interactions between the client and the server may be hidden by the software interface, however the interaction cannot be ignored. There needs to be supporting structures that maintain the session state for a service with little or no user involvement.

Another important consideration is the scalability of service provider interactions [Solo00c] [Solo00d]. One client of a service provider may be the service provider for another client. This creates a hierarchy of service provider–client interactions that could get quite complex. Creating services and managing them is a complex task that requires new tools.

8.2.3 Hierarchical Third-Party Service Provisioning

In a hierarchical inter-working scenario an architectural framework that allows third-party service providers to share their APIs, without any loss of security, efficiency and integrity, is desirable. This framework could be extended to whatever degree of the hierarchy is thought relevant to the business case.

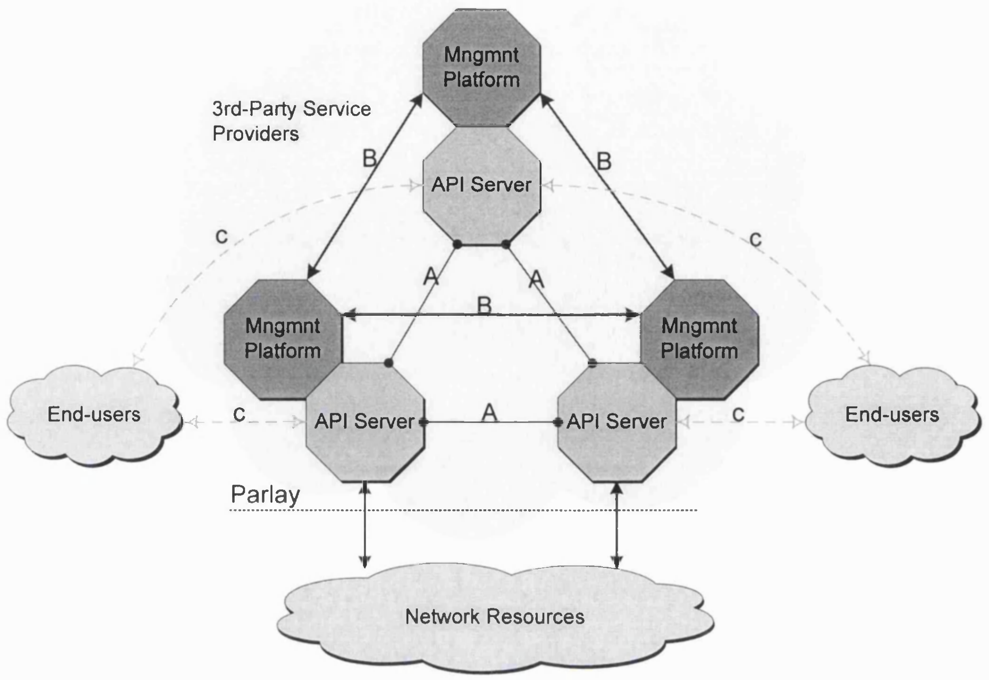


Figure 8.2: *Hierarchy of third-party service providers*

Figure 8.2 provides a graphical view of such a hierarchical scenario. Two third-party service providers (3SP) are using the Parlay API to access core network resources. In turn, proprietary functionality is introduced and offered in the form of new services to other 3SPs. The fact that the 3SP space is likely to be populated by a number of service providers is represented by the cloud in which the service providers are placed. Under this hierarchical scenario, mechanisms to maintain a uniform interface across service providers need to be implemented [Solo00b]. This is the application server architecture (API Server) that is presented in section 8.3.

The API Server consists of two parts: the API Server and the management platform. The API Server provides controlled access to the services whilst the management platform monitors the service lifecycle. The management platform communicates with another management platform of clients and servers interacting with the API Server. The aim is to provide a platform that is resilient to the problems inherent in the services on offer as they are developed and put through early deployment. The management platform allows other 3SPs to send requests that deal with service availability and cost and to change QoS parameters.

In figure 8.2 interface A allows service requests to be sent across API Servers. Requests across this interface are likely to be sent using CORBA [CORBA95] or EJB [EJB01]. Interface B allows the management platforms of different 3SPs to communicate and exchange management-type information. The management interface allows managers to inform each other of problems detected in the service. For example, if a deadlock situation occurs that is not detected by the service program, timers in the management platforms inform each side of the deadlock and trigger an exception. The management platforms can also trigger on other criteria such as frequency of requests and requests that are inappropriate. Interface C represents the service-specific requests received from end users. In most cases, the functionality offered across Interface C is likely to be a subset of the functionality provided by Interface A.

8.3 IN-Based Application Server

Traditionally, an application server is a platform that provides an environment where services can be executed. It may also provide a management layer for the manipulation of the services that are executing. Commercially available application servers include WebSphere by IBM and the Borland Application Server by Borland.

The proposed application server is not aimed at providing functionality that is already present in existing application servers. The aim of this IN-based application server is to provide features that in the author's view are desirable for such a platform and indeed within the operating environment. As such, existing application servers may decide to incorporate functionality that is proposed here.

The API Server architecture draws from the control plane of the IN and also from the flexibility and advantages that have arisen from the IP community.

8.3.1 Why IN-Based?

The concept of state models and state model behaviour in telecommunications has already proven essential to management and network intelligence applications of the core network. The IN control architecture and, in particular, the basic-call process and its state model, the BCSM, is possibly the most important state model in telephony, upon which the majority of IN services are based.

Call party handling (CPH) was discussed in section 2.4.6. The capabilities provided by CPH (such as the merging and splitting of call segments) are achieved at the expense of very complex and well-defined interactions between the switches involved. The concept of Connection View States in CS-2 (section 2.4.5) provides a way to capture all the possible configurations under which call parties can be connected to each other in a single state machine.

It is the author's view that the CPH capabilities of IN CS-2 provide a powerful way to describe the behaviour of complex multi-party and multi-service interactions. These ideas are incorporated into the definition of a similar mechanism in the API Server. One of the main advantages of such a state model is the resilience and the proven track record of the IN model. Of course a state-model approach imposes overheads in terms of managing the model; however, it is critical to maintain a guaranteed level of service for such an API Server.

To enable the API Server to have a view of the state of the services that are executing, a number of initial service segments have been identified. Service segments are grouped in a similar manner to the IN CPH operation (section 2.4.6): the API Server maintains a service segment association that contains service segments that describe the state of each instance of the service. The following initial service segments were identified as potentially useful states:

- Null State: This is the initial state for any new service.
- Stable Service Execution: A call segment association containing a service segment in this state indicates that the service is executing.
- Service Request: This state indicates that a new service request is received by the API Server.
- Ordered Request: This state indicates that certain actions need to be executed in a specific order.

8.3.2 API Server Logical Interfaces

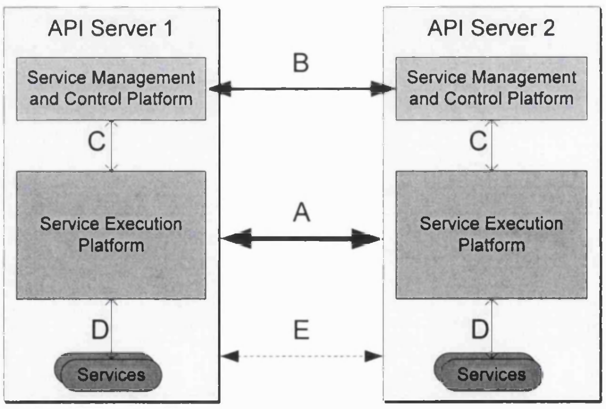

API Server 1
API Server 2
Service Management and Control Platform
Service Management and Control Platform
B
C
C
Service Execution Platform
Service Execution Platform
A
D
E
D
Services
Services

Figure 8.3: *Logical interfaces of the API Server*

The logical interfaces across two API servers are shown in figure 8.3. The A and B interfaces, introduced in section 8.2.3, are included here for completeness. The logical interfaces are as follows:

- Interface A enables one service provider to provide service functionality to another.
- Interface B is the interface between the service management platforms of two API servers. This interface provides advanced functionality such as the remote simulation of services.
- Interface C provides for internal communication between the service execution platform and the service control and management platform. It allows the service execution platform to

inform the service management platform of status information, such as the number of executing services, authentication violations and service agreement violations.

- Interface D allows local service implementations to inform the service execution platform of important events. For example, it enables a service to interrogate the service control platform for information pertaining to the status of the API Server.
- Interface E is the application server access API that is responsible for authentication and service discovery.

8.3.3 API Server Functional View

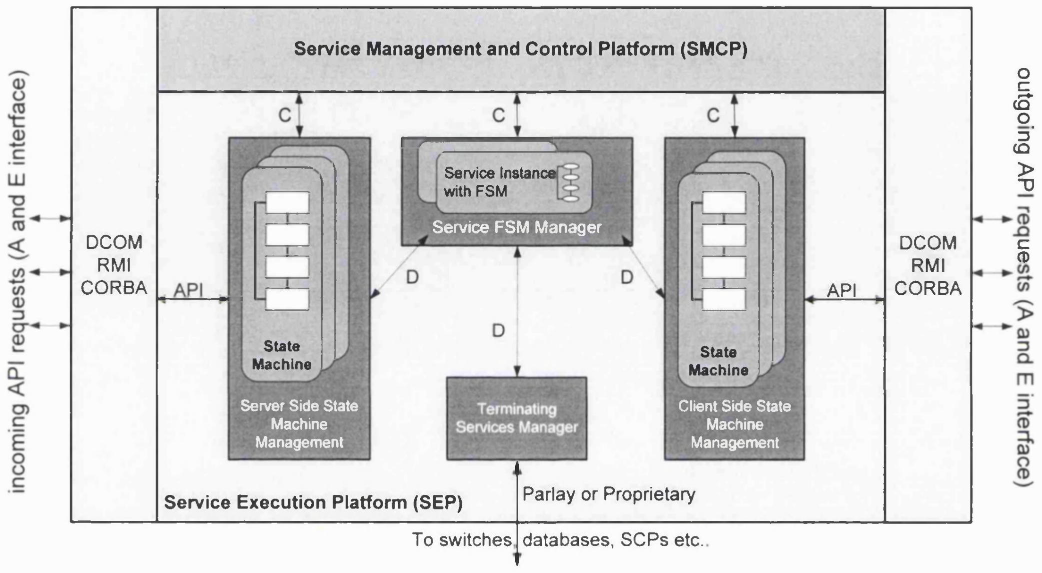

Service Management and Control Platform (SMCP)
C
C
C
Service Instance with FSM
Service FSM Manager
incoming API requests (A and E interface)
outgoing API requests (A and E interface)
DCOM
RMI
CORBA
API
D
D
D
State Machine
Server Side State Machine Management
Terminating Services Manager
Client Side State Machine Management
Service Execution Platform (SEP)
Parlay or Proprietary
To switches, databases, SCPs etc..

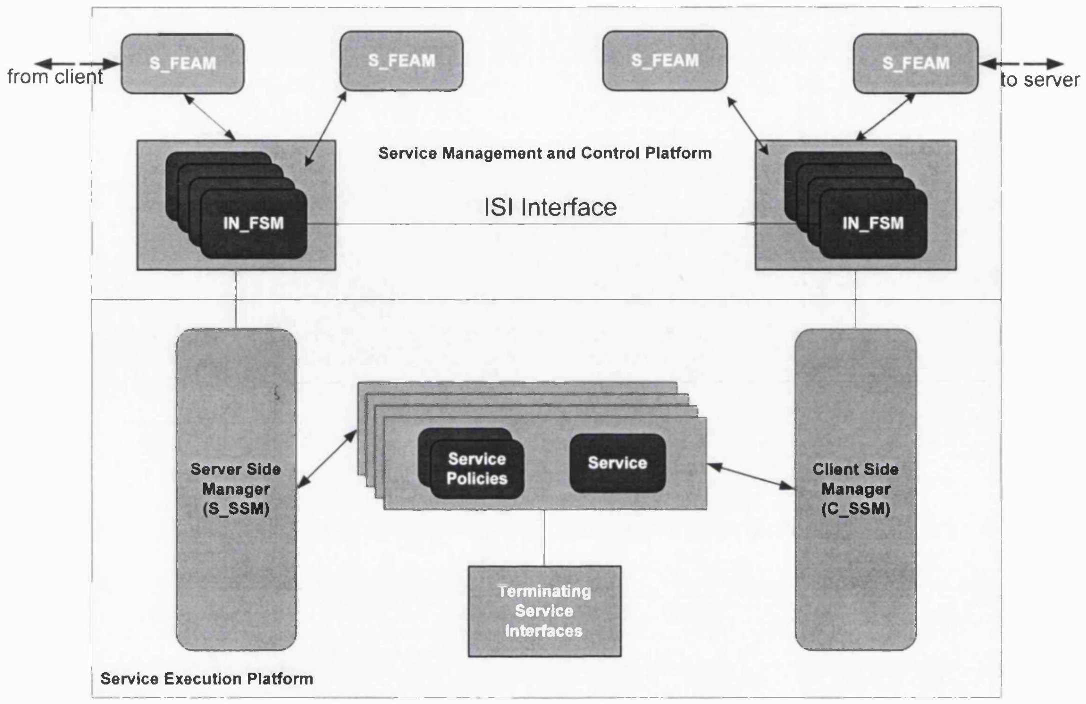
Figure 8.4: *The API Server architecture*

Figure 8.4 presents the functional composition of the API Server [Solo00b], which can be divided into the service execution platform (SEP) and the service management and control platform (SCMP). Access to the server is made possible through distributed technologies such as DCOM, CORBA or EJB.

The service execution platform contains all the functional elements necessary for the execution of service logic [Solo00c]. There are two state models, one for server requests (received from other service providers) and one for client requests (forwarded to other service providers). The server-side state machine manager (S_SSM) is responsible for handling incoming requests. These cause instances of services to be created and are managed using the CPH approach adopted by IN CS-2. Where requests need to be forwarded to other service providers, a client-side state machine (C_SSM) is created. Both the server and client SSMs are based on the IN BCSM.

The Terminating Service Interface allows access to proprietary requests. These are incoming service requests that can be served without a supplementary request to another service provider and are handled by the Terminating Services Manager.

In addition to the server and client SSMs, the API Server requires IN-type FSMs in order to achieve CPH-based behaviour [Solo00d]. These are depicted in figure 8.5, which is an extension of figure 8.4 and also shows the S/C_FEAMs and the IN_S/C_FEAMs.



from client
S_FEAM
S_FEAM
S_FEAM
S_FEAM
to server
Service Management and Control Platform
IN_FSM
ISI Interface
IN_FSM
Server Side Manager (S_SSM)
Service Policies
Service
Client Side Manager (C_SSM)
Terminating Service Interfaces
Service Execution Platform

Figure 8.5: *Details of the API Server architecture*

There are two instances of the server and client SSMs. When a new service request is received a service instance is created, together with the service policies. The server-side state machine monitors the activities of the server in the execution of a service: it goes through the lifecycle of instantiation, message passing, and termination and billing. The state model also implements points in call to allow the triggering of other services. The state machine is thus given some flexibility for manipulation similar to IN event and trigger detection points, although the supplementary services that this might facilitate are as yet undefined. The client-side state machine represents the client in the instantiation and message-passing phase of a service. The client-side is not essential in all cases. A client-side may not be implemented if end user applications choose not to implement it (although this is not recommended) or if a service request is a terminating service.

