An Empirical, in-depth Investigation into Service Creation in H.323 Version 4 Networks

THESIS

Submitted in fulfilment of
the requirements for the
degree of

MASTER OF SCIENCE

In the Department of
Computer Science

Of Rhodes University

By

Jason Barry Penton

February 2003

KEYWORDS

H.323, Internet Protocol, Legacy, Network Entity, Next Generation Network (NGN), Supplementary Service, Generic Extensibility Framework (GEF), H.450, H.460, H.225, H.245, RTP, RTCP, Voice over Internet Protocol (VoIP), Public Switched Telephone Network (PSTN), Service Creation, H.323 Annex K, Service Control, Service Management

ABSTRACT

Over the past few years there has been an increasing tendency to carry voice on IP networks as opposed to the PSTN and other switched circuit networks. Initially this trend was favoured due to reduced costs but occurred at the expense of sacrificing the quality of the voice communications. Switched circuit networks have therefore remained the preferred carrier-grade voice communication network, but this is again changing. The advancement in improved quality of service (QoS) of real-time traffic on the IP network is a contributing factor to the anticipated future of the IP network supplying carrier-grade voice communications. Another contributing factor is the possibility of creating a new range of innovative, state-of-the-art telephony and communications services that acquire leverage through the intelligence and flexibility of the IP network. The latter has yet to be fully explored. Various protocols exist that facilitate the transport of voice and other media on IP networks. The most well known and widely supported of these is H.323. This work presents and discusses H.323 version 4 service creation. The work also categorises the various H.323 services and presents the mechanisms provided by H.323 version 4 that have facilitated the development of the three services I have developed, EmailReader, Telgo323 and CANS.

ACKNOWLEDGEMENTS

I would like to start by thanking my supervisor Alfredo Terzoli for the incredible amount of time and effort he has dedicated to helping me with this project. Not only has he helped me in this project but has played a major role in my growth and maturity as an academic researcher. Thanks must also go to Prof. Peter Clayton for his input at the many project meetings we had during the course of the project.

Special thanks to two great friends I have met during the course of the project, Bill Tucker and Meryl Glaser. They have provided a significant amount of input into the study, and were the driving force behind the implementation of Telgo323. On behalf of Meryl and myself I would like to also thank Bill for doing the numerous radio interviews regarding Telgo323!

Thanks to the staff and postgraduate students of the Computer Science Department. They are an incredible group of people whom I strongly admire. They have all made my stay at Rhodes University an unforgettable and enjoyable experience.

This project would not have been possible if it were not for the financial support provided by Telkom. Thanks to my Telkom industry supervisors, David Browne, Baron Peterssen and Hein Ferreira, for the support they have given me over the years.

Finally I'd like to thank Cynthia Kulongowski for finding the time to professionally proofread the drafts of this document.

This project is dedicated to my family who have always been behind me and taught me to believe in myself and whatever I do.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

GENERAL INTRODUCTION

"All truth passes through three stages. First, it is ridiculed.
Second, it is violently opposed. Third, it is accepted as being self-evident."

Arthur Schopenhauer

This chapter discusses the motivation for the study and briefly introduces the research methodology. Several questions that provided direction in the study are also introduced, while the results and discussions around these questions are presented in Chapter 8. Finally, I provide an overview of the study by introducing the material covered in each of the chapters that follow.

1.1 Introduction and Motivation

The transfer of voice traffic over packet networks, and especially voice over IP, is growing rapidly. Initially, the major reason for this trend was cost savings on long-distance telephone calls. However, the convergence between the traditional Public Switched Telephone Network (PSTN) and the IP data network, often referred to as the Next Generation Network (NGN), has opened the door to many novel applications. Such applications often involve the combination of voice and data, such as instant messaging and click-to-dial.

In any communications network, some form of control is required to set up and tear down communications channels. In the telecommunications arena such control is defined by a signalling protocol. The signalling protocol defines the operations required to control a specific communications medium. For example, the communications control protocol present within the traditional public telephone network is called Signalling System number 7 (SS7). SS7 defines the set-up and teardown of bearer channels responsible for carrying voice over the telephone network. SS7 does not include definitions of the various voice services, like call transfer, call hold and call waiting. The control of these services is addressed by another protocol known as Intelligent Network (IN).

In the IP environment there are numerous voice signalling protocols, known as Voice over IP (VoIP) protocols. The most well known and widely used of these protocols is H.323, entitled *Packet-based Multimedia Communication Systems*. H.323 is currently responsible for more than a billion minutes of IP voice traffic globally per year and it continues to grow each day. A major factor contributing to the large market support of H.323 is its ability to interwork with the traditional PSTN. Its origins in the International Telecommunications Union (ITU) have resulted in its signalling operations being closely mapped to protocols responsible for signalling in the PSTN, for example SS7 and QSIG, lending H.323 to easier and more reliable PSTN/IP integration.

H.323 also addresses service control as of its latest ratification (version 4). Advanced service creation was overlooked in earlier versions and was a heavy point of criticism for H.323. The

inclusion of various mechanisms to aid service creation in H.323 networks has opened the door to a new range of innovative services. Additionally, these services can be made available to PSTN users, meaning that these new services, residing on an H.323 packet switched network, can be utilised by standard public telephones or cellular phones.

But why create new services in the IP network and then make them available to the PSTN? And, why not create the new services directly within the IN, the service layer of the PSTN?

The answer to these questions is based on a number of drawbacks with IN. Its lack of vendor independence, limited intelligence, slow and expensive service integration and poor service management (i.e., end users cannot configure services themselves) are some examples of the drawbacks it suffers [7].

With this in mind, I embarked on this project to fully explore service creation in H.323 version 4 networks ensuring the appropriate services could be made available to traditional PSTN users. Ultimately, this means that new and advanced voice services, which reach far beyond the intelligence of the IN, can be developed and integrated into packet-based networks, using H.323 [44]. Furthermore, traditional PSTN voice services like call hold and call transfer can be developed and integrated into H.323 networks and made available to the PSTN at a fraction of the cost and time of integrating them into the IN.

1.2 Research Methodology

To conduct this study, a fully functional and feature-rich H.323 (version 4) network was integrated into the active campus network at Rhodes University. A description of this environment can be found in Chapter 2. A detailed discussion concerning the deployment of the H.323 environment can be found in a document by Jason Penton [42].

With the H.323 environment in place, the various services that can be expected in next-generation telephony networks were identified and categorised according to their design, functionality and interaction with other H.323 network entities. Such services include the traditional telephony services, like call hold and call transfer, as well as new, innovative and

intelligent services explored during the project. I identified eight non-orthogonal categories for classifying H.323 services: basic, enhanced, centralised, distributed, non-standard, standard, non-signalling and signalling. Further information on service categorization can be found in Chapter 3.

Considering the empirical nature of this research, three services each with their own classification were chosen for further study. Specifically, research into design, development and integration of the service into a live H.323 (version 4) network. Where necessary, the service's interworking with the traditional PSTN was addressed and is also discussed in this thesis. Ultimately the research covers all aspects of service creation in H.323 version 4 networks, including the mechanisms provided by the protocol to support the service, service interworking with the PSTN and the implementation of the service itself.