Information held as part of a Service Policy includes supporting and conflicting services (which provide information on feature interaction), service order execution (necessary in situations where services must be executed in a specific order) and service timeout policies (required if the reply from a service may be needed for further processing). The service policies also contain information that indicates whether the particular service needs to initiate supplementary (i.e. dependent) services.

The IN_FSMs contain service segments and service segment associations (section 2.4.6). The S_FEAM creates new service segments (SS) and service segment associations (SSA) in a similar manner to the CPH model. The Inter-SSM Interface (ISI) allows communication between originating and terminating BCSM-type models. A direct link from the API Server to an IN CS-2 compliant node can be achieved by direct access to the IN_S_FEAMs and IN_C_FEAMs.

8.3.4 API Server State Model

The state model that describes the overall behaviour of the API Server is presented in figure 8.6. The state models that describe the behaviour of the API Server for each individual service request (incoming and outgoing) are described in the following sections.

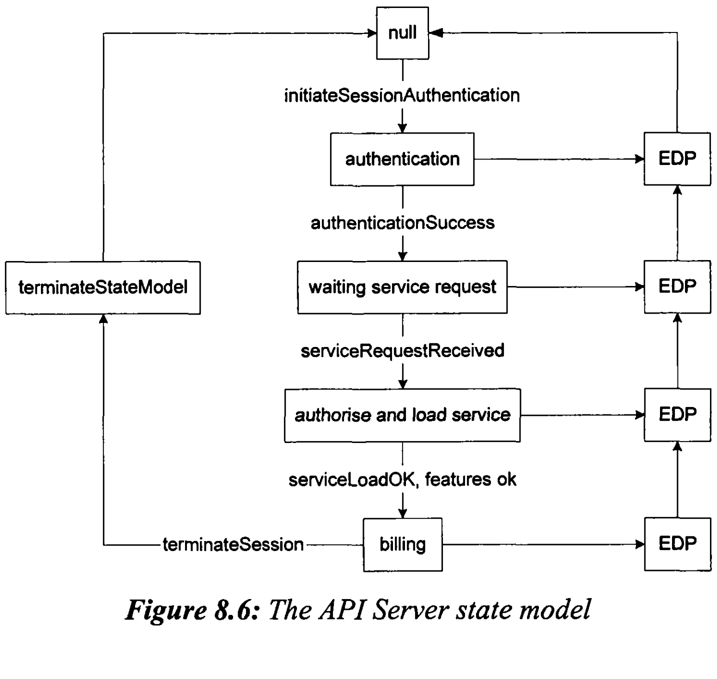


Figure 8.6: *The API Server state model*

A third-party service provider must be authenticated and authorised prior to issuing any service requests. In figure 8.6, following the null/idle state when the *initiateSessionAuthentication* request is received, the API Server moves to the authentication state. Here, the requesting third party is authenticated. A failed authentication results in an exception detection point (EDP). A successful authentication is followed by an authorisation check (which is part of the authentication state) and the *authenticationSuccess* event is generated internally.

When the *authenticationSuccess* message is received internally, the API Server moves to the *waitingServiceRequest* state. Here, the API Server awaits incoming service requests; when one is received, it moves to the *authoriseAndLoadService* state. Here, the service level agreements are checked from the management platform, to ensure that the requesting party is authorised to initiate the specific service requests.

The API Server then moves to the billing state and applies the billing is applied according to the service level agreement between the third-party service providers. Billing is an important element of the proposed state models. Existing application servers do not model this state and cannot, therefore, have a clear view (from a billing perspective) on the underlying services that are executing on the platform.

8.3.5 Server State Model

The server state model, presented in figure 8.7, is set up when the API Server receives an incoming service request from a third-party service provider.

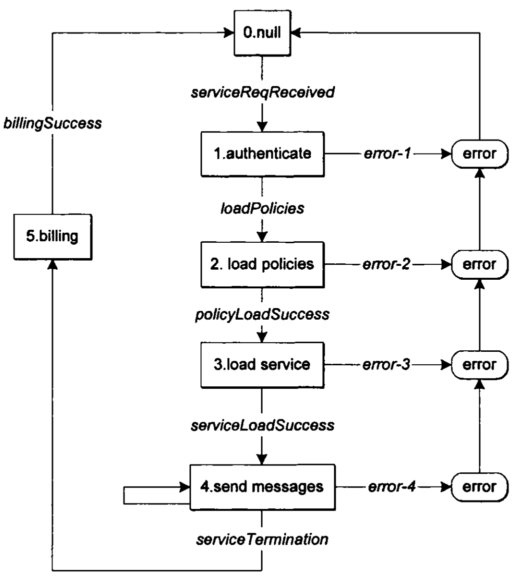


Figure 8.7: *The Server state model (S_SSM)*

The first state, authenticate, models the condition that an incoming request is authenticated. The issuing service-provider must have an agreement with the API Server prior to issuing any service requests. An authentication failure leads to the state model moving to the error-1 state. A successful authentication causes the *loadPolicies* transition and the state moves to the *loadPolicies* state.

Within the *loadPolicies* state, several metrics pertaining to the specific service are loaded. Service policy metrics include the following:

- *timeOutValue* – the timeout values for the specific service
- *processingPowerQuantifier* – a value denoting the processing power needed for the service
- *conflictingServices* – used for detecting possible feature interference (section 2.4.3.2)
- *supportingServices* – services required for the service (during the load procedure of the policy, the supporting services must be available)
- *billingFramework* – information regarding the billing policy of the service.

Additional service policy metrics that were identified include *ResponseTimes*, *ServiceGapping* and *numberOfConnections*.

The state model then moves to the *loadService* state, where the service implementation is loaded. This transition causes further state changes in the service segment association for the specific service. For example, the service segment association may move from the *serviceRequest* to the *stableServiceExecution* service segment.

The following state (*sendMessages*) indicates that the service was loaded and is now in a stable condition (represented by the *stableServiceExecution* service segment). When in this state, the service is able to execute within the service execution environment of the API Server.

Prior to the termination of the service, indicated by the *serviceTermination* transition, the model moves to the billing state. Here, the customer data record is charged by the service control and management platform according to the billing policy of the service.

8.3.6 Client State Model

The client state model, depicted in figure 8.8, is set up when a third-party service provider initiates a service request to the API Server.

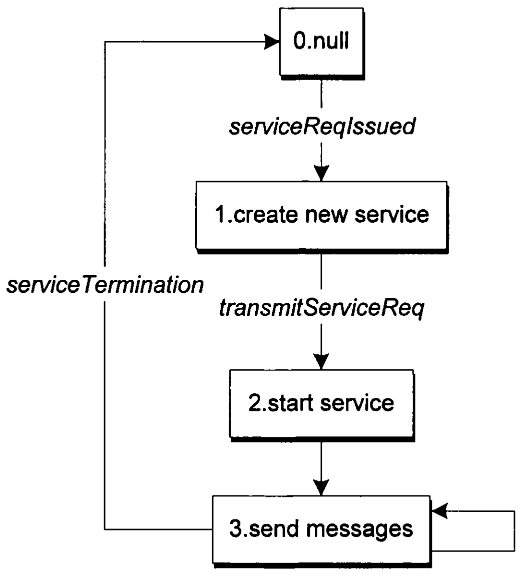


Figure 8.8: *The Client state model (C_SSM)*

The state model is used to monitor the progress of the outgoing service requests. For such requests, it is of interest to maintain the status of the requests, but at the same time not to overload the API Server with what may be pointless states. Therefore, the states that are maintained for the client-side state model are kept to a minimum.

Once a service request is received, the state model enters the *createNewService* state. Here, the internal state variables (section 7.2.4) are initialised. The service segment associations and the service segments are also instantiated. When the management platform receives the necessary internally-generated messages from the client-side control module, it issues a *transmitServiceReq* message that triggers the sending of the request and thus the starting of the service.

8.4 USING SDL TO SIMULATE THE API SERVER

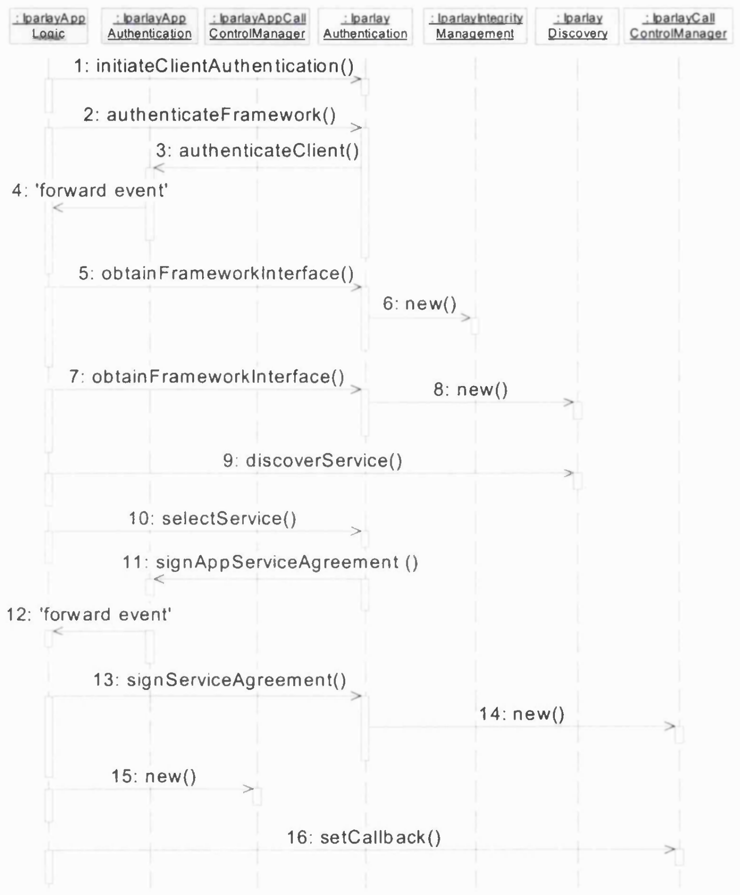
This section presents a simulation involving the Parlay WakeUp application example. The simulation was carried out using the Telelogic Tau tool, SDT. A complete description of the application example can be found in [ParlaySeq99]. In order to present the simulation of the Parlay WakeUp application example it is necessary to describe the WakeUp application.

8.4.1 UML Sequence Diagrams

This section presents two UML [Z.100] sequence diagrams. The first describes the sequence of events for any application to access a Parlay server. The second describes the operation of the Parlay WakeUp application example.

8.4.1.1 Accessing the Parlay Framework

The sequence diagram in figure 8.9 [ParlaySeq99] shows an application accessing the Parlay framework for the first time. In order for the application to use the Parlay services, it must first authenticate itself with the framework and then discover an appropriate service [ParlayFw99].



: IparlayApp Logic
: IparlayApp Authentication
: IparlayAppCall ControlManager
: Iparlay Authentication
: IparlayIntegrity Management
: Iparlay Discovery
: IparlayCall ControlManager
1: initiateClientAuthentication()
2: authenticateFramework()
3: authenticateClient()
4: 'forward event'
5: obtainFrameworkInterface()
6: new()
7: obtainFrameworkInterface()
8: new()
9: discoverService()
10: selectService()
11: signAppServiceAgreement ()
12: 'forward event'
13: signServiceAgreement()
14: new()
15: new()
16: setCallback()

Figure 8.9: *UML sequence diagram showing an application accessing a Parlay Server adopted from [ParlaySeq99]*

The messages, identified by their sequence numbers in figure 8.9, are described below:

1	Determines the authentication mechanism to be used between objects implementing the *IparlayAppLogic* interface and the *IparlayAuthentication* interface
2	Used by the client to authenticate the framework
3	Used by the framework to authenticate the client
4	Forwards message 3 to the *IparlayAppLogic*
5	Receives a reference to the object implementing the *IparlayIntegrityManagement* interface
6	Creates an object that implements the *IparlayIntegrityManagement* interface
7	Receives a reference to the object implementing the *IparlayDiscovery* interface
8	Creates an object implementing the *IparlayDiscovery* interface
9	Requests the object implementing the *IparlayDiscovery* interface to pass an appropriate service identifier back to the application (version 1.0.1 of the Parlay API specification supports only call control, messaging and user interaction)

10	Informs the object implementing the *IparlayAuthentication* interface of the service it requires The application is returned information relating to the service level agreement, which should be signed by both parties.
11	Used by the framework to ask the application to sign the service level agreement
12	Forwards message 11 to the *IparlayAppLogic*
13	Used by the application to ask the framework to sign the service level agreement
14	Creates an object implementing the *IparlayCallControlManager* interface. This only happens once the service level agreement is signed and before returning the signature and the reference to the service manager back to the application via the return parameter of message 13.
15	Creates the application service manager.
16	Passes the application service manager's callback reference to the object implementing the *IparlayCallControlManager*

Table 8.1: *Messages used by an application accessing a Parlay Server*

8.4.1.2 The Alarm Call (WakeUp) Application Example

Figure 8.10 depicts the UML sequence diagram for an alarm service that shows a reminder message. The message is delivered to a customer as a result of a trigger from an application. Typically, the application would be set to trigger at a certain time, however, it can also trigger on events. This sequence assumes that an *IparlayUICall* service interface [ParlaySeq99] has already been obtained.

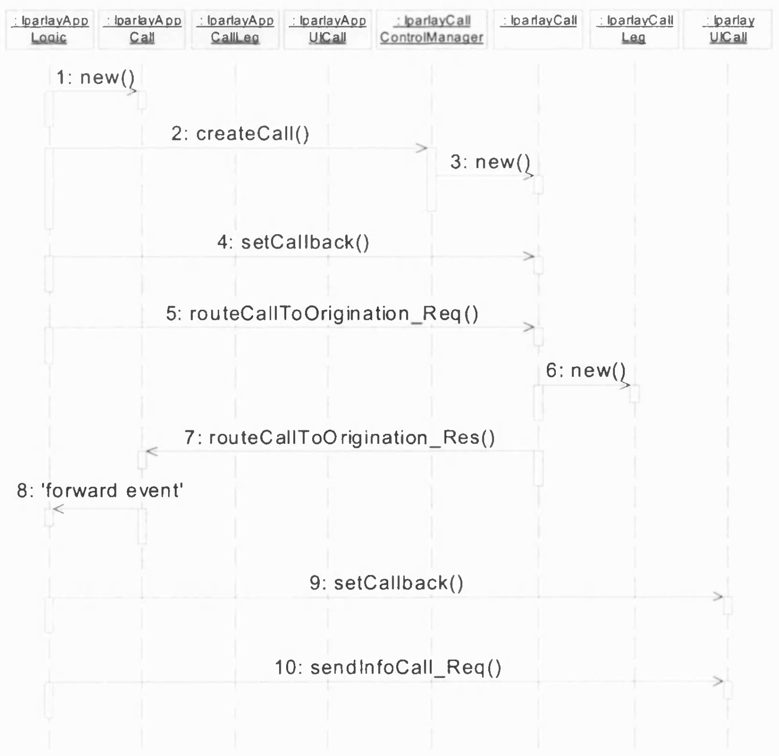


Figure 8.10: *UML sequence diagram for an Alarm Call Service adopted from [ParlaySeq99]*

The messages, identified by their sequence numbers in figure 8.10, are described below:

1	Creates an object implementing the *IparlayAppCall* interface
2	Requests the object implementing the *IparlayCallControlManager* interface to create an object implementing the *IparlayCall* interface
3	Creates an object implementing the *IparlayCall* interface if the criteria (e.g. load control values not exceeded) are met
4	Passes the reference of the object implementing the *IparlayAppCall* interface to the object implementing the *IparlayCall* interface
5	Instructs the object implementing the *IparlayCall* interface to route the call to the customer destined to receive the 'reminder message'
6	Used by the object implementing the *IparlayCall* interface, if the call is answered, to create an object implementing the *IparlayCallLeg* interface, which models the call leg of the customer to receive the alarm
7	Passes the result of the call being answered to its callback object
8	Forwards message 7 to the *IparlayAppLogic*
9	Forwards the address of the callback object
10	Instructs the object implementing the *IparlayUICall* interface to send the alarm to the customer's call leg

***Table 8.2:** Messages used by an alarm call service*

8.4.2 Simulation Overview

The UML specification corresponds to a one-to-one mapping of the objects that are described in the Parlay framework access interfaces [ParlayFw99].