1.3 Research Questions

The questions that guided the research are:

1. What is an H.323 service?
2. Can H.323 services be logically categorized in terms of their design and functionality?
3. What service architecture and mechanisms does H.323 version 4 provide to support service creation and protocol extension?
4. If possible, how can H.323 voice services accommodate traditional PSTN users?
5. Does H.323 provide service management?
6. How difficult/easy is it to create new H.323 services?

1.4 Document Overview

Chapter 2 This chapter provides background information on the H.323 protocol suite. It describes the protocol, the components that make up an H.323 environment and the mechanisms provided by H.323 version 4 that facilitate standard service creation.

Chapter 3 The focus of this study is H.323 service creation. This chapter introduces telephony services, both traditional telephony services and the newer, innovative H.323 services. The work proceeds to the process of categorising the services based on their particular attributes. The attributes of a service are governed by the network in which the service resides, and thus the categorisation is H.323-specific. The chapter concludes with brief descriptions of the three enhanced services chosen for implementation in this project.

Chapter 4 H.323 is a complex protocol and developing our own software stack would not have been worth while since perfectly useful stacks already existed. However, the commercial stacks available at the time were expensive and more importantly did not lend themselves to modification because of their closed nature. Instead I turned my attention to the OpenH323 project's H.323 protocol stack. OpenH323 is an open-source project that provides tools to developers (both commercial and non-commercial), which are necessary to create H.323 applications. This chapter presents the architecture of the chosen H.323 stack and also introduces essential tools, such as the Abstract Syntax Notation Number One (ASN.1) compiler that was used extensively in this study.

Chapter 5 This chapter describes, in detail, the standard service creation mechanisms provided by H.323 version 4. H.450 is the primary supplementary service-creation mechanism, while H.460 may also be used for service creation but is mainly used for service negotiation among H.323 entities. The factors contributing to the decision to use either H.450 or H.460 or a combination of the two are also discussed. Finally, the service management functionality provided by H.323 version 4 in the form of H.323 Annex K is discussed.

Chapter 6 Non-signalling services do not require modification to the underlying H.323 messages. They are simply H.323 endpoints that have embedded intelligence in the form of

other computer technologies. In the course of the study the embedded technologies that were integrated into H.323 components included Automatic Speech Recognition (ASR), Speech to Text (STT), Internet Message Access Protocol (IMAP) and radio teletype (RTTY). The chapter is example driven and is based on a study of the EmailReader service that was implemented. The implementation of EmailReader and its integration into the H.323 test network form a major part of this chapter.

Chapter 7 Signalling services require the underlying message structures of H.323 to be altered. In this chapter I discuss the reasons why modification of these messages is not a trivial task. This chapter is also example driven and refers in particular to the alarm clock service, called CANS, used as one of the example services. In this regard, the use of the H.323 version 4 service creation mechanisms in the implementation of CANS is considered. Adding information to the underlying H.323 message structure brings up a number of issues when one considers that other H.323 components may need to recognise the presence of such information and ultimately interpret it. Such issues are discussed in this chapter and solutions are proposed.

Chapter 8 This chapter summarizes the experience I gained from working with H.323 version 4 with particular emphasis on the protocol's service-creation mechanisms. I discuss the difficulties encountered during the design and implementation of certain services and whether or not such difficulties have significance to the category that the corresponding service falls into. I conclude with final remarks about the research, its contributions and possible extensions.

Chapter 2

H.323 VERSION-4 BACKGROUND

"When you are a Bear of very Little Brain, and you think of Things,

you sometimes find that a Thing which seemed very Thingish inside you is quite different

when it gets out into the open and has other people looking at it."

A.A. Milne: The House at Pooh Corner

This chapter provides information on the H.323 standard, as background to material presented in following chapters. It explains the basics of H.323, its components and the mechanisms it provides to facilitate service creation.

2.1 Introduction

H.323 is a standard for delivering multimedia over packet-based networks that was developed by the International Telecommunications Union (ITU) in 1996. H.323 defines the terminals, equipment, and services that enable multimedia communication over packet networks, and specifies how terminals and equipment carry real-time voice, video and data. Since the first release of the recommendation (version 1), there have been three new ratifications, versions 2, 3 and 4: This work deals only with the latest, version 4, released in November 2001.

H.323 is not an entire specification on its own, but rather acts as an umbrella for a suite of recommendations defined by the ITU [25]. In other words H.323 relies on supporting protocols to complete its operations; for example, H.323 specifies the use of the H.225 protocol for connection establishment, the Q.931 protocol for call setup, H.450 for supplementary services, RTP for transmission of media streams, etc.

2.2 H.323 Components

Figure 2.1 illustrates the components of a simple, single-domain H.323 environment. These components include the terminal, gatekeeper, gateway and Multipoint Control Unit (MCU). Terminals are mandatory in any H.323 network while the gatekeeper, gateway and MCU are optional.

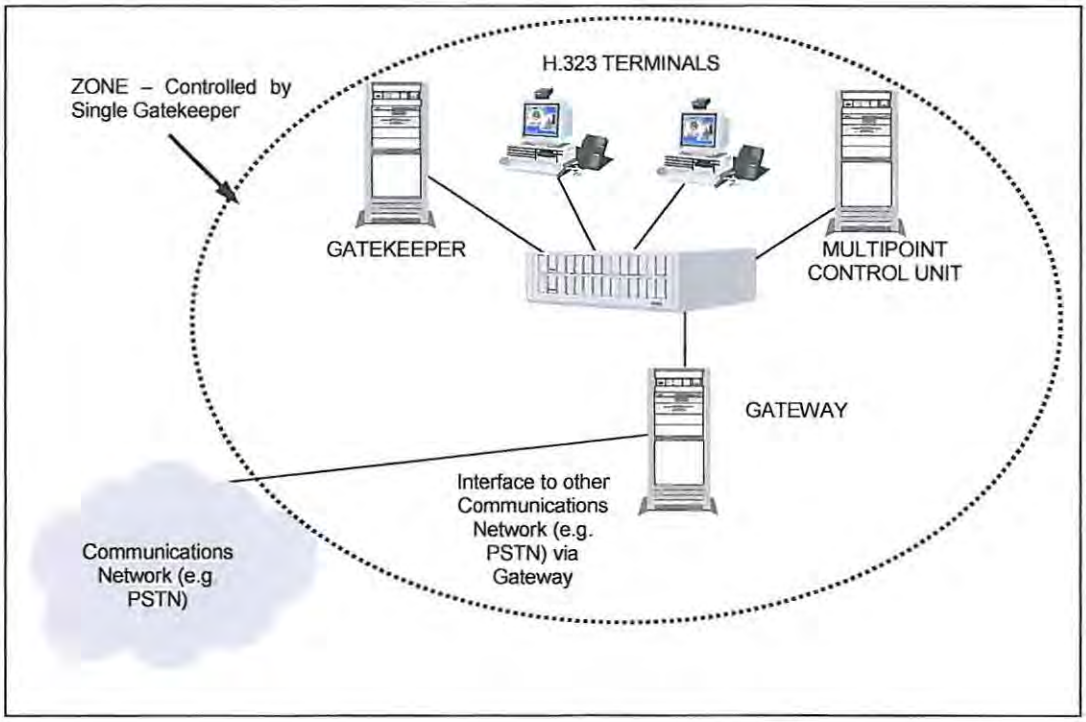

Figure 2.1 The components of a single-domain H.323 Environment.

2.2.1 Terminals

H.323 terminals are endpoints that provide real-time, two-way multimedia communications with other H.323 terminals. The terminal provides the coding and decoding of digitally compressed audio and video signals.