The following objects were included in the SDL design for the purposes of the simulation: *IparlayAppLogic*, *IparlayAppAuthentication*, *IparlayAppCall*, *IparlayCall, IparlayDiscovery, IparlayAuthentication* and *IparlayCallControlManager.* The objects *IparlayCallLeg*, and *IparlayUICall and IparlayIntegrityManagement* were not included because these objects do not play a major part in the exchange of messages between the Parlay server and the application.

The following sections give a description of the system specification.* The simulation of the system in SDL was accomplished by specifying first the client side of the Parlay API and then the server side.

8.4.2.1 Client-Side Specification of Parlay API

The first phase in developing the specification of the WakeUp Application was to design a system containing only the signals sent and received by the client side. In this case, client side means the service application which offers the service and sits on one side of the interface.

* Note that "system specification" in this chapter refers to the specification of the system in SDL terms.

Figure 8.11 shows the system *Wakeupapplication*, containing the following blocks and signals:

- The *ParlayAccess* block represents the sequence diagram of figure 8.9, grouping together the objects and processes responsible for the messages that establish access to the framework interface.
- The *AlarmCall* block represents the sequence diagram of figure 8.10, grouping together the objects and processes responsible for the messages that offer the alarm call service. In the case of *AlarmCall* this encapsulates only process *IparlayAppCall.*
- The *TimeOut* block represents timeouts generated by all processes. This block generates the following signals: *AccTimeOut1*, *AccTimeOut2*, *AlarmTimeOut*, *AlarmTimeOut1* and *AlarmTimeOut2*.
- Channels C1, C2, C3 and C6 are the paths which carry a number of signals between the environment and the system. During the simulation, incoming signals are under the control of the specification developer and timeouts are needed to simulate the timeout of specific responses. In this phase of the specification, inputs that are expected from the developer and cause timeouts include: *userReqForServ*, *signAppServiceAggremen, authenticateClient* and *routeCallToOrigination_Res* (via channels C1, C3 and C6).
- Channels C4, C5, C7 and C8 carry any signals which are internal to the system and therefore act as the interfaces between the system's blocks.
- A SIGNAL list is needed in the specification in which most of the signals are declared. Signals that have only local significance are declared internally within a block, and not in this list.

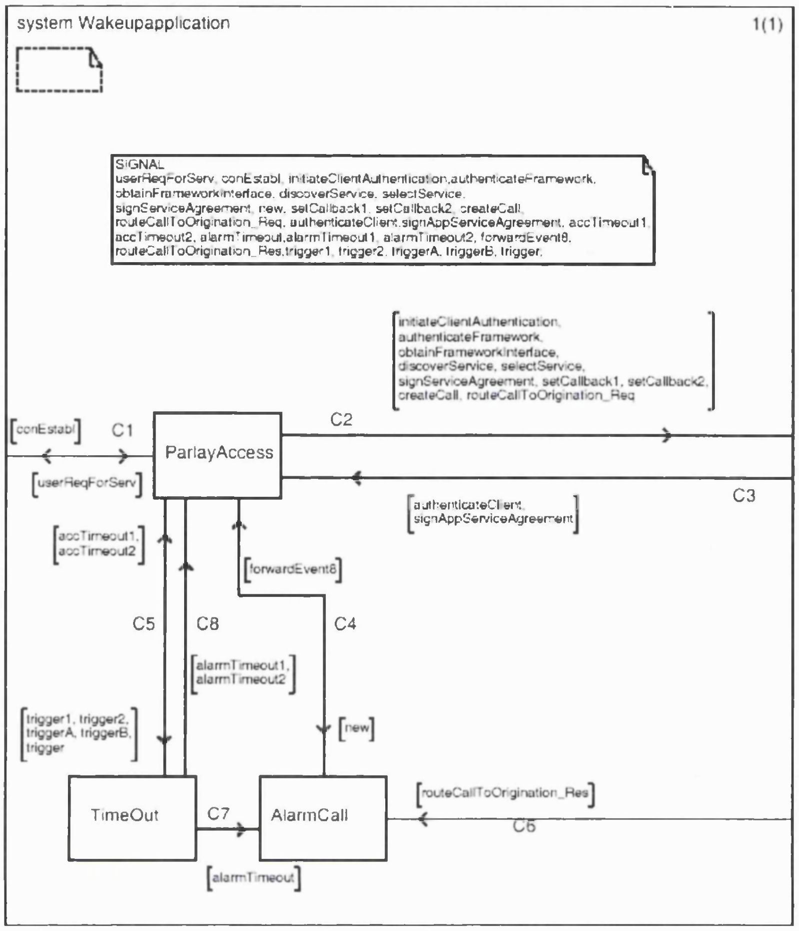

system Wakeupapplication
1(1)
SIGNAL
userReqForServ, conEstabl, initiateClientAuthentication,authenticateFramework,
obtainFrameworkInterface, discoverService, selectService,
signServiceAgreement, new, setCallback1, setCallback2, createCall,
routeCallToOrigination_Req, authenticateClient,signAppServiceAgreement, accTimeout1,
accTimeout2, alarmTimeout,alarmTimeout1, alarmTimeout2, forwardEvent8,
routeCallToOrigination_Res,trigger1, trigger2, triggerA, triggerB, trigger,
initiateClientAuthentication, authenticateFramework, obtainFrameworkInterface, discoverService, selectService, signServiceAgreement, setCallback1, setCallback2, createCall, routeCallToOrigination_Req
C2
conEstabl
C1
ParlayAccess
userReqForServ
authenticateClient, signAppServiceAgreement
C3
accTimeout1, accTimeout2
forwardEvent8
C5
C8
C4
alarmTimeout1, alarmTimeout2
trigger1, trigger2, triggerA, triggerB, trigger
new
routeCallToOrigination_Res
TimeOut
C7
AlarmCall
C6
alarmTimeout

Figure 8.11: *Specification of the Wakeupapplication system*

The simulation begins as follows: the *ParlayAccess* block encapsulates the processes *IparlayAppLogic* and *IparlayAppAuthentication*. *IparlayAppLogic* receives the *userReqForServ* signal via C1 and initiates the system behaviour by outputting the first of the signals needed to access the Parlay framework interface to the environment. The *userReqForServ* signal must be sent initially by the user, in order to initiate the state diagram of process *IparlayAppLogic.* The process thus begins exhibiting the system behaviour.

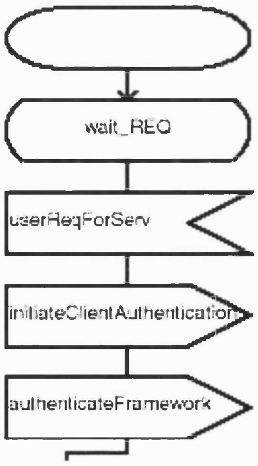
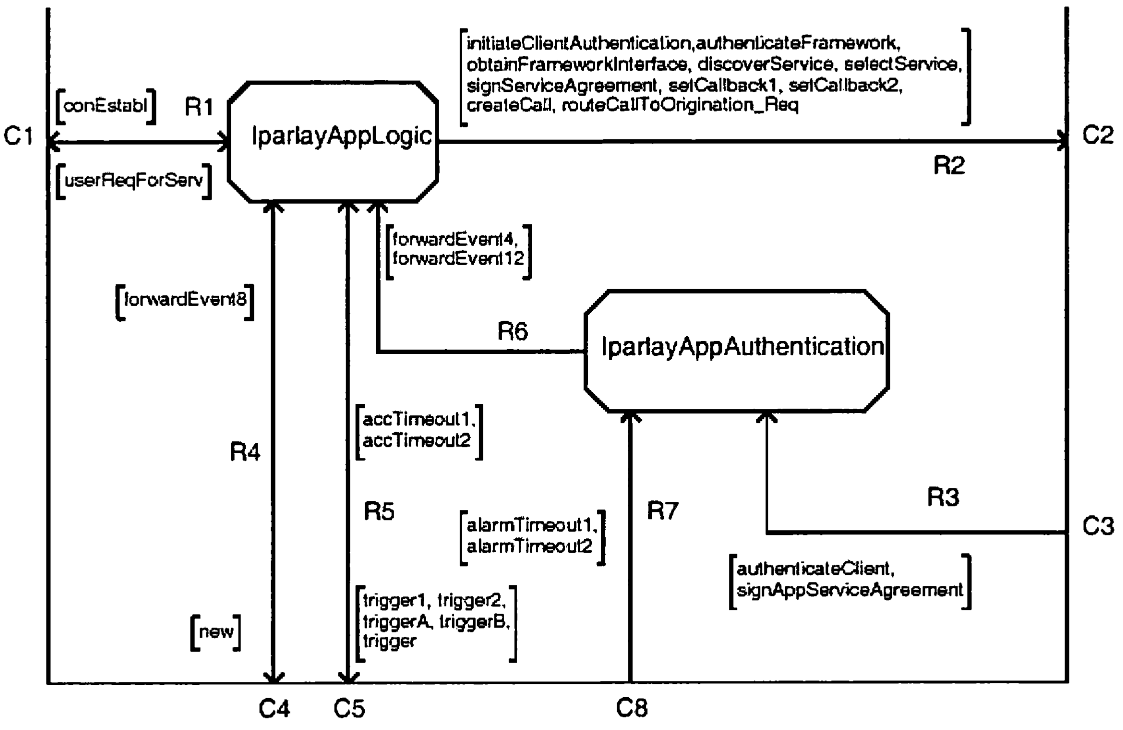


Figure 8.12: *Part of the IparlayAppLogic state diagram*

Figure 8.12 shows part of the state diagram for the *IparlayAppLogic* process. The process diagram begins with the *start* symbol and then enters the *wait_REQ* state until receiving the first signal from the user. When *userReqForServ* is received and consumed by the process, the first outputs are sent to the environment: *initiateClientAuthentication* and *authenticateFramework.* In addition to the above signals, the *IparlayAppLogic* process also outputs *trigger1* and *trigger2* to the *TimeOut* block via channel *C5*. Process *IparlayAppLogic* sends these signals to trigger processes *AccTimeOut1* and *AlarmTimeOut1* into generating the timeouts that indicate that the corresponding responses to signals *initiateClientAuthentication* and *authenticateFramework* have not arrived within the expected time limits. Two timeouts were needed because process *IparlayAppLogic* was not expecting the response from the environment. Process *IparlayAppAuthentication* gets the response from the environment (signal *authenticateClient*) and then reports back to process *IparlayAppLogic* (signal *forwardEvent4*). Under normal operation, the two responses arrive on time and the timeout signals are not taken into account by the processes. Instead they are discarded and a transition to the next state takes place.

Figure 8.13 shows the *IparlayAppAuthentication* process. Signal *authenticateClient* should arrive by signal route *R3*. If timeout *alarmTimeout1* arrives first, from signal route *R7*, the process terminates, as shown in figure 8.14.



C1
[conEstabl]
R1
[userReqForServ]
IparlayAppLogic
[initiateClientAuthentication,authenticateFramework, obtainFrameworkInterface, discoverService, selectService, signServiceAgreement, setCallback1, setCallback2, createCall, routeCallToOrigination_Req]
C2
R2
[forwardEvent4, forwardEvent12]
[forwardEvent8]
R6
IparlayAppAuthentication
[accTimeout1, accTimeout2]
R4
R5
[alarmTimeout1, alarmTimeout2]
R7
R3
C3
[authenticateClient, signAppServiceAgreement]
[trigger1, trigger2, triggerA, triggerB, trigger]
[new]
C4
C5
C8

Figure 8.13: *Part of the ParlayAccess block showing process IparlayAppAuthentication receiving signal authenticateClient from the environment via channel C3*

The major sub-blocks and signals of block *ParlayAccess* (figure 8.13) are as follows:

- The *IparlayAppLogic* process can be regarded as the central process which initiates and offers the service. Of course, the process cannot function on its own; instead, the collaboration of all the processes defines the complete system behaviour.
- The *IparlayAppCall* process establishes the service in this specification.
- The *IparlayAppLogic* process sends the final message (signal *conEstabl*) indicating to the user that a service has been granted.

A number of timeout signals are generated throughout the simulation. The names of the processes generating timeouts were chosen according to which block was expecting a response from the environment, e.g. the *ParlayAccess* or *AlarmCall* block.

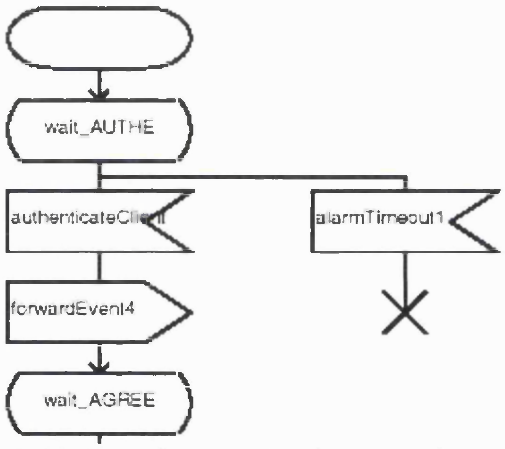


Figure 8.14: *Part of process IparlayAppAuthentication.*

If process *IparlayAppAuthentication* receives input *alarmTimeout1* before signal *authenticateClient,* the process terminates. Figure 8.15 shows process *AccTimeOut1*generating output *accTimeout1*. Timer *T* is set to one unit of time from the moment the process receives signal *trigger2*.

The rest of the timers are generated in the same fashion in their corresponding processes. All timers have a duration of one unit since this is only a theoretical model which does not take into account any real parameters of a network (for example, delay) and therefore it is not important to have some specific duration of time in which to generate a timeout. Of course, in practice the timers would take network characteristics into account.

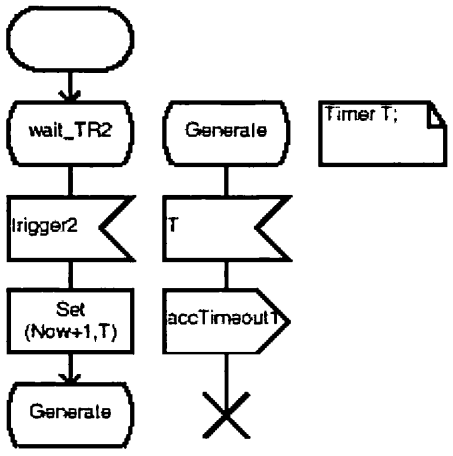


Figure 8.15: *Process AccTimeOut1 generating output signal accTimeout1*

8.4.2.2 Server-Side Specification of Parlay API

These processes define the behaviour of the server side of the API. Block *ParlayServer* encapsulates processes *IparlayAuthentication, IparlayDiscovery, IparlayCallControlManager,* and *IparlayCall.*

These processes are responsible for sending the various responses from the Parlay Server back to the client application Wake Up (alarm call) and, of course, forwarding the service to the customer. The forwarding of the service, by the server, is not shown in this specification since this was not implemented in the application.