In order for a terminal to be H.323 compliant, it must support, at minimum, voice communications. H.323 terminals can optionally support data and/or video. The H.323 environment deployed for this study is complete and supports audio, video and data conferencing. However, the services explored in this research are specifically voice services. Reasons for this choice include the fact that the PSTN is currently a voice-only network, and that a major part of this study focuses on the interworking of the PSTN and the H.323 network. Omitting video and data from an H.323 service investigation does not limit the study, since services developed using voice, video, data, or any combination of the three, have similar attributes in terms of service categorisation and implementation. The only difference is the

media conveyed in the service and the processing of that media at each terminal involved in the service.

H.323 terminals are not limited to software applications residing on a PC, but can also resemble a standard analogue telephone that one would see connected to the public telephone network. Such phones are known as IP phones. The major difference between the standard telephones we have become accustomed to on the PSTN and the various H.323 terminals is the amount of intelligence they possess. The architecture of the PSTN is such that most of the intelligence exists within the network, making standard telephones (edge devices) basic stimulus-response devices. In H.323 networks, terminals, like IP phones, are intelligent and can be customised to suit many applications and services. H.323 endpoint intelligence is exhaustively explored in this project to provide new and powerful telephony services.

2.2.2 Gatekeepers

H.323 gatekeepers are administration servers that provide call control services to H.323 endpoints. Services that gatekeepers provide include address resolution, admissions control, bandwidth control and zone management.

- Address Resolution – Gatekeepers perform alias address to transport address translation. Endpoints register one or more aliases with a gatekeeper. This means that other endpoints can query the gatekeeper for a specified alias. The gatekeeper performs the address translation, if such an alias exists, and returns it to the requesting endpoint.

- Admission Control – H.323 gatekeepers and endpoints communicate using the Request, Admission, and Status (RAS) channel. Endpoints attempt to register with the gatekeeper. Gatekeepers are configured with a set of rules that govern which endpoints they will allow to register. Depending on these rules gatekeepers will either grant or deny an endpoint's request. Once registered, terminals are required to request permission from the gatekeeper to make H.323 calls. This is an important function, required for billing and pre-paid calling services.

- Bandwidth Control – When initiating a call, a terminal requests a predetermined amount of bandwidth from the gatekeeper. The gatekeeper responds by either granting or denying the request, based on bandwidth allocation limits set by the H.323 network administrator for the gatekeeper's zone. For example, consider a gatekeeper servicing a zone that is configured to allow a maximum of 50 Mbps of H.323 traffic on the network. Let us assume that there are five conferences in progress, each utilising 1 Mbps of bandwidth. This means that the terminals involved in these conferences have collectively requested 50 Mbps of H.323 traffic from the gatekeeper. If a sixth conference is attempted by another H.323 terminal the gatekeeper will refuse the call because there is no bandwidth available until one of the five established conferences is terminated.

- Zone Management – A zone is defined to be a collection of terminals, gateways and MCUs all monitored by a single gatekeeper. A zone is designed by the H.323 network administrator and comprises all the H.323 endpoints that the gatekeeper services. The instructions regarding which endpoints to service are provided by the H.323 network administrator in the form of IP addresses. For multiple addresses, IP address patterns can be used (for example, 146.231.121.* using the IP address/subnet mask pair). The zone is therefore a logical association of H.323 endpoints and may span multiple networks.

2.2.3 Gateways

H.323 gateways enable H.323 endpoints to communicate with non-H.323 endpoints by providing translation between the associated protocols, media and networks. For example an H.323/H.320 gateway is one that provides translation between the H.323 and H.320 protocols at the protocol level, and the IP network and Integrated Services Digital Network (ISDN) at the network layer. (H.320 is the standard for real-time multimedia communication on ISDN networks). The media may also be transcoded depending on the voice/video codec being used on each network.

An H.323 gateway has been used extensively in this project. It is not an H.323/H.320 gateway like the one described above but rather an H.323/ISDN gateway. This gateway bridges the H.323 IP network and the PSTN via the ISDN network. Basically, it allows standard analogue telephones and cellular phones on the public telephone network to interoperate with H.323 network elements, terminals, gatekeepers, MCUs, answering machines, Interactive Voice Response (IVR) systems, etc. Further description of the H.323 environment setup for this project can be found in Chapter 4.

2.2.4 Multipoint Control Unit (MCU)

H.323 MCUs provide support for multipoint conferences between three or more endpoints. An MCU consists of a Multipoint Controller (MC) and any number of optional Multipoint Processors (MP).

The MC is responsible for the call signalling between the MCU and participating endpoints. This standard H.323 signalling (H.225 and H.245 procedures) ensures a common level of communication for the multipoint conference.

The MP is responsible for mixing, switching, transcoding, or any other processing of media streams between connected participants.

2.3 H.323 Protocol Stack

The H.323 protocol stack is based on several protocols, as illustrated in Figure 2.2. The stack supports call control (setup, tear-down, etc.), media packetization and transportation, RAS signalling and all messages in H.323 systems, including supplementary service control messages.

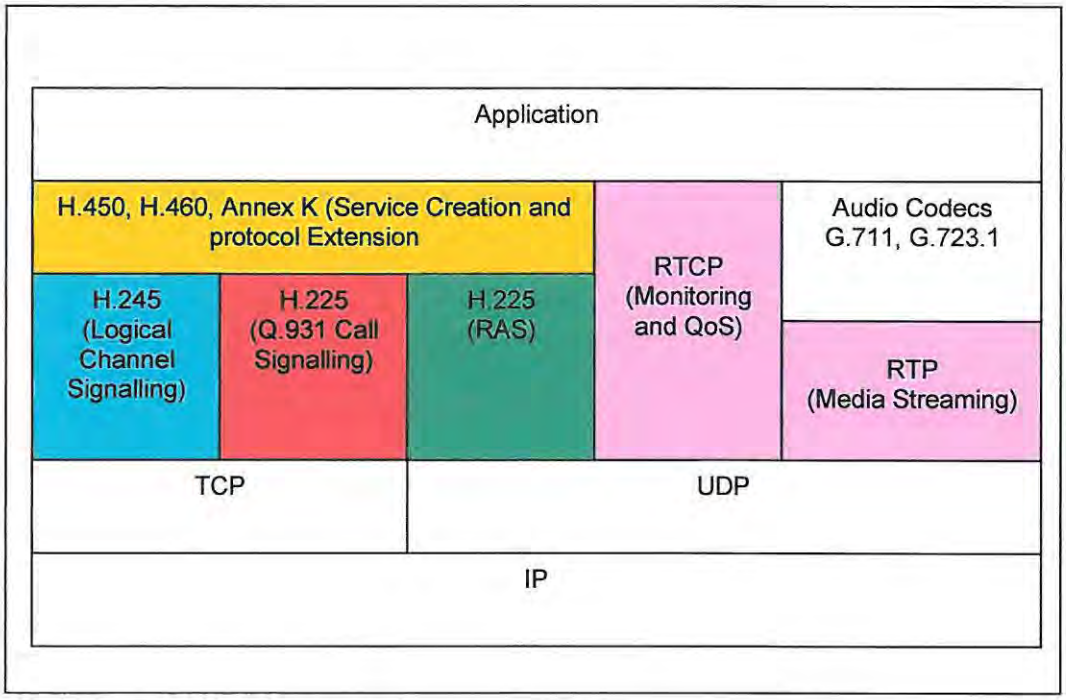

Figure 2.2 H.323 Protocol Stack

The H.323 protocol suite is split into four main areas of control:

- Call Signalling (Red) – Responsible for connection, maintenance and clearing of calls between H.323 endpoints. Call control signalling is addressed by the H.225 protocol, which specifies the messages and operations for H.323 call control and signalling.

- Media Control (Blue) – Provides the reliable H.245 control channel responsible for the control of the media channels used to transmit and receive media. H.245 specifies the messages for the opening and closing of these media channels, and handles other commands, requests and indications.

- Media Transport (Purple) – The protocols (RTP and RTCP) responsible for the packaging and transmission of real time data between H.323 entities. Real-Time Protocol (RTP) defines the end-to-end transport of real-time data. Real-Time Control Protocol (RTCP) is part of the RTP specification and defines the end-to-end monitoring

of data delivery and quality of service (QoS) by providing statistical information about the media streams.

- Registration, Admission, and Status (RAS) Signalling (Green) – Provides communications active between H.323 endpoints (terminals, gateways, and MCUs) and their controlling gatekeeper.

- Service Control and Signalling (Yellow) – Responsible for service signalling and interaction among H.323 network endpoints. The protocols involved in service control are a focus of this study. H.450 is responsible for supplementary service control; while H.460 can also be used for creating services, it is mostly used for its feature negotiation capability. H.450, H.460 as well as other H.323 annexes that address service control, creation and extension in H.323 networks are discussed throughout this thesis.

2.3.1 H.225 Protocol

H.323 call control procedures are based on the protocol H.225, which specifies the use and support of Q.931 signalling messages. H.225 also specifies the use of Q.932 messages for supplementary services. The following Q.931 and Q.932 messages are the most commonly used signalling messages in H.323.

- Setup – Initiated by calling H.323 entity to establish a connection with a remote H.323 entity.

- Call Proceeding – Sent from called entity to calling entity indicating that call establishment procedures are in progress.

- Alerting – Sent from called entity to calling entity indicating that the called entity is ringing.

- Connect – Sent from called entity to calling entity indicating that the called part answered the call.

- Release Complete – Sent by the endpoint initiating the disconnect indicating that the call is to be torn down or disconnected.

- Facility – A Q.932 message used to request or acknowledge supplementary services. This is an important message used in service creation within H.323.

The call control functionality of H.225 is central to this study as it carries the service information between network entities within the H.323 network. The way this service information is formatted and transported over the network is defined in the H.450 and H.460 recommendations. This requires a close interaction between H.225 and the two service control protocols, H.450 and H.460. In addition, service management (H.323 Annex K) information is transported within H.225 call signalling messages.

2.3.2 H.245 Protocol

Once the H.225 connection has been established between two terminals, the H.245 media control and transport channel is opened. H.245 is responsible for the end-to-end control messages between H.323 entities. H.245 procedures establish logical channels for transmission of audio, video, data, and control-channel-data information. One H.245 channel is established for each call with the participating entity. The H.245 control channel is responsible for:

- Capabilities Negotiation – Messages that exchange the capabilities of the corresponding entities (terminals) within a call. These messages identify the terminal's transmit-and-receive capabilities.

- Master-Slave Determination – Procedures used to determine which endpoint is the master and which endpoint is slave for a particular call. This process is used for conflict resolution between endpoints (e.g., when two endpoints request similar actions simultaneously), thus preventing possible deadlock.

- Logical Channel Signalling – Messages responsible for the opening and closing of channels carrying audio, video and data.

2.3.3 RTP/RTCP

The Real-Time Protocol (RTP) is responsible for the transmission of audio and video in H.323 environments. It is an Internet Engineering Task Force (IETF) standard that allows any real-time data to be transported over an IP network. RTP relies on the connectionless transport protocol, UDP. This is because connectionless protocols have much less overhead than their connection-oriented counterparts (TCP). Lost and delayed packets are of no use to the smooth operation of real-time communications. A lost or excessively delayed packet can negatively affect real-time communication. Such packets can usually be dropped if there are relatively few frames. With UDP these packets can be easily dropped, while TCP will attempt to retransmit the packets and ultimately cause more damage to the live real-time stream. Packets on an IP network take many different paths en route to their destination; to ensure that out-of-order media packets are delivered at the proper time and synchronized between various media streams, RTP packets are timestamped so they can be processed at their destination and correctly ordered [32].

Real-Time Control Protocol (RTCP) is a control channel used for monitoring data delivery. It is also responsible for compiling important information and statistics about the media being transported on the corresponding RTP channel [32].

2.4 H.323 Supplementary Service and Service Creation

H.323 version 4 has brought with it many enhancements, especially with areas of focus in new services and service creation mechanisms. These enhancements are brought about via H.450, H.460 and various H.323 annexes.

2.4.1 H.450 Protocol

Traditional telephone services are being developed for H.323 IP telephony networks. These services are defined as supplementary services and are covered by the H.450 recommendation. A good introduction to H.450 can be gained by investigating its architecture, protocol design and service development, all of which have strong advantages over the PSTN's service layer, the Intelligent Network (IN).

The architectural model of H.450 is peer-to-peer, placing intelligence at the edge nodes (terminals) of the H.323 network, as opposed to being located within the network, which is characteristic of the PSTN. This important feature of the H.323 network is exploited in this project. It appears that a wealth of innovative services can be developed with no detrimental impact to the normal operation of the H.323 network.

H.450's protocol design is based on Q-interface Signalling protocol (QSIG), which is the voice communications protocol found in Private voice networks. Both QSIG and H.323 rely on Q.931 for call signalling, and this reduces the complexity involved in inter-working H.323 with the traditional circuit-switched equipment of the PSTN. This architecture can enable a smooth transition from currently installed PBX voice networks to the more intelligent H.323 multimedia networks that combine voice, video and data into a single communications network.

The major driving force behind the exploration of H.323 voice services is the fact that development and integration of new services into the PSTN's service layer, the IN, is complex, proprietary, slow and expensive (Chapter 1). In sharp contrast, H.450 services are implemented as software packages and are deployed in much the same way as any other software package purchased from a store or downloaded off the Web and installed on the edge device or terminal. These services are fully standardized by the ITU and therefore can be implemented by any software vendor and deployed in any H.323 network, interworking with any other vendor's implementation of the same service, a feature the IN lacks. However, some incompatibility of services between terminals is inevitable. This situation can be resolved through the use of service or feature negotiation. Thus, terminals may exchange information identifying the services and features they support and/or require before the services are exeeeuted.

Service development in H.450 is based on a multi-tier approach. Basic services consist of building blocks from which more complex services are developed, while compound services are developed from two or more basic services. Both basic and compound services are used by applications to provide multimedia services to the user. Examples of basic services supported in H.323 include multiple call handling, call transfer, call forwarding, call park and pickup, call waiting, message waiting indication, and n-way conference. Examples of compound services

include consultation transfer and conference out of consultation. Consultation transfer uses call hold, multiple calls, and call transfer. Conference out of consultation uses call hold, multiple calls, and n-way conference.