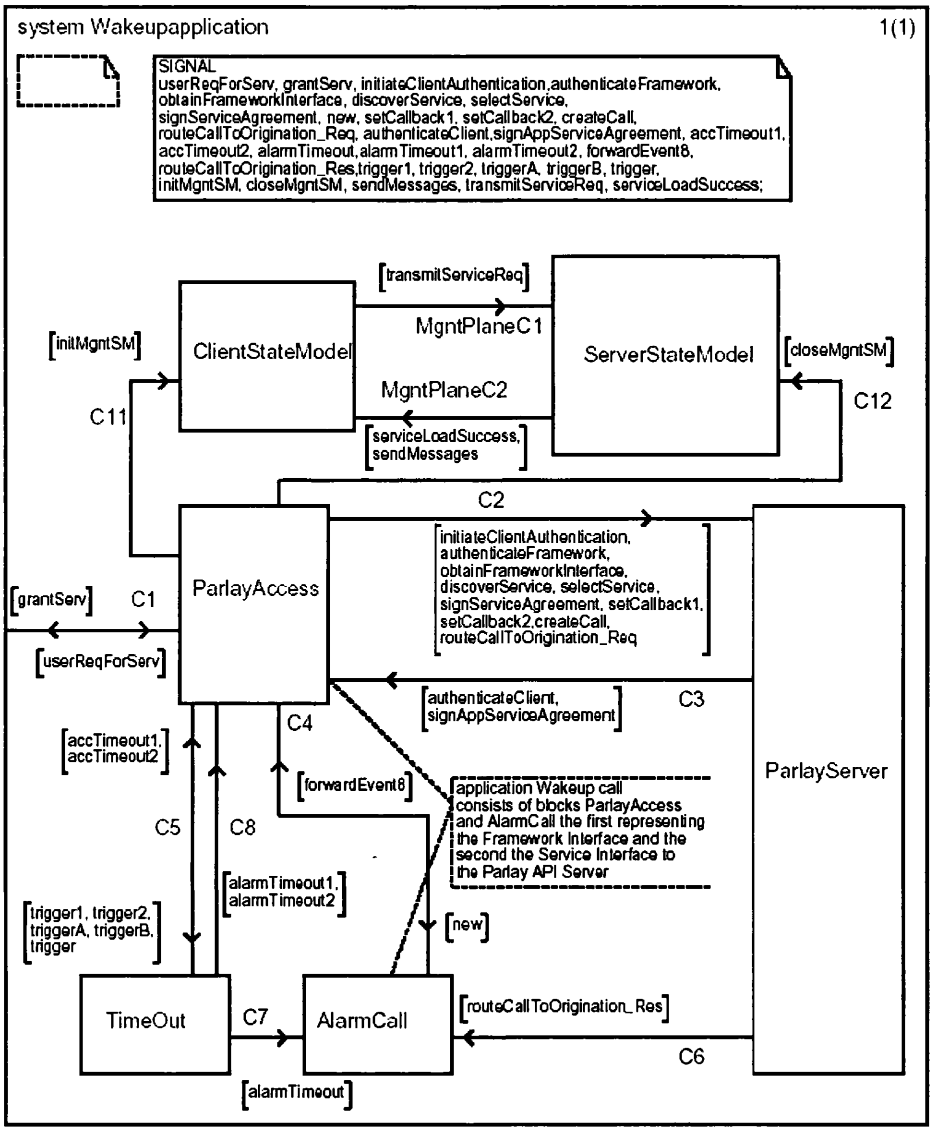

system Wakeupapplication
1(1)
SIGNAL
userReqForServ, grantServ, initiateClientAuthentication,authenticateFramework,
obtainFrameworkInterface, discoverService, selectService,
signServiceAgreement, new, setCallback1, setCallback2, createCall,
routeCallToOrigination_Req, authenticateClient,signAppServiceAgreement, accTimeout1,
accTimeout2, alarmTimeout,alarmTimeout1, alarmTimeout2, forwardEvent8,
routeCallToOrigination_Res,trigger1, trigger2, triggerA, triggerB, trigger,
initMgntSM, closeMgntSM, sendMessages, transmitServiceReq, serviceLoadSuccess;
transmitServiceReq
MgntPlaneC1
ClientStateModel
ServerStateModel
initMgntSM
closeMgntSM
MgntPlaneC2
C11
C12
serviceLoadSuccess, sendMessages
C2
ParlayAccess
initiateClientAuthentication, authenticateFramework, obtainFrameworkInterface, discoverService, selectService, signServiceAgreement, setCallback1, setCallback2,createCall, routeCallToOrigination_Req
grantServ
C1
userReqForServ
authenticateClient, signAppServiceAgreement
C3
C4
accTimeout1, accTimeout2
forwardEvent8
application Wakeup call consists of blocks ParlayAccess and AlarmCall the first representing the Framework Interface and the second the Service Interface to the Parlay API Server
ParlayServer
C5
C8
alarmTimeout1, alarmTimeout2
trigger1, trigger2, triggerA, triggerB, trigger
new
TimeOut
C7
AlarmCall
routeCallToOrigination_Res
C6
alarmTimeout

Figure 8.16: *Sequence diagram of alarm call service, adopted from [ParlaySeq99]*

Figure 8.16 shows the *Wakeupapplication* system, modified to incorporate the *ParlayServer* block. The *ParlayServer* block was introduced in the design for the specification to be able to show the complete behaviour of the Parlay API in an automated way. That is, the system simulation should run without interference from the user, except of course from the initial input *userReqForServ*. This exchange of messages between the client and server blocks needs to be automatic as the next phase of the specification specifies a management plane and illustrates how this plane could be used to monitor the application messages without interfering. The management plane's function is independent from the Parlay API and runs in parallel with the application.

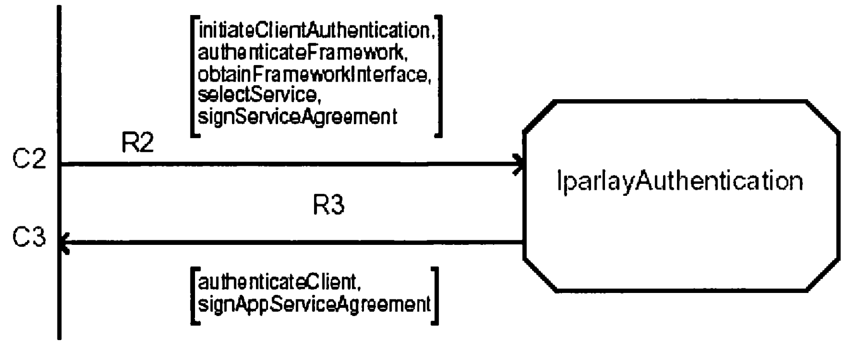


Figure 8.17: *Part of block ParlayServer showing the implementation of process IparlayAuthentication interfacing to the system via R2 and R3*

Figure 8.17 shows process *IparlayAuthentication* receiving some of the messages destined for block *ParlayServer.* It replies with the necessary outputs via signal route *R3* interfaced to the system via channel *C3.*

8.4.3 Simulation

The Telelogic Tau [Tau3.5] tool with which the system was specified is supported by the Telelogic MSC simulation tool [Ekka95]. The MSC tool provides a way of executing the system under the user's control. Any signals arriving from the environment, therefore, were triggered by the developer, who can observe a visual display of the generation of signals, exchanged between instances in the form of an MSC. Timers within the system are displayed, as well as the states into which the system has entered at a given point during the execution cycle.

During the various stages of development of the system *Wakeupapplication,* a number of simulations were tried out as a verification of system behaviour in order to test if the design was created according to the UML sequence diagrams of the Parlay specifications. The simulations showed that the proposed state models for the API Server architecture can function within the Parlay framework.

8.5 Summary and Conclusions

This chapter presented the issues relating to open programmable networks and the way such a change will impact on service provisioning. Industry initiatives, such as that of Parlay, have generated a large interest in opening up core network resources through APIs.

However, as identified in this chapter, it is crucial that a framework exists that allows the communication of third-party service providers. The framework manifested in the form of the API Server enables third-party service providers to communicate using a common framework. This allows the management interfaces to access remote features of other API servers and, in doing so, they may even request that a service be simulated before it is released in the real network environment, thus reducing any effects of failure.

The API Server that was presented in this chapter is based on the proven principles of the IN control architecture, but also incorporates the flexibility of the IP domain. The service execution platform is supported by a number of robust state models that enable the local and remote management platforms to have a complete view of the status of the executing services.

Furthermore, the work presented here was simulated using a real Parlay example, the WakeUp service. The sequence diagrams of the WakeUp service were used and implemented in UML. The UML simulation also incorporated the proposed state models of the API Server.

The following chapter contains the conclusions and suggestions for further work that is needed as a result of the work undertaken by the author.

CHAPTER 9

CONCLUSIONS AND FURTHER WORK

This chapter provides a summary of the main conclusions that can be extracted as a result of the work presented in this thesis. It also puts forward suggestions for further work in the field.

9.1 DISCUSSION

The thesis provided an examination of network intelligence architectures within the converging telecommunications environment. Network intelligence is provided through the control plane, which is responsible for the establishment, operation and termination of calls and connections. The control plane provides a robust foundation for the communication of the functional components (e.g. switches) and a resilient environment for the execution of advanced services.

One of the aims of the thesis was to understand the traditional approach to network intelligence in the PSTN. This was achieved through the discussion of the Intelligent Network architecture, which is a dedicated control plane architecture. The IN defines the BCP that is encapsulated in the BCSM. The BCSM identifies the logical points in basic call-processing where the IN service logic located in the switch is permitted to interact with basic call capabilities provided by the switch. Service initiations can only take place within the boundaries set by the BCSM. Furthermore, the IN capability sets were presented and it was noted that an important and significant addition to the capabilities of the IN CS-2 is that of CPH (see section 2.4.6).

In the PSTN, control of the bearer connections is achieved through:

- the extensive use of state machines
- the ubiquitous description of the BCM
- the complete encapsulation of the call process in the BCSM and
- the robust nature of the signalling network and its protocols.

The state models in the PSTN control plane offer sufficient capabilities to capture the unpredictable behaviour of end users.

To understand the evolution of network intelligent architectures, the work examined the role of traditional network intelligence architectures in the converging environment. Within the converging telecommunications environment it is essential to enable the IN architecture to inter-work with IP-based architectures. The ways in which the IN architecture can be used with new network intelligence architectures were presented in the form of a taxonomy reference model. The model provided a classification of network intelligence architectures and protocols in relation to the Intelligent Network. Within the taxonomy, a number of areas of inter-working were identified. One such area examined the utilisation of existing IN capabilities to support IP-based services.

The IEPS application was presented within the context of utilising existing IN capabilities to support IP-based services. This approach utilises the existing IN CS-1 architecture to allow charging of electronic transactions on telephone bills. The IEPS does not require users to be holders of credit cards and therefore avoids the controversy surrounding the privacy and security issues associated with credit cards. The IEPS was designed, implemented, simulated and a patent application has been filed in the UK [IEPSPat].

Following this, the thesis examined the role of state models in IP-based systems. The work presented a literature review of the ways that state-driven behaviour can be implemented; using optimistic or pessimistic policies. More importantly the work examined IP-based protocols and provided a classification of these with regard to their utilisation of state information. To the author's knowledge such a classification has not been provided in the existing literature. The work provided a classification of IP-based systems using metrics such as whether:

- state is critical
- state plays a supplementary role or the system is stateless
- state between sessions is a requirement
- persistent session management is mandatory.

However, following the work in this area, it is concluded that there is no clear line between a state-dependent and stateless system. A system may exhibit a state-driven behaviour but may not be state-based. State-driven behaviours are the actions performed in a system as a result of the occurrence of certain events. Such a system could be characterised as state-based even if it does not track the point (i.e. state) in the behaviour model of the system. Furthermore, in a distributed system, a heavily state-based design would track the state of all the distributed sub-systems with which it interacts. Therefore, the reliance on state models for any system is not defined in a formally quantified manner. It is further concluded that the classification of a protocol as state-based or stateless depends on the viewpoint, level of abstraction and specific instance in time one takes on the protocol (sections 6.3.7 and 6.5).

To examine the interactions in distributed, IP-based, state-dependent systems the work presented the design and implementation of a Network Management System (NMS). The NMS provided the foundations for examining the issues involved in implementing a distributed system that was state-driven.

A separate concept that was presented and is supported by a number of publications by the author is the role of the TINA service architecture in relation to the converging telecommunications environment (sections 2.5 and 3.4). It was put forward that the TINA service architecture provides a useful reference model for viewing new technologies, irrespective of whether a complete TINA architecture is deployed.

Through the discussions of the IN CS-2, it was put forward that the SCF in the IN CS-2 (over IN CS-1) is of higher importance for establishing connections. This translates to allowing greater control of the TINA Service Session through the SCP that is located in the communications session. Furthermore, in the converging environment, the work presented the need for two APIs: one between the service and communications sessions and one between a gateway (in the service session) and service providers (in the service session).

Finally, the more general case of inter-working is one in which quality of service guarantees can be made between homogeneous network types as well as heterogeneous network types. For example, one may wish to establish a call between three parties, each originating from a different network type. In the author's view, the IN architecture provides an existing and evolving set of standards that will facilitate the migration towards these types of scenarios. Hence, the IN should be viewed as an important architecture for the TINA communication session.

Within the open service provisioning framework, the thesis identified a framework for the communication of third-party service providers. Such a framework is necessary as, in the author's view, it will provide a solution to problems (refer to section 8.2 for discussion) that may arise as a result of open third-party service provisioning. The work presented an API server architecture that complements the Parlay API. The API server architecture draws from the advantages and proven track record of state machines, in the PSTN, but also incorporates features and the flexibility of the IP domain.

9.2 Summary of Contributions

In **chapter 1** the problem field and the approach were put forward. The chapter identified the aims of the work and detailed the approach that was followed in achieving the aims of the thesis.

Chapter 2 provided a historical background to telecommunications and examined the traditional approach to network intelligence in the PSTN. Network intelligence in the PSTN is provided in the form of the IN architecture, which was examined as a case scenario. The chapter provided an in-depth analysis of the IN architecture and showed that the control plane of the PSTN is heavily based on ubiquitous state models that inter-work closely. The main contribution from the chapter, presented in section 2.5, proposed a different role for the TINA architecture, one in which the TINA architecture can be used as a reference model for viewing new technologies.

The main contribution in **chapter 3** is the work from the taxonomy reference model (section 3.3). The model provided a unique view of network intelligence architectures in the current state of convergence and their relation to the IN. This unique view provides a clearer understanding of the issues involved in providing network intelligence in a converging environment. Developing the taxonomy model was a complex task because of the large number of propositions that arise in such an environment; however, the taxonomy reference model manages to overcome the complexities and provides a useful contribution.