H.450 is a powerful mechanism for supporting a range of telephony services that may be implemented by different vendors and still be interoperable. To maintain this interoperability, new services, entirely unique and more advanced than traditional telephony services must conform to the message format and signalling rules defined by the H.450 base recommendation (H.450.1). H.450.1 ensures that all services conforming to its base specification have the following attributes:

- The same service implemented by different vendors can interoperate with one another.
- The H.225 base messages, responsible for transporting the service operations and data, are not made more complex than need be (i.e., the standard structure of H.225 messages, defined by the ITU, is not altered in any way)
- Backwards compatibility between H.323 versions is maintained. For example, H.323 version 4 supports the Call Intrusion service (H.450.11) that allows a calling terminal to interrupt an existing call between two other terminals. Previous H.323 versions do not support H.450.11 and various actions may be taken when a version 4 terminal tries to intrude on a call between two version 2 or version 3 terminals.

The third attribute above has become a controversial issue in terms of service compatibility. H.450 ensures that a safe recovery occurs when one terminal initiates a service with another terminal that does not support the service. A safe recovery implies that the H.323 call is maintained even though the service invocation fails. However, there are methods specified in H.450 that force calls to be dropped when an unrecognised service is initiated. In fact, there are several actions a terminal can take when it receives an invocation of a service it does not support. These actions are discussed in Chapters 5 and 7.

In some cases it may be imperative that a particular service is supported by two terminals before they establish an H.323 call with each other. Such a scenario can be resolved by using the feature negotiation procedures of H.460, which are described below.

2.4.2 H.460 Protocol

H.460, also known as the *Generic Extensibility Framework* (GEF), allows new features to be readily added to H.323 without affecting the underlying H.225 base specification.

GEF's most valuable function is its feature negotiation which resolves the problem involving service capability exchange (see Section 2.4.1). Feature negotiation involves the communication of feature sets among all entities involved in an H.323 call. The capability sets are as follows:

- Desired Features – Represents all features a specific entity would like to use. This means the entity can still function if the remote entity does not support the features in this set.
- Needed Features – represents all features that a specific entity needs (i.e., features that the entity can not function without).
- Supported Features – Represents all features that an entity is capable of.

These feature sets are communicated over the H.225 RAS channel, when gatekeepers are involved in call setup and over the H.225 call signalling channel in direct point-to-point calls with no gatekeeper. When routing call signalling, entities in the middle of the call signalling path, like gatekeepers and gateways, may add to or subtract from the specified feature sets if they can/can't perform the feature on behalf of one of the endpoints in the call. This means that, where necessary, more intelligence may be built into the network without having to add the intelligence to the endpoints in a call. For example, if a new service is to be integrated into all H.323 terminals within an H.323 zone, instead of integrating it into every single terminal in the network (perhaps thousands), it could be integrated into the gatekeeper that services the terminals. In such a case the gatekeeper performs the service on behalf of all the H.323 terminals it services. When a call is initiated from one of these terminals, the gatekeeper adds the

information to the supported feature set of the H.225 message before routing it to the remote H.323 terminal.

The mechanism of feature negotiation is based on the request response paradigm. To illustrate the process I present the most basic example, a standard point-to-point call between two H.323 terminals, with no intervening network entities. Figure 2.3 shows an example of an H.460 or GEF call signalling negotiation for a fictitious service, called X.

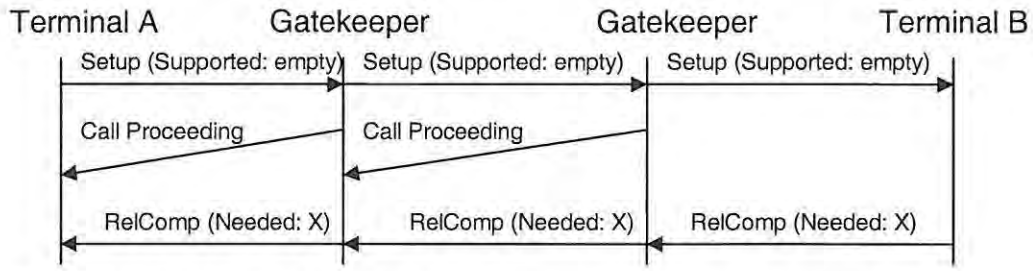

Figure 2.3 H.460 Capability Negotiations

In this example the initiating endpoint is a standard H.323 terminal with no additional H.460 features or services. The example shows an attempt to set up a call to the remote terminal (B). Terminal B, however, requires service X for a successful call, but terminal A's setup message has no indication that it supports service X (i.e., A's supported feature set is empty as shown). Terminal B is thus responsible for clearing the call, which it does by sending a H.225 release complete message (RelComp message) to terminal A. Included in the RelComp message is the description of the required service, service X. This description lets terminal A know the service it requires to engage in a call with terminal B.

GEF was released with H.323 version 4. This means that prior versions do not support H.460. In the above example, terminal A did support GEF, but it did not support any of the services defined by GEF. Hence, terminal A could be an H.323 version 1, 2 or 3 terminal. In other words, a terminal that does not support GEF is masked as a terminal that does support GEF but with no GEF features or services.

In addition to feature negotiation, GEF provides a low-overhead means of adding functionality to H.323 without adding to the H.225 base specification. Thus it allows application-specific extensions to be made to the H.323 suite of recommendations without burdening all H.323 implementations with all specified extensions. This function of GEF combats one of the major criticisms of H.323; as it matures the number of parameters that exist in the base specification (ASN.1) significantly increase making the protocol bloated and more complex. Thus GEF prevents continued and unbounded growth of the ASN.1 modules that define the H.225.0 protocol.

2.4.3 Annex K

New, standardised functionality added to H.323 between the release of major versions is released by the ITU as annexes. These annexes address various issues related to H.323, from call signalling enhancements to advanced service creation and interaction. Annex K, released in 2000 specifies a functionality that falls under the scope of this project, namely service creation. This functionality is responsible for one of the major innovations of H.323 version 4, which defines how HTTP can be used for the transport of service control information with both call and non-call-related H.323 signalling. Specifically, H.323 Annex K provides a means of service management in H.323 version 4 networks.

Annex K enables H.323 users to access, personalise and control many aspects of the services in H.323 environments. It has opened the door for web-based services that can be directly incorporated into a range of H.323 applications. Examples of such services are:

- Click to dial
- Point and click for multiparty conference setup
- Interactive web/voice response
- Web instant messaging and presence
- Follow-me services

2.5 Summary

H.323 specifies the use of four network components that may comprise an H.323 environment or network. These components are terminals, gatekeepers, gateways and MCUs. In this chapter the functionality that these components provide to an H.323 network has been introduced. The description of an actual H.323 environment, as deployed for this study, is provided in Chapter 4. H.323 is a relatively complicated protocol that specifies the use of other protocols to carry out its operations. This portion of the work has introduced and briefly discussed the various protocols that are vital for H.323 operation, including H.225, H.245, RTP, and RTCP. In addition, the mechanisms that facilitate service creation in H.323 version 4 networks were also introduced. These mechanisms are specified in the ITU recommendations H.450 and H.460. Finally, I introduced H.323 Annex K, an H.323 version 4 service management feature.

CHAPTER 3

H.323 SERVICES

"In theory, there is no difference between theory and practice. But, in practice, there is."

Jan L.A van de Sneupscheut

This chapter introduces and discusses H.323 services. These services are categorised in terms of their design, functionality and implementation. I also discuss the architecture of H.323 networks and the services therein, with emphasis on where the services are deployed and how they can be made available to PSTN users.