Chapter 4 described the IEPS application. The importance of the system is that it enables the utilisation of existing IN architecture to support IP-based services. The system is based on existing IN CS-1 information flows and, therefore, does not necessitate any changes in the telecommunications environment. Furthermore, the IEPS is unique in that it provides support for micro-transactions. This is advantageous because it overcomes the limitations of existing payment systems (refer to section 4.2). The implementation and simulation of the system were presented in **chapter 5**.

The main research contribution from **chapter 6** is the examination of IP-based protocols and architectures with regard to their utilisation of state. The work provided an overview of numerous protocols and architectural frameworks from the viewpoint of state utilisation. The importance of this is the conclusion that in some systems, the utilisation of state is imposed by the concurrency complexities of the system.

To understand the complexities in implementing state in a distributed system, **chapter 7** examined presented the design and implementation of the network management system. The NMS is a useful contribution as it can be used as a platform for further research in determining the overheads involved in maintaining state-based information.

The contributions in **chapter 8** are many-fold. These include the development of an architectural framework for the communication of third-party service providers within an open network environment. Furthermore, the chapter identified an API Server architecture that draws from the robust foundations of the IN, but also incorporates the flexibility of the IP-domain. A further contribution of this chapter is the definition of state machines that comply with the Parlay API. The state machines provide the foundations for easily providing a framework for the billing of services within the open service provisioning environment.

9.3 FURTHER WORK

As a result of the work undertaken in this thesis, the following areas have been identified as requiring further research activities.

Firstly, there is a need to examine distributed state-based systems. The work presented in this thesis provided the foundations for the classification of IP-based systems using metrics that were identified. These metrics are not formally defined or quantified. There needs to be a formal definition of metrics that enable the classification of systems on a scale from state-full to stateless. Such work could result in the definition of a state-based coefficient, which could be used in software engineering practices.

Secondly, the existing NMS system could be modified so that the modules are aware of the state the other modules are in. This would enable the quantification of the overheads involved in maintaining state information and such quantification is desirable. In the author's view, when reading journals and publications relating to state models and concurrency, one gets the feeling that the overheads and complexity imposed in controlling concurrency and multi-threading are generally accepted whereas state models for management or billing are generally viewed as creating unnecessary overheads.

Thirdly, work could be undertaken to provide a quantified and fully-defined metric of network intelligence. Such work is desirable as there is currently no means to measure the intelligence of a network. However, at the same time it would be difficult to perform such measurements, especially in networks such as the Internet, where the topologies are not visible.

Finally, further work is necessary to fully evaluate the work presented in chapter 8. The API server architecture presented has provided the foundations for a framework that enables a unified communication between third-party service providers. Further work in this area should initially focus on identifying the view of open standards groups of this architecture, although informal discussions by the author have shown that such a framework is desirable. Then there needs to be extended simulation of the existing state models with newer versions of the Parlay API and cooperation with existing research activities that focus on the specification of policies within the open network framework. Finally, the incorporation of mobile agent technologies within the architectural framework should be evaluated.

CHAPTER 10

REFERENCES

[ABA01] American Bar Association, *"Digital Signature Guidelines: Legal Infrastructure for Certification Authorities and Secure Electronic Commerce"*, American Bar Association, August 1996, ISBN 1-57073-250-7, Available online at: http://www.abanet.org/ftp/pub/scitech/ds-ms.zip

[Adis98] H. Adiseshu, G. Parulkar, R. Yavatkar, *"A state management protocol for IntServ, DiffServ and label switching"*, Applied Research Laboratories, Washington Univ., St. Louis, USA, Network Protocols, 1998. Sixth International Conference on Network Protocols, 13-16 October 1998.

[Advent] AdventNet, "*SNMP API Release 3.2*", AdventNet Inc. Available online at: http://www.adventnet.com

[AGCS] AGCS, "History of AG Communication Systems", AGCS Web Page. Available online at: http://www.agcs.com

[Alex93] V. Alexiev, *"Mutable Object State for Object-Oriented Logic Programming: A Survey, Technical Report"*, TR 93-15, Department of Computing Science, University of Alberta, 1993.

[Alex98] Alexander, *"Safety and security of programmable network infrastructures"*, Pennsylvania Univ., Philadelphia, PA, USA, IEEE Communications Magazine, pp. 84-92, October 1998.

[Alvi98] L. Alvisi and K.Marzullo, "*Message Logging: Pessimistic, Optimistic, Causal, and Optimal*", IEEE Transactions on Software Engineering, Vol. 24 No. 2, pp. 149-159, February 1998.

[Amar01] Amarach Consulting, *"Eir-Commerce 2001 Ireland's Consumer Internet Economy: Gloom or Boom?"*, June 2001. Available online at: http://www.amarach.com/study_rep_downloads/eir-commerce2001.pdf

[Ambr89] W. D. Ambrosch, A. Maher, B. Sasscer, *"The Intelligent Network: A joint study by Bell Atlantic, IBM and Siemens"*, Springer-Verlag, ISBN 3-540-50897-X, 1986.

[Aror98] A. Arora and S. Kulkarni, "*Component Based Design of Multi-tolerant Systems*", IEEE Transactions on Software Engineering, Vol. 24, No. 1, pp. 63-78. January 1998.

[Asat01] K. Asatani, F. Bigi, and Pierre-Andre Probst, "T*elecommunications Standardization for the New Millennium: ITU–T's Strategies"*, IEEE Communications Magazine, pp. 124-130, April 2001.

[Aust00] T. Austin, *"PKI: A Wiley Tech Brief"*, John Wiley & Sons, Inc., ISBN: 471353809, December 2000.

[Baet99] C.M. Baeten (Editor), S. Mauw (Editor), *"Concurrency Theory"*, Springer-Verlag Berlin and Heidelberg GmbH & Co. KG; ISBN: 3540664254, 1999.

[Basle99] Basle Committee on Banking Supervision, *"Risk Management for Electronic Banking and Electronic Money Activities"*, March 1999.

[Baum92] S. Baum, *"Linking Security and the Law of Computer-based Commerce"*, National Institute of Standards Technology, Workshop on Security Procedures for the Inter-exchange of Electronic Documents, Maryland, 1992.

[Baum94] S. Baum, *"Federal Certification Authority; Liability and Policy"*, US Department of Commerce, National Institute of Standards and Technology, Maryland, 1994.

[BBC00] BBC News, *"BT Network Fault Fixed"*, BBC, 26th February 2000, Available online at: http://news.bbc.co.uk

[Bedd00] Beddus, G. Bruce, S. Davis, *"Opening Up Networks with JAIN Parlay"*, IEEE Communications Magazine, pp. 136-143, April 2000.

[Bhat00] R. Bhat and R. Gupta, *"JAIN Protocol APIs"*, Trillium Digital Systems, Inc., IEEE Communications Magazine, pp. 100-107, January 2000.

[BIS96] Bank for International Settlements, *"The Security of Electronic Money"*, Report by the Committee on Payment and Settlement Systems and the Group of Computer Experts of the Central Banks of the Group of Ten countries, Basle, August 1996, ISBN 92-9131-119-7.

[Bisw98] J. Biswas, *"The IEEE P1520 standards initiative for programmable network interfaces"*, Kent Ridge Digital Labs., Singapore, IEEE Communications Magazine, pp. 64-70, October 1998.

[Boge01] M. Boger, *"Java in Distributed Systems: Concurrency, Distribution and Persistence"*, John Wiley & Sons Inc., ISBN 04-7149-838-6, 2001.

[Booc99] G. Booch, B. Cummings, *"Object-Oriented Analysis and Design with Applications"*, Addison Wesley; ISBN: 020189551X, November 1999.

[Bran93] S. Brands, *"Untraceable off-line cash in wallet with Observers"*, Advances in Cryptology, Crypto 1993, pp. 302-0318, 1993.

[Brot93] R. Brothers, *"Feature interaction detection"*, Bellcore, USA, IEEE International Conference on Communications, 1993, ICC'93 Geneva 23-26 May 1993.

[Brus98] A. Brusilovsky, *"A Proposal for Internet Call Waiting Service using SIP: An Implementation Report"*, IETF PINT Working Group, Internet Draft, November 1998.

[BTPr01] British Telecom, *"BT to Extend Reach of UK Broadband ADSL Service"*, BT Press Release NR0140, 25th June 2001.

[Calh01] P. R. Calhoun, *"Diameter Framework Document"*, IETF AAA Working Group, Internet Draft, March 2001.

[Cape96] C. Capellmann, K. Kimbler, *"Towards efficient feature interaction handling"*, Deutsche Telekom, Darmstadt, Germany, IEEE Intelligent Network Workshop, 1996. IN '96. 21-24 April 1996.

[Carg92] T. Cargill, *"C++ Programming Style"*, Addison-Wesley, MONTH 1992.

[Cham93] D. de Champeaux, D. Lean, P. Faure, *"Object-Oriented System Development"*, Addison-Wesley, ISBN 020156355X, 1993.

[Chau82] D. Chaum, *"Blind Signatures for untraceable payments"*, Lecture Notes in Computer Science, Advances in Cryptology, Crypto 82, Springer-Verlag, pp. 199-203, 1982.

[Chau83] D. Chaum, *"Security without identification: Transaction Systems to make Big Brother Obsolete"*, Communications of the ACM, Vol. 28, pp. 1030-1044, 1985.

[Chau92] D. Chaum, *"Achieving electronic privacy"*, Scientific American, August 1992.

[Chen00] M. Chen, *"Evolution to the Programmable Internet"*, IEEE Communications Magazine, pp. 124-128, March 2000.

[Cian99] M. C. Ciancetta, G.Colombo, R.Lavagnolo, D.Grillo, *"Convergence Trends for Fixed and Mobile Services"*, IEEE Personal Communications Interactive, pp. 14-21, April 1999.

[Coli91] A. Colin, *"Object-Oriented Reuse, Concurrency and Distribution"*, Addison-Wesley, 1991.

[Cons79] L. Constantine, E. Yourdon, *"Structured Design : Fundamentals of a Discipline of Computer Program and Systems Design"*, Prentice-Hall, 1979.

[CORBA95] Object Management Group, *"The Common Object Request Broker: Architecture and Specification (CORBA), Version 2.0"*, Object Management Group, 1995.

[Cout97] L. Couto, R. Guimarães, *"Information Technology and Its Social-Economic Impact in a Modern Society"*, Washington DC, 1997. Available online at: http://www.gwu.edu/~ibi/minerva/Spring1997/Luciano.Guimaraes/Luciano.Guimaraes.html

[Cowa01] D. Coward, *"JavaTM Servlet 2.3 and JavaServer PagesTM 1.2 Specifications"*, JSR-000053, (Proposed Final Draft 2), Sun Microsystems, April 2001.

[Curr01] I. Curry, *"An Introduction to Cryptography and Digital Signatures"*, Entrust Corporation, White Paper, March 2001.

[Databeam98] Databeam, *"A Primer on the H.323 Series Standard"*, Databeam Corporation, White Paper, 1998.

[Davi97] G. Davidson, *"Telecommunications and the Rural Economy"*, The 1997 Economic & Technology Development Journal of Canada, 1997. Available online at: http://www.edco.on.ca/journal/story5.htm

[Dawk97] S. Dawkins, *"An Introduction to JTAPI (Java Telephony API) Release 1.2 Rev. 0.7"*, Enterprise Computer Telephony Forum JTAPI White Paper, November 1997. Available online at: http://java.sun.com/products/jtapi/jtapi-1.2/JTAPIWhitePaper_0_7.html

[DCOM] M. Horstmann and M. Kirtland, *"DCOM Architecture"*, Microsoft Developer Network, July 1997.

[Deci97] M. Decina, V. Trecordi, *"Convergence of telecommunications and computing to networking models for integrated services and applications"*, CEFRIEL, Politecnico di Milano, Italy, Proceedings of the IEEE, Vol 85. No. 12, December 1997.

[DOC00] US Department of Commerce, *"Digital Economy 2000"*, Economics and Statistics Administration, Office of Policy Development, 2000.

[DOC94] US Department of Commerce, *"Policy of Certificate-Based Public Key and Digital Signatures"*, Michael S. Baum, National Institute of Standards and Technology, 1994.

[EC00] Fischer & Lorenz, *"Internet and the Future Policy Framework for Telecommunications Study on the development of new telecommunications services, in particular those exploiting Internet, and their impact (economic and otherwise) on the European Union regulatory and policy framework for telecommunications"*, Fischer & Lorenz European Telecommunication Consultants, A Report for the European Commission, January 2000. Available online at: http://europa.eu.int/ISPO/infosoc/telecompolicy/en/Fischer31a.pdf

[ECD98/10/EC] European Commission, *"Directive 98/10/EC of the European Parliament and the Council of 26 February 1998 on the application of open network provision (ONP) to voice telephony and on universal service for telecommunications in a competitive* *environment"*, The European Parliament and the Council of the European Union, February 1998.

[ECMA-143] ECMA, *"Private Integrated Services Network (PISN) - Circuit Mode Bearer Services - Inter-exchange Signalling Procedures and Protocol"*, ECMA QSIG-BC, June 1997.

[ECTF97] ECTF, *"C.001 Call Control Model, Revision 1.0"*, Enterprise Computer Telephony Forum, 1997.

[EJB99] Sun Microsystems, *"Enterprise JavaBeans Specification Version 1.1"*, Sun Microsystems, December 1999. Available online at: http://java.sun.com/products/ejb/docs.html

[Ekka95] Ekkart *et al.*, *"Tutorial on Message Sequence Charts"*, in O. Haugen (Ed.), SDL 95 with MSC in CASE, September 1995.

[Ells97] J. Ellsberger, D. Hogrefe, A. Sarma, *"SDL: Formal Object-oriented Language for Communicating Systems"*, Prentice Hall, ISBN: 0136213847, 1997.

[Eple90] R. Epley, B.A. Polonsky, S. Yeh, M. Hill, *"Advanced intelligent network services evolution"*, AT&T Bell Lab., Murray Hill, NJ, USA, IEEE International Conference on Communications, ICC'90, Including Supercomm Technical Sessions, SUPERCOMM/ICC '90, 16-19 April 1990.

[Essi92] J. Essinger, *"Electronic Payment Systems"*, Chapman and Hall, ISBN: 0412462907, August 1992.

[ETSI EG 201-722] ETSI, *"Intelligent Network (IN); Service provider access requirements; Enhanced telephony services – Version 1.1.1"*, ETSI, France 1999.

[ETSI EG 201-766] ETSI, *"Intelligent Network (IN); Benchmark services features and network capabilities for the support of IN CS-3 and IN CS-4 – Version 1.1.1"*, ETSI, France, 2001.

[ETSI ETR 055-1] ETSI, *"Universal Personal Telecommunication (UPT); The Service Concept, Part 1: Principles and Objectives"*, ETSI Technical Report, March 1993.

[ETSI ETR 199] ETSI, *"Network Aspects (NA); Enhancement to the modelling and capabilities of the Specialised Resource Function"*, ETSI Technical Report 199, France, June 1995.

[ETSI TR 101-300] ETSI, *"Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON); Description of Technical Issues"*, ETSI Technical Report 101-300 (V2.1.1), ETSI France, October 2000.

[ETSI TR 101-779] ETSI, *"Enhanced SRF phase 2; Enhancement to the modelling and capabilities of the SRF phase 2"*, ETSI Technical Report 101-799 (V1.1.1), France, 2000.

[ETSI TS 101-235] ETSI, *"Specification of Dual Tone Multi-Frequency (DTMF) Transmitters and Receivers; Part 1: General"*, ETSI Technical Specification 101-235, May 2000.

[ETSI TS 101-285] ETSI, *"CAMEL Service Definition - Stage 1"*, ETSI Technical Specification 101-285, August 1999.

[ETSI TS 101-312] ETSI, *"Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON); Network architecture and reference configurations; Scenario 1"*, ETSI Technical Specification 101-312, France, 1998.

[ETSI TS 101-313] ETSI, *"Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON); Network architecture and reference configurations; Phase II: Scenario 1 + Scenario 2"*, ETSI Technical Specification 101-313 (V0.4.2), France, September 2000.

[ETSI TS 101-314] ETSI, *"Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON): Network architecture and reference configurations; TIPHON Release 2"*, ETSI Technical Specification 101 314 (V1.1.1), France September 2000.

[Ever97] K. Evers, W. Lanowski, L. Kersting, I. Mokry, H. Wagner, *"An algorithmic approach for feature interaction detection"*, Siemens AG, Berlin, Germany, IEEE Intelligent Network Workshop, IEEE IN'97, 4-7 May 1997.

[Fayn99] I. Faynberg *et al.*, *"Toward Definition of the Protocol for PSTN-initiated Services Supported by PSTN/Internet Inter-working"*, IETF Internet Draft, draft-faynberg-spirits-protocol-00.txt, October 1999.

[Feyn97] I. Feynberg, L R. Gabuzda, M. P. Kaplan and N. J. Shan, *"The Intelligent Network Standards: Their Application to Services"*, McGraw Hill Series on Telecommunications, ISBN 0-07-021422-0, 1997.

[Filk00] M. Filkenstein, J. Garrahan, D. Shrader and G.Webber, *"The future of the Intelligent Network"*, Telcordia Technologies, IEEE Communications Magazine, pp. 100-106, June 2000.

[FT01a] FT.com, *"Bell Mobility and Nortel Networks Announce CDN$180 Million Contract to Expand Bell Mobility network into Western Canada"*, Business Wire, FT News Archives. Available online at: http://globalarchive.ft.com/globalarchive/articles.html?id= 010807005931&query=invest+in+IP+network# docAnchor010807005931

[FT01b] FT.com, *"NetNumber and MIP Telecom Team to Deliver Simplified VoIP"*, Business Wire, FT News Archives. Available online at: http://globalarchive.ft.com/globalarchive/articles.html?id=010827004318&query=IP+network#docAnchor010827004318

[FT01c] FT.com, *"NEC Unveils New IP Gateway for Remote Office Connectivity"*, Business Wire, FT News Archives. Available online at: http://globalarchive.ft.com/globalarchive/articles.html?id=010821005919&query=IP+network#docAnchor010821005919

[Furc96] A. Furche and G. Wrightson, *"Computer Money: A Systematic Overview of Electronic Payment Systems"*, Die Deutsche Bibliothek, ISBN 3-920993-54-3, 1996.

[Gamm95] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *"Design Patterns"*, Addison-Wesley Pub Co, ISBN: 0201633612, 1995.

[Ganl01] M. Ganley, *"Certification Authorities"*, Thales e-Security White Paper, Available online at: http://thales-esecurity.com/CMS/docs/certification_authories3.pdf

[Gill94] S. Gill, *"AIN service creation and issues"*, NEC America Inc., Irving, TX, USA, Third Annual International Conference on Universal Personal Communications, 27 Sept.-1 Oct. 1994.

[H.225] ITU-T, *"Call Signalling Protocols and Media Stream Packetization for Packet-Based Multimedia Communications Systems"*, International Telecommunication Union Draft Recommendation H.225, Geneva, May 1996.

[H.245] ITU-T, *"Control protocol for multimedia communication"*, International Telecommunication Union Recommandation H.245 (V3), Geneva, January 1998.

[H.323] ITU-T, *"Infrastructure of audiovisual services - Systems and terminal equipment for audiovisual services"*, International Telecommunication Union Recommendation H.323, September 1997.

[Haer99] F. Haerens, *"Using Intelligent Networks in an H.323 Environment"*, ETSI TIPHON 12 TD 27 Philadelphia, March 1999.

[Hale98] S. Halevi, H. Krawczyk, *"Public-Key Cryptography and Password Protocols"*, Proceedings of the Fifth ACM Conference on Computer and Communications Security, 1998.

[Hatt97] L. Hatton, *"Software Failures: Follies and Fallacies"*, IEE Review, pp. 49-52, March 1997.

[Haug95] O. Haugen, *"SDL '95 with MSC in CASE"*, Tutorials 95.09.25, 1995.

[Huan95] Y. Huang and Yi-Min Wang, *"Why optimistic message logging has not been used in telecommunications systems"*, Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing (FTCS '95), pp. 459-463.

[Haya98] K. Hayashi, *"Changes and Deregulation in the Japanese Telecommunications Market"*, IEEE Communications Magazine, pp. 46-53, November 1998.

[Haze98] R. Hazem, *"Call party handling aspects evolution from IN CS-2 to IN CS-3"*, CNET, Bagneux, France, Proceedings of 6th IEEE Intelligent Network Workshop, IN'98. 10-13 May 1998.

[Henn] H. Schulzrinne, J. Rosenberg, *"A Comparison of SIP and H.323 for Internet Telephony"*, Online Tutorials.

[Hink98] J.G. Hinkelmann, P.K. Tollkuehm, *"IN service improvement by call party handling"*, Proceedings of International Conference on Communication Technology Proceedings, ICCT'98. 22-24 Oct. 1998.

[IEPSPat] Patent application reference number GB9811877.1, 2nd June 1998.

[IETF] IETF Internet Engineering Task Force Web Site, http://www.ietf.org

[Ishi98] H. Ishikawa, *"Impact and Preliminary Results of Telecommunications Deregulation in Japan"*, IEEE Communications Magazine, pp. 100-104, July 1998.

[ISOC] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, S. Wolff, *"A Brief History of the Internet"*, Internet Society Web Site. Available online at: http://www.isoc.org

[ISO-IS7498-1] ISO, *"Information technology -- Open Systems Interconnection -- Basic Reference Model: The Basic Model"*, International Standardization Organisation, 1994.

[ITU97] ITU-T, *"Challenges to the Network - Telecoms and the Internet"*, International Telecommunication Union, Geneva, September 1997.

[ITU00] ITU-T, *"IP Telephony Report"*, International Telecommunication Union, Geneva, ISBN 92-61-08621-7, December 2000.

[ITU98a] ITU-T, *"World Telecommunication Development Report 1998; Universal Access"*, International Telecommunication Union, Geneva, March 1998.

[ITU98b] ITU-T, *"General Trends in Telecommunication Reform; World Telecommunication Development Report 1998"*, International Telecommunication Union, Geneva, June 1998.

[Jaco92] I. Jacobson, *"Object-Oriented Software Engineering"*, Addison-Wesley Pub Co; ISBN: 0201544350, 1992.

[JAIN] Sun Microsystems, *"The JAIN APIs: Integrated Network APIs for the Java Platform"*, Sun Microsystems, White Paper, June 2001.
Available online at: http://java.sun.com/products/jain/WP2001.pdf

[Jain00] A. Jain, Farooq M. Anjum, Paolo Missier, and S. Shastry, *"Java Call Control, Coordination, and Transactions"*, IEEE Communications Magazine, pp. 108-114, January 2000.

[Jako01] K. Jakobs, *"The Making of Standards: Looking Inside the Work Groups"*, IEEE Communications Magazine, pp. 102-107, April 2001.

[JAVA] J. Gosling, B. Joy, G. Steele, *"The JAVA™ Language Specification; Version 1.0"*, Addison-Wesley, 1996.

[JAVA131] Sun Microsystems, *"Java™ 2 SDK, Standard Edition Documentation"*, Sun Microsystems, 2001. Available online at: http://java.sun.com/j2se/1.3/docs/

[JTAPI] Sun Microsystems, *"The Java Telephony API (JTAPI) Overview"*, Sun Microsystems JTAPI Specification V1.3, June 1999.
Available online at: http://java.sun.com/products/jtapi/download3.html

[Kala97] R. Kalakota, A. B. Whinston, *"Electronic Commerce, A Manager's Guide"*, Addison-Wesley 1997, ISBN 0-201-88067-9, 1997.

[Kali80] D. Kalish, R. Montague, G. R. Mar, *"Logic: Techniques of Formal Reasoning"*, ISBN: 0155511815, 1980.

[Kali93a] B. S. Kaliski Jr., *"An Overview of the PKCS Standards"*, RSA Laboratories Technical Note, November 1993.
Available online at: ftp://ftp.rsasecurity.com/pub/pkcs/doc/overview.doc

[Kali93b] B. S. Kaliski Jr., *"Some Examples of the PKCS Standards"*, RSA Laboratories, November 1993, Available online at: ftp://ftp.rsasecurity.com/pub/pkcs/doc/examples.doc

[Kapr01] R Kaprinski, *"E-Retailers Bounce Back"*, Internet Weekly, August 2001.
Available online at: http://www.internetweek.com/transtoday01/ttoday080301.htm

[Keiz00] J. de Keijzer, D. Tait, R. Goedman, *"JAIN: A New Approach to Services in Communication Networks"*, Sun Microsystems, Inc., IEEE Communications Magazine, pp. 94-99, January 2000.

[Kell94] Kelly, *"Feature interaction detection using SDL models"*, Network Intelligence Engineering Centre, British Telecom Research Laboratories, Ipswich, UK, IEEE Global Telecommunications Conference, 1994. GLOBECOM '94, 28 Nov.- 2 Dec. 1994.

[Ku94] B.S. Ku, "A reuse-driven approach for rapid telephone service creation", MCI Telecommun. Corp., Richardson, TX, USA, Third International Conference on Software Reuse: Advances in Software Reusability, pp. 64-72, 1-4 November 1994.

[Kumm98] N. Kummer, *"IN SSF Modelling: User's Guide"*, Nortel Telecommunications, 1998.

[Kemp01] J. Kempf *et al.*, "The DIAMETER API", IETF Internet Draft, July 2001.

[Kun97] K. S. Kun, T. M. Han, P. D. Cho, *"Performance evaluation of communications processing module in PSTN access subsystem for interworking with PSDN"*, IEEE International Conference on Communications, ICC'97, 8-12 June 1997.

[Laat00] C. de Laat, *et al.*, *"Generic AAA Architecture"*, IETF Internet Draft, January 2000.

[Laur98] S. St. Laurent, *"Java Cookies"*, McGraw-Hill, ISBN:0070504989, 1998.

[Laza97] A. Lazar, *"Programming Telecommunication Networks"*, Columbia University, New York, IEEE Network, pp.2-12, October 1997.

[Lea99] D. Lea, *"Concurrent Programming in Java"*, Addison Wesley, ISBN: 0201310090, 1999.

[Lebo00] D. Lebovits, *"SIP/IN Interworking"*, IETF Internet Draft, December 2000.

[Lesh98] S. Lesher, *"CUSTOMER: Almon Stroger: Dialing for Dollars"*, Siemens Customer Magazine, Volume VIII, No. 1, May 1998.
Available online at: http://www.siemenscom.com/customer/9801/11.html

[Lin98] Yow-Jian Lin, *"Managing Feature Interactions Telecommunications Software Systems - Guest Editorial"*, University of California,
IEEE Transactions on Software Engineering, Vol. 24, No. 10, October 1998.

[Liu00] Liu and P.Mouchtaris, *"Voice over IP Signaling: H.323 and Beyond"*, Telcordia Technologies, IEEE Communications Magazine, pp. 142-148, October 2000.

[Lodg97] F. Lodge, J. Mitchener, B. Strulo, *"Rapid, User-friendly Open Service Creation"*, TOSCA Deliverable 1, October 1997.

[M.3320] ITU, *"Management requirements framework for the TMN X-interface"*, International Telecommunication Union Recommendation M.3320, April 1997.

[Mage96] T. Magedanz and R. Popescu-Zeletin, *"Intelligent Networks: Basic Technology, Standards and Evolution"*, ISBN 1-85032-293-7, 1996.

[Magee99] J. Magee, J. Kramer, *"Concurrency: State Models & Java Programs"*, John Wiley and Sons, ISBN: 0471987107, March 1999.

[Maho01] D. O'Mahony, H. Tewari, M. Peirce, H. Tewari, *"Electronic Payment Systems for E-commerce"*, Artech House; ISBN: 1580532683, July 2001.

[Marco00] D. Marco, *"Building and Managing the Meta Data Repository: A Full Lifecycle Guide"*, John Wiley & Sons; ISBN: 0471355232, 2000.

[McDo92] C. McDonald, *"Public Networks - Dependable?"*, IEEE Communications Magazine, April 1992.

[McKe00] N. McKeown, *"A Fast Switched Backplane for a Gigabit Switched Router"*, Business Communications Review, White Paper, 2000.

[McMil96] Millan, R.U.Telstra, *"Analysis of congestion control for SCCP traffic and the impact on intelligent network services"*, IEEE Intelligent Network Workshop,
IN'96, 21-24 April 1996.

[MEGACO] IETF Media Gateway Control Protocol Home Page, http://www.ietf.org/html.charters/megaco-charter.html

[Meht98] S. N. Mehta, *"AT&T is Seeking Cause of a Big Outage in Data Network Used by Corporations"*, The Wall Street Journal, p B15., 15th April 1998.

[Melo97] W. Melody (editor), *"Telecom Reform: Principles, Policies and Regulatory Practices"*, Center for Tele-Information, Technical University of Denmark, ISBN 87-7381-071-1, Lyngby, 1997.

[Memm90] W.H. Memmer, *"Telephone switching systems software architecture, where have we been-where are we going"*, IEEE International Conference on Communications, Including Supercomm Technical Sessions. SUPERCOMM/ICC '90. 16-19 , pp. 852-856, April 1990.

[Mi98] Z.Mi and Y.Ying, *"An enhanced DFP Architecture to support IN/Internet Inter-working"*, Delayed Contribution D.657-GEN/11R1, ITU SG-11, May 98, in Mi Zhengkun, *"Architecture and Protocols for IN/INTERNET Inter-working"*, International Conference on Communication Technology, October 1998.

[Moha99] E. A. Mohammed, *"The Growth of Internet Telephony: Legal and Policy Issues"*, First Monday, 1999.

[Moye01] S. Moyer and A. Umar, *"The Impact of Network Convergence on Telecommunications Software"*, Telcordia Technologies, Inc., IEEE Communications Magazine, pp. 78-84, January 2001.

[Naka95] M. Nakamura, *"A method for detecting and eliminating feature interactions using a frame model"*, Fujitsu Labs. Ltd., Kawasaki, Japan, IEEE International Conference on Communications, ICC'95 Seattle, 18-22 June 1995.

[Niits95] Y. Niitsu, O. Mizuno, S. Oyamada, *"Design of an integrated service creation environment for the advanced intelligent network"*, NTT Network Service Syst. Labs., Tokyo, Japan, IEEE Global Telecommunications Conference, GLOBECOM'95, 13-17 Nov. 1995.

[OBri89] O. J. O'Brien and S. G. Webster, *"Intelligent Networks: British Telecom's First Realisation"*, Second IEE National Conference on Telecommunications, pp.407-410, 1989.