3.1 Introduction

The initial benefits of carrying voice over the Internet (IP), using protocols like H.323, concerned low-cost/low-quality toll bypass scenarios. This situation is, however, changing as new and innovative services are introduced and IP quality of service (QoS) is addressed. Communication networks, such as H.323 are not limited to new services but must also support traditional telephony services like call forward and call hold. Support for traditional services is vital for successfully interworking with conventional private telephony networks - and ultimately for accomplishing a smoother inevitable migration from the switched telecommunication networks to packet based multimedia telecommunications networks.

3.2 Service Categorization

To develop, design and integrate a new service into an H.323 network, it is necessary to consider all of its properties. Such properties hinge on the service's functionality and its interaction, or lack thereof, with other network entities. A number of H.323 service categories were identified by considering their properties:

- Basic and Enhanced services
- Signalling and Non-Signalling services
- Standard and Non-Standard (Proprietary) services
- Centralised and Distributed services

In many cases the implementation of an H.323 service influences the properties it possesses and consequently affects the category in which it can be classified. This means that the same service may fall into different categories depending on its implementation. There is also overlap among the different categories, since a specific service can fall into more than one category simultaneously (i.e., H.323 services are non-orthogonal). As an example, there is strong overlap

between signalling services and distributed or centralised services. Figure 3.1 illustrates the H.323 service categories identified in this work, with all possible category overlaps.

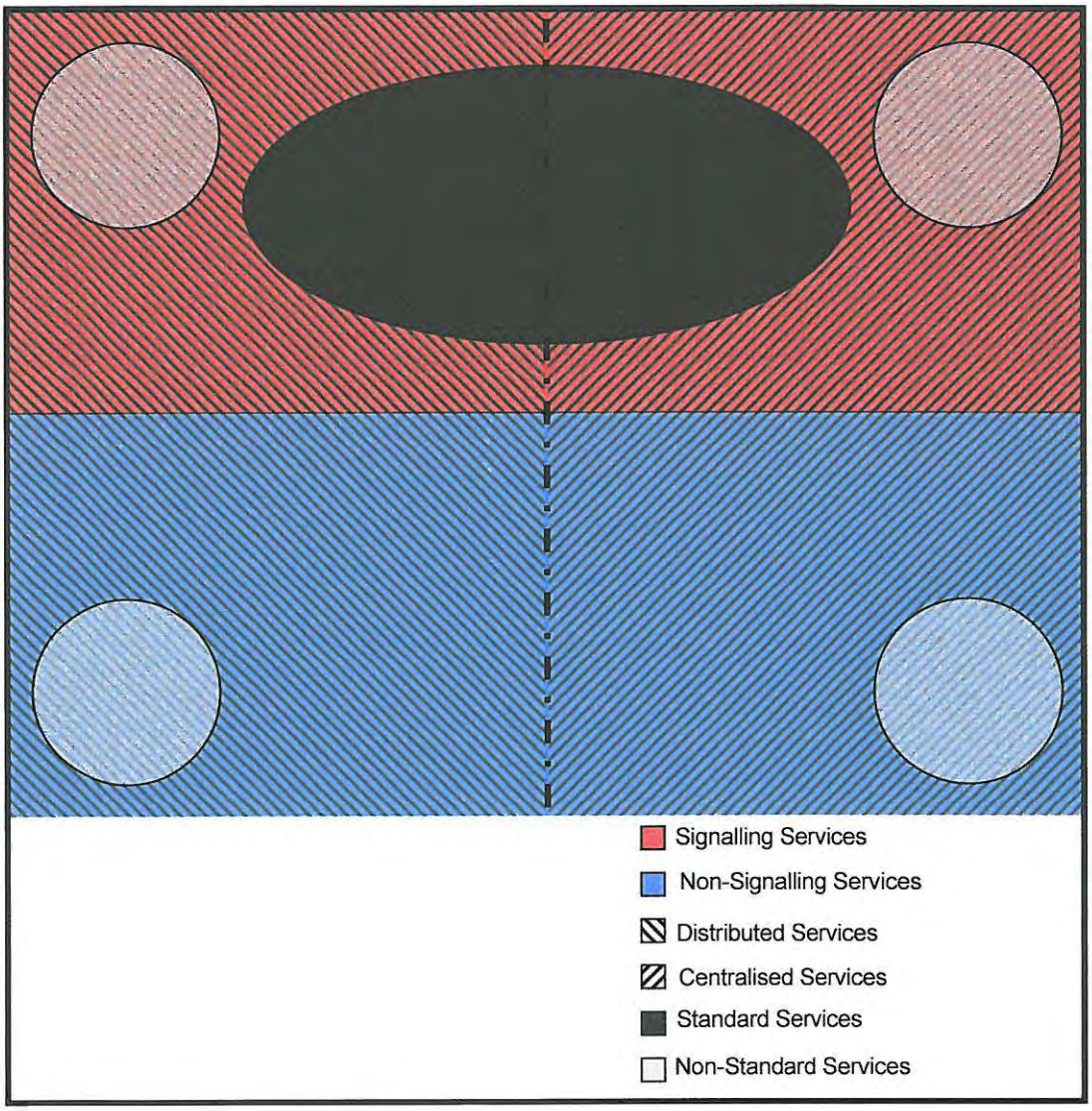

Figure 3.1 H.323 Service Categories

3.2.1 Basic vs. Enhanced Services

The first categorization distinguishes between basic and enhanced services. Basic services are based on the set of services that most of us have become accustomed to with the PSTN. These services, often referred to as traditional telephony services, include the services provided by the

IN portion of the PSTN (e.g., like call hold, call transfer and call waiting). They are termed *basic services* because of their lack of intelligence in terms of H.323 service capabilities. Also, they are related only to call signalling and do not include the set of services that enhance the use of multimedia communication on computer networks. As mentioned earlier, the IN of the PSTN limits the functionality of the value-added services it supports because of the simple, non-intelligent network on which it resides, the PSTN. The IP network is considerably more intelligent and the traditional services ported to this environment are indeed basic in comparison to the enhanced services that draw on the data network's intelligence and multimedia capabilities.

Enhanced services include the three services that were developed during the course of this project: EmailReader (Chapter 6), CANS (Chapter 7), and Telgo323 (Appendix C). Such services draw on the packet-based network's flexibility and scalability. For example, conference calls can be easily implemented by deploying modern-day computers that are capable of accepting multiple H.323 connections and so use their processing power to process, mix and route audio data among the connected participants. Directory services can be made available by drawing on existing data storage protocols responsible for storing large amounts of processed, structured information. Such data storage protocols include the Lightweight Directory Access Protocol (LDAP) already used by many H.323 directory services.

Service Properties	Basic Service	Enhanced Service
Requires intelligent underlying network	NO	YES
Similarity to Traditional PSTN Services	YES	NO
Complexity of implementation in H.323	HIGH	HIGH

Table 3.1 Basic vs. Enhanced Services

Table 3.1 illustrates the properties associated with the basic and enhanced service categories.

3.2.2 Signalling vs. Non-Signalling Services

In many cases an H.323 service may need to interact with other H.323 entities that participate in the delivery of the service at a call signalling level. To interact in this manner, service information is transported within the H.225 call signalling messages (e.g., Setup, Release Complete, Facility) between H.323 entities. Services that operate in such a manner can be defined as signalling services.

Signalling services are complex services to implement because:

- They require information to be transported within H.225 messages, and
- All H.225 messages are standardised by the ITU and specified using ASN.1. These specifications describe each H.323 message's structure and format for transmission across the network. This means that if we add information to the messages in an ad hoc manner, we compromise the standard nature of the H.323 protocol.

In order to develop signalling services, H.225 message structures must be examined to locate areas in which signalling service information can be placed. Placeholders do exist within the H.225 message structures and can be used to carry the necessary service information. The use of these placeholders is defined by the protocols, H.450 and H.460. More information on signalling service creation using H.450 and H.460 can be found in Chapter 5.

Non-signalling services use basic H.323 procedures to carry out their execution. No additions to the standard H.323 messages are required which means that non-signalling services are easier to implement, even with little knowledge of H.450, H.460 or the message structures of H.225.

Non-signalling services are usually implemented as callable endpoints with additional non-H.323 functionality. A standard H.323 terminal is simply a communications device residing on an intelligent computer network that supports many other applications, like email and speech recognition. Basic H.323 terminals can be made more innovative by integrating these other technologies into them. In this work I have termed the integration of other technologies into H.323 terminals as *embedded intelligence*. H.323 embedded intelligence makes it possible to

create a range of new H.323 services that combine state-of-the-art computer technologies with the multimedia communications of H.323.

Service Properties	Signalling Service	Non-Signalling Service
Complexity of implementation and integration into H.323 networks	HIGH	MEDIUM TO LOW
Governed by H.323 standards	YES	NO
Amount of H.323 knowledge required for implementation	HIGH	MEDIUM
Require service control within H.225 messages	YES	NO

Table 3.2 Signalling vs. Non-Signalling Services

Table 3.2 illustrates the properties associated with the signalling and non-signalling service categories.

3.2.3 Standard vs. Non-Standard (Proprietary) Services

H.323 is standard for multimedia communication on packet-based networks. The advantage of adopting standards-based protocols is that vendors' products and applications can interoperate with each other. H.450 and H.460 (see Chapter 2) do this for standard services. For example, H.450.1 defines the generic use of the Supplementary Service (SS) framework in H.323 while, the standardised services H.450.2 and H.450.3 define the call transfer and call hold services, implemented using the SS framework. Standard services are a subset of signalling services (see Figure 3.1) because they rely on the standard H.323 service frameworks, H.450 and H.460, for their operation. Both H.450 and H.460 interact with the H.225 call signalling layer of H.323; therefore I have categorised all services that use H.450 or H.460 as signalling services.

Non-Standard, or Proprietary, services are not standardised by the ITU. They are often implemented in such a way that they become vendor-specific, meaning that they will not interoperate with H.323 components or networks from other vendors. If these services are implemented using H.450 and/or H.460, they remain proprietary (not an officially standardised service) but meanwhile adhere to the rules specified by the ITU (in the form of H.450 and H.460); H.450 and H.460 both aid service interoperability and negotiation with H.323 entities from other vendors. Services implemented in this manner are classified as signalling services because of the use of H.450 and/or H.460 in their implementation.

The previous paragraph notes that the same service can be categorised differently depending on its implementation. A service can be implemented using standard H.450 and/or H.460 mechanisms or it can be implemented in an ad hoc, non-standard manner, for example. A service implemented in either of these two ways will be either a standard or non-standard (proprietary) service, respectively. This means that the inherent properties or nature of the services do not exclusively influence their categorisation as standard or proprietary. The possibility of implementing the service using H.450 and/or H.460 mechanisms also influences their respective categorisation.

Table 3.3 on the following page illustrates the properties associated with the standard and non-standard service categories.

Service Properties	Standard Service	Non-Standard Service
Standardised by ITU	YES	NO
Vendor specific	NO	YES
Interoperable with other vendors' implementations	ALWAYS	YES/NO (depends on implementation)
Is a subset of signalling services	YES	NO
Degree of complexity of implementation and integration into H.323	HIGH	MEDIUM

Table 3.3 Standard vs. Proprietary Services

3.2.4 Distributed vs. Centralised Services

In one sense, the term distributed can be used to describe the division of (a whole or collective body) into parts having distinct characters or functions. The term centralised may be used to describe the process of concentrating parts into a single centre. Both terms are used often in various areas of computer science, including database architecture, software development, computer networks, and parallel computing. From a broad perspective, the words distributed and centralised are used to describe the location of processing power, intelligence, data, etc.

My classification of the distributed or centralised nature of a service depends on the physical location of the service's logic (i.e., the intelligence responsible for the entire service's execution). Distributed services have all service logic located in H.323 terminals and do not require the assistance of other H.323 network entities to perform their operation (i.e., no network

intelligence is required to be spread among other H.323 network entities). The reason for their distributed nature is that the service intelligence is located at the edge or in terminal network devices and not within the central network entities (e.g., gatekeepers).

Non-signalling services are almost always classified as being distributed because of their inability to interact with other H.323 network entities using the call control layer. Exceptions to this general rule are brought about by implementers who embed service information in unusual areas of the H.225 messages, enabling H.323 non-signalling services to use H.225 in a proprietary manner.

In some cases a service's logic may be located in the network in the form of gatekeepers, specialised proxies, directory servers, and other H.323 network elements. Some of the service logic may be located within the edge devices, but still not capable of completing the service execution without interacting with logic located elsewhere in the network. Centralised services are defined as ones that are not capable of focusing the service logic within a single H.323 terminal.

Service Properties	**Distributed**	**Centralised**
All Service logic located in H.323 Terminal	YES	NO
Service information transported within H.225 (signalling services)	NOT NORMALLY	MOSTLY
Complexity of implementation and integration into H.323	MEDIUM	HIGH

Table 3.4 Distributed vs. Centralised Services

Table 3.4 illustrates the properties associated with the distributed and centralised service categories.

3.3 Example Categorisations

To illustrate service categorisation, I have chosen four services to introduce, describe and classify.

3.3.1 Example Service 1 – Call Diversion

Call Diversion is a service that is initiated during call establishment and provides the diversion of an incoming call from one endpoint to another. Various conditions exist that may result in a redirected call, for example when the initial destination endpoint is busy or does not answer.

In H.323 networks, *Call Diversion* is supported by the ITU standard H.450.3, or *Call Diversion Supplementary Service*. It is nowadays a common service in traditional telephony networks and therefore it can be classified as a basic H.323 service. In section 3.2.3 I concluded that standard services are a subset of signalling services which means that *Call Diversion* is a signalling service by default. Hence, H.323 *Call Diversion* is categorised as a basic, standard, signalling service. However, is the service distributed or centralised? An answer requires a detailed investigation of its operation.