[OECD97a] OECD, *"Dismantling the Barriers to Global Electronic Commerce"*, Organisation for Economic Co-operation and Development, Background paper, Paris, 1997. Available online at: http://www.oecd.org

[OECD97b] OECD, *"Electronic Commerce: Opportunities and Challenges for Government"*, Organisation for Economic Co-operation and Development, Paris, 1997. Available online from http://www.oecd.org

[Oftel99] Oftel, *"Guidelines on Interconnection and Inter-operability"*, Issued by the Director General of Telecommunications, UK Office of Telecommunications, July 1999. Available online at: http://www.oftel.gov.uk/publications/1999/competition/gii799.htm

[Okam89] T. Okamoto, K. Ohta, *"Disposable zero-knowledge authentication and their applications to untraceable electronic cash"*, Advances in Cryptography - Crypto 1989.

[Okam93] T. Okamoto, E. Fujisaki, *"On Comparison of Practical Digital Signature Schemes"*, NTT Review 5, No.1, January 1993.

[OMA97] OMA, *"A discussion of the Object Management Architecture"*, Object Management Group, January 1997.

[ORei98] O'Reilly-Roche, *"Call party handling using the connection view state approach, a foundation for intelligent control of multiparty calls"*, Annapolis, MD, USA, IEEE Communications Magazine, pp. 60-66, June 1998.

[Orfa99] R. Orfali, D. Harkey, J. Edwards, *"Client/Server Survival Guide"*, 3rd Edition, John Wiley & Sons; ISBN: 0471316156, 1999.

[P508] EURESCOM Project P508, *"Evolution, Migration Paths and Interworking to TINA"*, EURESCOM Project P508 Deliverable 3, EURESCOM Web Site. December 1997.

[P1110] EURESCOM Project P1110, *"Open Service Access: Advantages and opportunities in service provisioning on 3G Mobile Networks"*, EURESCOM Web Site, Available online at:

http://www.eurescom.de/public/projects/P1100-series/P1110/default.asp# Project %20 Information

[Pala00] C. Palamidessi (editor), *"CONCUR 2000. Concurrency Theory"*, Springer-Verlag Berlin and Heidelberg GmbH & Co. KG, ISBN: 3540678972, 2000.

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

[ParlBuss99] The Parlay Group, *"Parlay API, Business Benefits White Paper"*, The Parlay Group, June 1999.

[ParlPR] The Parlay Group, *"Parlay Group Triples Membership since June"*, Parlay Group Press Release, November 2nd 2000.

[ParlSeq99] The Parlay Group, "Parlay API Specification V1.2: Sequence Diagrams", The Parlay Group, September 1999.

[ParlFw99] The Parlay Group, "Parlay API Specification V1.2: Framework Interfaces", The Parlay Group, September 1999.
[ParlUg99] The Parlay Group, "Parlay API Specification V1.2: User Guide", The Parlay Group, September 1999.

[ParlSpec00] The Parlay Group, *"The Parlay API Specifications Version 2.1"*, The Parlay Group, June 2000. Available online at: http://www.parlay.org.

[Part96] C. Partridge, *et al.*, *"A Fifty-Gigabit* per s*econd IP Router",* Submitted to IEEE/ACM Transactions on Networking, 1996.

[Peng98] Y. Peng, *"Detecting feature interactions at specification stage"*, Concordia Univ., Montreal, Que., Canada, Proceedings of 7th IEEE Intelligent Network Workshop, IN'98, 10-13 May 1998.

[Ponc98] O. Poncin, *"Examining How Operators Are Developing Strategies for Successfully Delivering Converged Services to Increase Acquisition and Reduce Churn"*, Proceedings of 7th International Forum on Fixed-Mobile Convergence, London, September 1998.

[PSS99] Fujitsu, INPRISE *et al.*, *"Persistent State Service V 2.0"*, Joint Revised Submission from Fujitsu Limited, INPRISE Cooperation *et al.*, August 1999.

[Q.701] ITU, *"Functional description of the message transfer part (MTP) of Signalling System No. 7"*, ITU-T Recommendation Q.701 Geneva, March 1993.

[Q.711] ITU, "*Signalling System No. 7 – Functional Description of the Signalling Connection Control Part*", International Telecommunication Union Recommendation Q.711, Geneva, March 1993.

[Q.712] ITU, "*Signalling System No. 7 – Definition and Function of SCCP Messages*", International Telecommunication Union Recommendation Q.712, Geneva, March 1993.

[Q.713] ITU, "*Signalling System No. 7 – SCCP Formats and Codes*", International Telecommunication Union Recommendation Q.713, Geneva, March 1993.

[Q.714] ITU, "*Signalling System No. 7 – Signalling Connection Control Part Procedures*", International Telecommunication Union Recommendation Q.714, Geneva, March 1993.

[Q.716] ITU, "*Signalling System No. 7 – Signalling Connection Control Part (SCCP) Performance*", International Telecommunication Union Recommendation Q.716, Geneva, March 1993.

[Q.721] ITU, "*Functional Description of the Signalling System No. 7 Telephone User Part (TUP)*", International Telecommunication Union Recommendation Q.721, Geneva, March 1993.

[Q.722] ITU, "*General Function of Telephone Messages and Signals*", International Telecommunication Union Recommendation Q.722, Geneva, March 1993.

[Q.725] ITU, "*Signalling System No. 7 – Signalling Performance in the Telephone Application*", International Telecommunication Union Recommendation Q.725, Geneva, March 1993.

[Q.762] ITU, "*General Function of Messages and signals of the IDN User Part of Signalling System No. 7*", International Telecommunication Union Recommendation Q.725, Geneva, March 1993.

[Q.763] ITU, "*Formats and Codes of the ISDN User Part Of Signalling System No. 7*", International Telecommunication Union Recommendation Q.725, Geneva, March 1993.

[Q.764] ITU, "*Signalling System No. 7 – ISDN User Part Signalling Procedures*", International Telecommunication Union Recommendation Q.725, Geneva, March 1993.

[Q.766] ITU, "*Performance Objectives in the Integrated Services Digital Network Application*", International Telecommunication Union Recommendation Q.766, Geneva, March 1993.

[Q.767] ITU, "*Application of the ISDN User Part of CCITT Signalling System No. 7 or International ISDN Interconnections*", International Telecommunication Union Recommendation Q.725, Geneva, February 1991.

[Q.771] ITU, "*Signalling System No. 7 – Functional Description of Transaction Capabilities*", International Telecommunication Union Recommendation Q.725, Geneva, March 1993.

[Q.775] ITU, "*Signalling System No. 7 – Guidelines for using Transaction Capabilities*", International Telecommunication Union Recommendation Q.725, Geneva, March 1993.

[Q.931] ITU, "*Digital Subscriber Signalling System No. 1 (DSS 1) – ISDN User-Network Interface Layer 3 Specification for Basic Call Control*", International Telecommunication Union Recommendation Q.931, Geneva, March 1993.

[Q.1200] ITU, "Q-SERIES Intelligent Network Recommendation Structure", International Telecommunication Union, Recommendation Q.1200, March 1993.

[Q.1201] ITU, *"Principles of the Intelligent Network Architecture"*, International Telecommunication Union Recommendation Q.1201, Geneva, October 1992.

[Q.1202] ITU, *"Intelligent Network Service Plane Architecture"*, International Telecommunication Union Recommendation Q.1202, Geneva, October 1992.

[Q.1203] ITU, *"Intelligent Network Global Functional Plane Architecture"*, International Telecommunication Union Recommendation Q.1203, Geneva, October 1992.

[Q.1204] ITU, *"Intelligent Network Distributed Functional Architecture"*, International Telecommunication Union Recommendation, Q.1204, Geneva, March 1993.

[Q.1205] ITU, *"Intelligent Network Physical Plane Architecture"*, International Telecommunication Union Recommendation Q.1205, Geneva, March 1993.

[Q.1208] ITU, *"General Aspects of the Intelligent Network Application Protocol"*, International Telecommunication Union Recommendation Q.1208, Geneva, March 1993.

[Q.1211] ITU, *"Introduction to the Intelligent Network Capability Set 1"*, International Telecommunication Union Recommendation Q.1211, Geneva, March 1993.

[Q.1213] ITU, *"Global Functional Plane Architecture for Intelligent Network CS-1"*, International Telecommunication Union Recommendation Q.1213, Geneva, March 1993.

[Q.1214] ITU, *"Distributed Functional Plane for Intelligent Network CS-1"*, International Telecommunication Union Recommendation Q.1214, Geneva, March 1993.

[Q.1221] ITU, *"Introduction to the Intelligent Network Capability Set 2"*, International Telecommunication Union Recommendation Q.1221, Geneva, September 1997.

[Q.1223] ITU, *"Global Functional Plane Architecture for Intelligent Network CS-2"*, International Telecommunication Union Draft Recommendation Q.1223, Geneva, September 1997.

[Q.1224] ITU, *"Distributed Functional Plane for Intelligent Network CS-2"*, International Telecommunication Union Draft Recommendation Q.1224, Geneva, September 1997.

[Q.1228] ITU, *"General Aspects of the Intelligent Network Application Protocol Capability Set 2"*, International Telecommunication Union Draft Recommendation Q.1228, Geneva, September 1997.

[Q.1244] ITU, *"Distributed functional plane for Intelligent Network Capability Set 4"*, International Telecommunication Union Recommendation Q.1244, July 2001.

[QOSNet] Qos Networks, *"QoS Networks Completes Initial Funding of $100 Million and Launches Global Content Delivery Network"*, QoS Networks Press Release, 6 September 2000.

[Quantitative97] Quantitative Data Systems, *"The Object: CORBA Basics"*, Quantitative Data Systems, Tutorial, December 1997.

[Rational] Rational Software Corporation, *"Unified Modelling Language"*.
Available online at: http://www.rational.com/

[Rao00] S. Rao, L. Alvisi and H. M. Vin, *"The Cost of Recovery in Message Logging Protocols"*, IEEE Transactions on Knowledge and Data Engineering, Vol. 12, No. 2; pp. 160-173, March/April 2000.

[RFC760] Postel, J., *"Internet Protocol"*, RFC 760, USC/Information, Sciences Institute, January 1980.

[RFC768] IETF, *"User Datagram Protocol"*, J. Postel, IETF, August 1980.

[RFC793] IETF, *"Transmission Control Protocol - Protocol Specification"*, IETF, 1981.

[RFC1098] IETF, *"A Simple Network Management Protocol (SNMP)"*, J. Case *et al.*, STD 16, RFC 1098, April 1989.

[RFC1155] IETF, *"Structure and Identification of Management Information for TCP/IP-based internets"*, Rose, M. and K. McCloghrie, STD 16, RFC 1155, May 1990.

[RFC1212] IETF, *"Concise MIB Definitions"*, Rose, M. and K. McCloghrie, STD 16, RFC 1212, March 1991.

[RFC1213] IETF, *"Management Information Base for Network Management of TCP/IP-based internets: MIB-II"*, K. McCloghrie, RFC 1213, March 1991.

[RFC1441] IETF, *"Introduction to version 2 of the Internet-standard Network Management Framework"*, J. Case, RFC 1441, April 1993.

[RFC1591] J. Postel, *"Domain Name System Structure and Delegation"*, IETF, March 1994.

[RFC1831] IETF, *"RPC: Remote Procedure Call Protocol Specification Version 2"*, R. Srinivasan, Sun Microsystems, August 1995.

[RFC1889] IETF, *"RTP: A Transport Protocol for Real-Time Applications"*, H. Schulzrinne, January 1996.

[RFC2138] IETF, *"Remote Authentication Dial In User Service (RADIUS)"*, C. Rigney, April 1997.

[RFC2205] IETF, *"Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification"*, R. Braden, Ed., *et al.*, IETF Network Working Group, September 1997.

[RFC2327] IETF, *"SDP: Session Description Protocol"*, M. Handley, *et al.*, Network Working Group, IETF RFC 2327, April 98.

[RFC2475] IETF, *"An Architecture for Differentiated Services"*, S. Blake, IETF Network Working Group, December 1998.

[RFC2543] IETF, *"SIP: Session Initiation Protocol"*, M. Handley et.al, IETF, Network Working Group, March 1999.

[RFC2570] IETF, *"Introduction to Version 3 of the Internet-standard Network Management Framework"*, J. Case, *et al.*., RFC 2570, April 99

[RFC2705] M. Arango, A. Dugan, I. Elliott, C. Huitema, S. Pickett, *"Media Gateway Control Protocol (MGCP) Version 1.0"*, Network Working Group, IETF RFC 2705, October 1999.

[RFC2748] IETF, *"The COPS (Common Open Policy Service) Protocol"*, D. Durham, Ed., *et al.*, IETF RFC 2748, January 2000.

[RFC2753] IETF, *"A Framework for Policy-based Admission Control"*, R. Yavatkar, *et al.*, IETF RFC2753, January 2000.

[RFC2995] IETF, *"Pre-SPIRITS Implementations of PSTN-initiated Services"*, I. Faynberg, *et al.*, Network Working Group, IETF RFC 2995, November 2000.

[RFC3136] IETF, *"The SPIRITS Architecture"*, L. Slutsman, *et al.*, Network Working Group, IETF RFC 3136, June 2001.

[Rive83] R. Rivest, A. Shamir, L. Adleman, *"A method for obtaining digital signatures and public-key cryptosystems"*, Communications of the ACM 21/2, pg. 120-126, 1978.

[RMI] Sun Microsystems, *"Java Remote Method Invocation Specification"*, Sun Microsystems, Available online at: ftp://ftp.java.sun.com/docs/j2se1.3/rmi-spec-1.3.pdf

[Rosc97] A.W. Roscoe, *"The Theory and Practice of Concurrency"*, Prentice Hall; ISBN: 0136744095, October 1997.

[RSA98] RSA Laboratories, *"PKCS #1 v2.0: RSA Cryptography Standard"*, RSA Laboratories, October 1998. Available online at: ftp://ftp.rsasecurity.com/pub/pkcs/doc/pkcs-1v2.doc

[Rumb91] J. Rumbaugh *et al..*, *"Object-Oriented Modelling and Design"*, Prentice-Hall, 1991.

[Russ98] T. Russel, *"Signalling System #7, Second Edition"*, McGraw-Hill Telecommunications, ISBN 0-07-058032-4, 1998.

[Scer97] L.J. Scerbo, *"The US Telecom Reform Act of 1996-, Will too many cooks ruin the stew"*, 19th International Telecommunications Energy Conference, 1997. INTELEC 97, 19-23 Oct. 1997.

[Scho98] U. Schoen, J. Hamann, A. Ugel, H. Kurzawa, C. Schmidt, *"Convergence between public switching and the Internet"*, Siemens AG, Germany, IEEE Communications Magazine, pp. 50-65, January 1998.

[Schu98] H. Schulzrinne, *"The Session Initiation Protocol (SIP) - Tutorial on SIP"*, Columbia University, New York, 1998.

[SIP] The SIPCenter.com Home page, http://www.sipcenter.com

[SOFTSWITCH] SoftSwitch Consortium Web Site, http://www.softswitch.org/

[Solo98] C. Solomonides, M. Searle, *"IN and the INternet"*, University College London, London Telecommunications Research Symposium, London, July, 1998.

[Solo99a] C. Solomonides, M. Searle, *"An Intelligent Network E-Commerce Protocol"*, University College London, Utrecht, Netherlands, 38th European Telecommunications Congress, Utrecht, The Netherlands, August 1999.

[Solo99b] C. Solomonides, M. Searle, *"Relevance of Existing Intelligent Network Infrastructure to the Internet"*, University College London, Lecture Notes in Computer Science, Springer-Verlag, 6th International Conference on Intelligence and Services in Networks, IS&N'99, Barcelona, Spain, 1999

[Solo00a] C. Solomonides, M. Searle, *"Evolution Towards the TINA Service Architecture Through PINT and Parlay"*, 6th International Conference on Intelligence in Networks, 17-20 January 2000, Palais Des Congres D'Archachon, Bordeaux, France, 2000.

[Solo00b] C. Solomonides, M. Searle, *"Network Intelligence, APIs and Service Creation"*, University College London, 39th European Telecommunications Congress, Limerick - Ireland, August 2000.

[Solo00c] C. Solomonides, M. Searle, *"Intelligent Network Application Programming Interface Server Architecture"*, University College London, IEEE Intelligent Network Workshop, IN'00, Cape Town, South Africa, May 2000.

[Solo00d] C. Solomonides, M. Searle, *"Application Server Architecture for Open Networks"*, University College London, London Telecommunications Symposium, London, September 2000.

[Somm01] I. Sommerville, "Software Engineering", 6th Edition, Addison-Wesley Pub Co; ISBN: 020139815X, 2001.

[Song95] X. (Carol) Song, Jane W.S. Liu, *"Maintaining Temporal Consistency: Pessimistic vs. Optimistic Concurrency Control"*, IEEE Transactions on Knowledge and Data Engineering 1041-4347, Vol. 7, No. 5; pp. 786-796, October 1995.

[SPIRITS] IETF SPIRITS Working Group Home Page, http://www.ietf.org/html.charters/spirits-charter.html

[Stal99] W. Stallings, *"Snmp, SnmpV2, SnmpV3, and Rmon 1 and 2"*, Addison-Wesley Pub Co, ISBN: 0201485346, January 1999.

[Stev94] W. R. Stevens, *"TCP/IP Illustrated, Volume 1, The Protocols"*, Addison-Wesley Professional Computing Series, 1994.

[Stev99] M. Stevens, *"Policy Framework"*, IETD Internet Draft, draft-ietf-policy-framework-00.txt, September 1999.

[Stro85] R. E. Strom and S. Yemini, *"Optimistic recovery in distributed systems"*, ACM Transactions on Computer Systems, Vol. 3, No. 3; pp. 204-226, August 1985.

[Stro99] K.G. Strouse, *"Marketing Telecommunications Services: New Approaches for a Changing Environment"*, Artech House Telecommunications Library, Artech House; ISBN: 158053015X, June 1999.

[Suzu93] S. Suzuki, *"IN rollout in Japan"*, NTT, Tokyo, Japan, IEEE Communications Magazine, p. 48-55, March 1993.

[SWIFT] SWIFT Web Site, http://www.swift.com

[Tarj97] P. Tarjanne, *"Telecoms and the Internet: Convergence or Collision?"*, Secretary-General, International Telecommunication Union, ISS 1997, Toronto, 1997.

[Tau3.5] Telelogic Tau 3.5, SDT and ITEX Tools, http://www.telelogic.se

[TeRe00] Technology Review Magazine, *"Trailing Edge: No Operator, Please"*, Trailing Edge Technology Review, January/February 2000, Available online at: http://www.techreview.com/magazine/jan00/trail_edge.asp

[Thom95] A. Thomasian, *"Checkpointing for Optimistic Concurrency Control Methods"*, IEEE Transactions on Knowledge and Data Engineering 1041-4347, Vol. 7, No. 2, pp. 332-339, April 1995.

[Thom00] W. C. Thomas, G. Thiruvathukal, *"High Performance Java Computing"*, Prentice Hall; ISBN: 0130161640, June 2000.

[TINA-BM] TINA-C, *"TINA-C Business Model and Reference Points Version 4.0"*, H. Mulder (editor), May 1997.

[TINA-CA] TINA-C, *"Computing Architecture Version"*, TINA Consortium, December 1994.

[TINA-DPE] TINA-C, *"Engineering Modelling Concepts (DPE Architecture)"*, Version 2.0, TINA Consortium, December 1994.

[TINA-GA] TINA-C, *"Overall Concepts and Principles of TINA"*, TINA Consortium, February 1995.

[TINA-GU] TINA-C, *"Guidelines to TINA: Overall Concepts and Principles to TINA"*, TINA Consortium, Version 1.0, February 1995.

[TINA-MA] TINA-C, "Management Architecture", TINA Consortium, December 1994.

[TINA-NA] TINA-C, "Network Architecture", TINA Consortium, December 1994.

[TINA-SA] TINA-C, *"Service Architecture Version: 5.0"*, L. Kristiansen (editor), June 1997.

[TIPHON] ETSI, *"Telecommunications and Internet Protocol Harmonization over Networks (TIPHON)"*, TIPHON Web Site, http://www.etsi.org/tiphon/

[TOSCA-IS] TOSCA, "TOSCA: Initial Specification of Process Architecture", F. Costello, February 1997.

[UML99] OMG, "Unified Modelling Language Specification Version 1.3", Object Management Group, June 1999.

[UPT] ETSI UPT Web Site, http://www.etsi.org/user/archives/universal.htm

[USDOC] US Department of Commerce, *"The Emerging Digital Economy"*, August 1998.

[Veer99] M. Veeraraghavan, M. Karol, *"Internetworking Connectionless and Connection-Oriented Networks"*, Polytechnic University, Lucent Technologies, IEEE Communications Magazine, pp. 130-138, December 1999.

[Vemu99] K.Vemuri, *"Call Model Integration Framework"*, Lucent Technologies, Inc, IETF Internet Draft, I-D <draft-vemuri-cmi-framework-00.txt>, IETF, January 1999.

[Veni98] I. Venieris and H. Hussmann, *"Intelligent Broadband Networks"*, John Wiley & Sons Ltd, ISBN 0-471-98094-3, 1998.

[Veni00] I. Venieris (editor), F.Zizza, T. Magedanz, F. Zizza, *"Object Oriented Software Technologies in Telecommunications: From Theory to Practice"*, John Wiley & Sons; ISBN: 0471623792, June 2000.

[Venk97] S. S. Venkatesan, T. Tong-Ying Juang and Sridhar Alagar, *"Fault Tolerance/Reliability Optimistic Crash Recovery Without Changing Application Messages"*, IEEE Transactions on Parallel and Distributed Systems, IEEE Vol. 8, No. 3; pp. 263-271, March 1997.

[VISA] VISA, *"Emerging Acceptance: Overview"*, Available online at: http://www.visa.com/nt/suppliers/vendor/accept/main.html

[Voll00a] J. Vollbrecht, *et al.*, *"AAA Authorization Framework"*, IETF Internet Draft, draft-irtf-aaaarch-authorization-framework-00.txt, January 2000.

[Voll00b] J. Vollbrecht, *et al.*, *"AAA Authorization Application Examples"*, IETF Internet Draft, draft-irtf-aaaarch-authorization-apps-00.txt, January 2000.

[Voss97a] G. Voss, *"JavaServer Technologies: Part I"*, JavaSoft, April 1997. Available online at: http://developer.java.sun.com/developer/technicalArticles/Servlets/ JavaServerTech1/index.html

[Voss97b] G. Voss, *"JavaServer Technologies: Part II"*, JavaSoft, May 1997. Available online at: http://developer.java.sun.com/developer/technicalArticles/Servlets/ JavaServerTech2/index.html

[Walk97] P. Walker, *"How to Compete and Connect: Setting the Business and Regulatory Context"*, IEE Colloquium on "How to Compete and Connect: Understanding the Engineering of Telecommunications Network Interconnection", Digest No: 1997/179, 1997.

[Wang01] Pi-Chung Wang, *et al.* *"A Fast IP Routing Lookup Scheme"*, IEEE Communications Magazine Letters Vol. 5 No. 3, p. 125, March 2001.

[Ward95] K. Ward, *"Impact of Network Interconnection on Network Integrity"*, British Telecommunications Engineering, Vol. 13, pp. 296-303, January 1995.

[Will94] S. Willliams, C. Kindel, *"The Component Object Model: A Technical Overview"*, Microsoft Corporation, October 1994.

[Wong99] W. Wong, *"Telcos to push IP telephony in 1999"*, CNet News.com, January 1999. Available online at: http://news.cnet.com/news/0-1004-200-336979.html

[WResInc98] World Research Inc., *"The Internet Transaction Survey Results"*, 1998. Available online at: http://www.survey.com/transresults.html

[Z.100] ITU, *"Specification and description language (SDL)"*, International Telecommunication Union Recommendation Z.100, November 1999.

[Zhen98] M. Zhengkun, *"Architecture and protocols for IN,Internet interworking"*, Nanjing Univ. of Posts & Telecommun., China, International Conference on Communication Technology Proceedings, ICCT '98, 22-24 Oct. 1998.

[Zolz92] P. Zolzettich, A.R. Ephrath, *"Customized service creation, a new order for telecommunication services"*, IEEE 11th Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM'92, 4-8 May 1992.

APPENDIX A

LIST OF ACRONYMS & ABBREVIATIONS

AAA	Authentication Authorisation Accounting
API	Application Programmer Interface
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
BCM	Basic Call Model
BCP	Basic Call Process
BCSM	Basic Call State Model
BCUP	Basic Call Unrelated Process
BES	Back End Server
B-IP	Broadband Intelligent Peripheral
BRI	Basic Rate Interface
C/B GF	Call/Bearer Control Gateway Function
CA	Certification Authority
CAMEL	Customised Applications for Mobile Enhanced Logic
CAO	Call Association Object
CC	Call Configuration
CCAF	Call Control Agent Function
CCF	Connection Control Function
CCF	Call Control Function
CCS7	Common Channel Signalling System No. 7
CGI	Common Gateway Interface
CMF	Call Manager Function
CMI	Call Model Integration
COPS	Common Open Policy Service Protocol
CORBA	Common Object Request Broker Architecture
CP	Connection Point
CPE	Customer Premises Equipment
CPH	Call Party Handling
CS	Capability Set
CSA	Call Segment Association
CSCV	Call Segment Connection View
CSE	CAMEL Service Environment
CT	Computer Telephony
CTI	Computer Telephony Integration
CUSF	Call Un-Related Service Function
CV	Connection View
DCOM	Distributed Component Object Model
DFP	Distributed Functional Plane
DNS	Domain Name Server
DOT	Distributed Object Technologies
DP	Detection Point
DPE	Distributed Processing Environment
DPNSS	Digital Private Network Signalling System
DSS1	Digital Subscriber Signalling No.1

DTMF	Dual Tone Multiple Frequency
EJB	Enterprise Java Beans
EPS	Electronic Payment System
ETSI	European Telecommunications Standardisation Institute
FE	Functional Entity
FEA	Functional Entity Action
FEAM	Functional Entity Access Manager
FSM	Finite State Machine
FTP	File Transfer Protocol
GFP	Global Functional Plane
GK	Gatekeeper
GMS	Generic Messaging Service
GMSC	Gateway Mobile Switching Centre
GPRS	General Packet Radio Service
G-SCF	Gateway-Service Control Function
GSL	Global Service Logic
GSM	Global System Mobile
GW	Gateway
H.323 GKF	H.323 Gatekeeper Function
HLR	Home Location Register
HTTP	Hyper-text transfer protocol
IAF	Intelligent Access Function
ICW	Internet Call Waiting
IDL	Interface Definition Language
IEPS	Intelligent Electronic Payment System
IETF	Internet Engineering Task Force
IF	Information Flow
IN	Intelligent Network
INAP	Intelligent Network Application Part
INCM	Intelligent Network Conceptual Model
IN-SSM	IN-Switching State Model
IP	Internet Protocol
IPCF	IP Control Function
IS	Internet Shop
ISDN	Integrated Services Digital Network
ISO	International Organisation for Standardisation
ISP	Internet Service Provider
ISUP	ISDN User Part
ITU	International Telecommunication Union
IVR	Interactive Voice Response
JAIN	Java APIs for Integrated Networks
JCAT	JAIN Coordination and Transactions
JCC	JAIN Call Control
JSLEE	JAIN Service Logic Execution Environment
JTAPI	Java Telephony API
MACF	Multiple Association Control Function
MAP	Mobile Application Part
MG	Media Gateway
MGC	Media Gateway Controller
MGCP	Media Gateway Control Protocol
MGF	Management Gateway Function
MIB	Management Information Base
MSC	Message Sequence Chart
MTP	Message Transfer Part
NAP	Network Access Point

NAS	Network Access Server
N-ISDN	Narrowband-ISDN
NMS	Network Management System
NNI	Network Node Interface
NO	Network Operator
O_BCSM	Originating Basic Call State Model
OAM	Operations Administration and Management
OMA	Object Management Group Architecture
ORB	Object Request Broker
OSI	Open Systems Interconnection
PBX	Private Branch Exchange
PDP	Policy Decision Point
PE	Physical Entity
PEP	Policy Enforcement Point
PIC	Points in Call
PIN	Personal Identification Number
PINT	PSTN and Internet Internetworking
PKI	Public-Key Infrastructure
PLMN	Public Land Mobile Network
PNNI	Private network-network interface
POI	Point of Initiation
PoP	Point of Presence
POR	Point of Return
POTS	Plain Old Telephone Service
PP	Physical Plane
PRI	Primary Rate Interface
PSS	Persistent State Service
PSTN	Public Switched Telephony Network
QoS	Quality of Service
RA	Registration Authority
RADIUS	Remote Authentication and Dial-In User Service
RM	Resource Manager
RMI	Remote Method Invocation
RSVP	Resource ReserVation Protocol
SAO	Single Association Object
SCCP	Signalling Connection Control Part
SCE	Service Creation Environment
SCEF	Service Creation Environment Function
SCF	Service Control Function
SCGF	Service Control Gateway Function
SCN	Switched-circuit Network
SCP	Service Control Point
SCUF	Service Control User Agent Function
SDF	Service Data Function
SDL	Specification and Description Language
SDP	Service Data Point
SDP	Session Description Protocol
SE	Service Environment
SIB	Service Independent Building Block
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SMAF	Service Management Agent Function
SMF	System Management Function
SMP	Service Management Platform
SN	Service Node

SNMP	Simple Network Management Protocol
SP	Service Plane
SPC	Stored Program Control
SPIRITS	Service in the PSTN/IN Requesting InTernet Services
SQL	Structured Query Language
SRF	Specialised Resource Function
SS7	Signalling System No. 7
SSF	Service Switching Function
SSM	Switching State Model
SSP	Service Switching Point
SCUF	Service Control User Agent Function
STP	Signalling Transfer Point
T_BCSM	Terminating Basic Call State Model
TC	Transaction Capability
TCAP	Transaction Capabilities Application Part
TCP	Transmission Control Protocol
TINA	Telecommunications Information Network Architecture
TINA-C	TINA Consortium
TIPHON	Telecommunications and Internet Protocols Harmonisation over Networks
TMN	Telecommunication Management Network
TMN X	The TMN X-Interface
TN	Transaction Number
TUP	Telephony User Part
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
UI	User Interaction
UML	Unified Modelling Language
UMTS	Universal Mobile Telephony Service
UPT	Universal Personal Telecommunication
VoIP	Voice-over-IP
VPN	Virtual Private Network