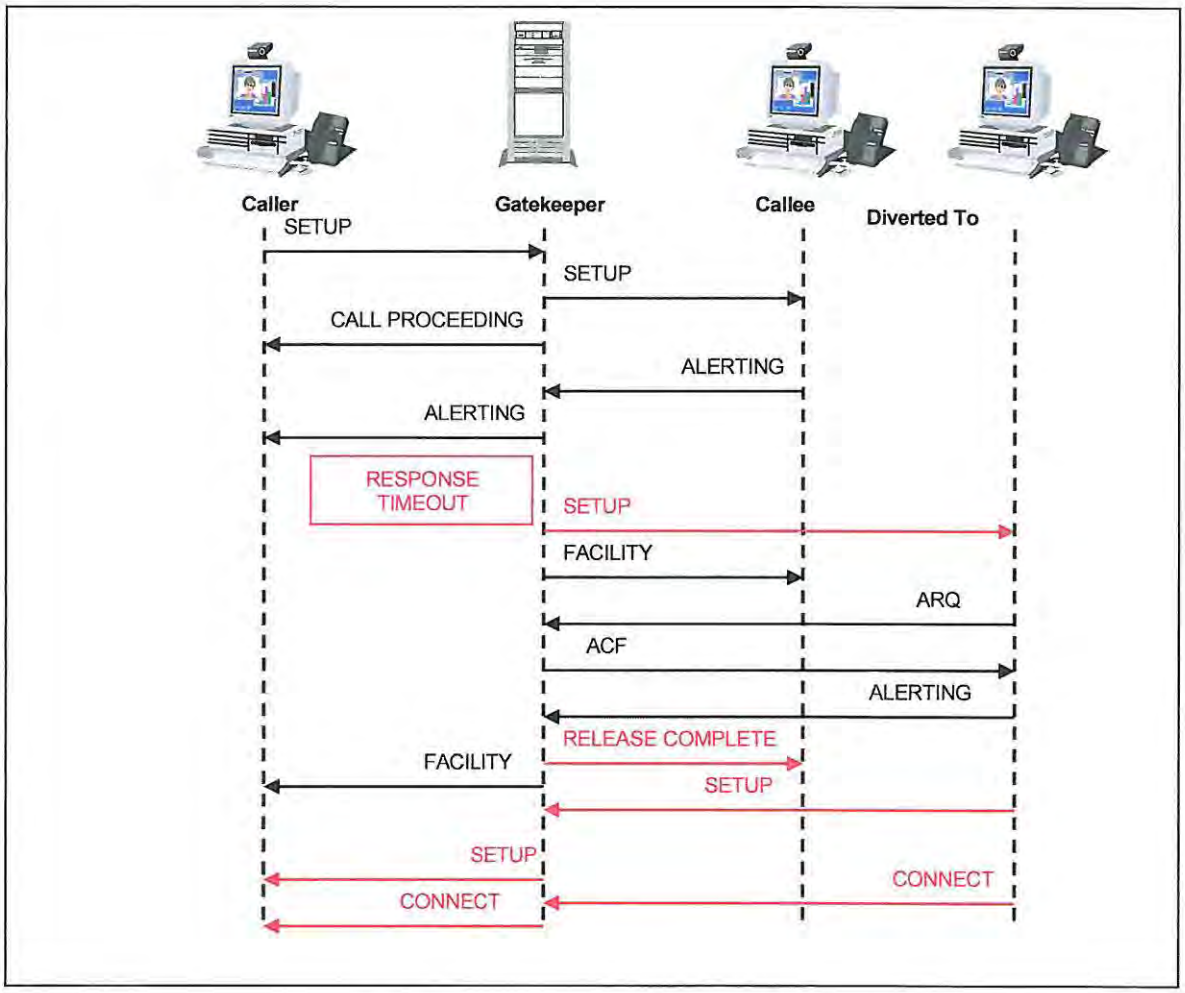

Figure 3.2 Call Diversion Call Scenario

Figure 3.2 illustrates the signalling operations for the H.323 *Call Diversion when-no-reply* service as defined by H.450.3. The call model used in this scenario is gatekeeper routed with all endpoints registered with a gatekeeper. In such scenarios all call signalling messages are routed via the gatekeeper. The interaction of the gatekeeper in the gatekeeper-routed call model does not necessarily make H.323 *Call Diversion* a centralised service. The key is to consider the operation of the service only and not the call signalling messages used to carry the service information. For simplicity, the key messages involved in this classification are highlighted in red in Figure 3.2.

When the response timeout of the initial call has been reached (the caller has waited for the maximum amount of time for the called terminal to answer), the gatekeeper acts as a proxy, attempting to connect the call to an alternate, pre-defined endpoint (the diverted-to endpoint). If the directed-to endpoint accepts the call, the gatekeeper connects the new call to the callee and the call can be completed successfully.

Thus, the *Call Diversion when-no-reply* service relies on the gatekeeper to act as a proxy and redirect the failed call to another endpoint (the address of this alternate endpoint is known by the gatekeeper only). For this reason, I have classified H.323 *Call Diversion* as a centralised service.

3.3.2 Example Service 2 – Call Hold

Call Hold is another common service found in traditional telephony networks and I therefore classify it as a basic H.323 service. *Call Hold* is also standardised by the ITU, as H.450.4, or *Call Hold Supplementary Service*, and is therefore also a signalling service. However, in comparison to H.450.3 *Call Diversion*, H.450.4 *Call Hold* is a distributed service. This is because all service logic pertaining to *Call Hold* in H.323 is located in the user's H.323 terminal and no network logic is required for its operation. Users of H.323 terminals indicate their desire to hold a call, resulting in the closure of the logical channels responsible for carrying audio to and from the remote H.323 terminals. In this state we refer to the call as being *held*. Users can then indicate their desire to restore the call (unhold), resulting in the logical channels being reopened and the call continuing as usual.

3.3.3 Example Service 3 – EmailReader

EmailReader was the first enhanced service to be implemented and explored in the test network. It enables a PSTN user to dial in via a regular touch-tone phone, enter in their username and password, retrieve their email messages from an Internet Mail Access Protocol (IMAP) server and have these messages read to them over the phone using a text-to-speech (TTS) speech synthesis system.

EmailReader was uniquely developed for this project and is not standardised by the ITU; therefore it is a proprietary service. It is implemented as a callable H.323 terminal that has

various embedded computer technologies like TTS and IMAP. This and the fact that EmailReader does not require the aid of any other H.323 network entity make it a distributed service. All interaction with the service is conducted using out-of-band DTMF tones and no additional H.323 signalling is required for its operation. EmailReader is therefore a non-signalling, enhanced, proprietary, and distributed service.

3.3.4 Example Service 4 – Customisable Alarm Notification Service (CANS)

CANS is a complex service that was also developed and deployed in the test network. It enables users to establish H.323 calls with a CANS server. Once in a call a user can supply a date and time to be called back and notified of an event they wish to be reminded of. To add context to their reminders, the system offers users the option of leaving a voice or text message that can be returned upon successful establishment of their call-back notification.

Similarly to EmailReader, CANS was implemented for this project and is not standardised by the ITU. It is a proprietary service, and in this form, as an enhanced service, is not a service commonly found in traditional telephony networks. CANS' implementation is based on a callable H.323 endpoint and does not require the intervention of other network entities for its operation at a service layer (distributed service). Finally, CANS is implemented using H.450 and H.460 standard service creation mechanisms which make it a signalling service because of its requirement for service information to be transported within H.225 call signalling messages. CANS is therefore classified as a non-signalling, enhanced, proprietary distributed service.

3.4 Availability of H.323 Services to PSTN Terminals

The PSTN will continue to be used for many years because of the capital invested into it, its large installed base and reliability. In order to leverage past investment in PSTN, service providers will need to provide seamless services between PSTN and IP [46]. H.323 version 4 facilitates interoperability with such legacy domains.

In terms of the research, Figure 3.3 illustrates the H.323 test environment linked to the PSTN via the H.323/ISDN gateway.

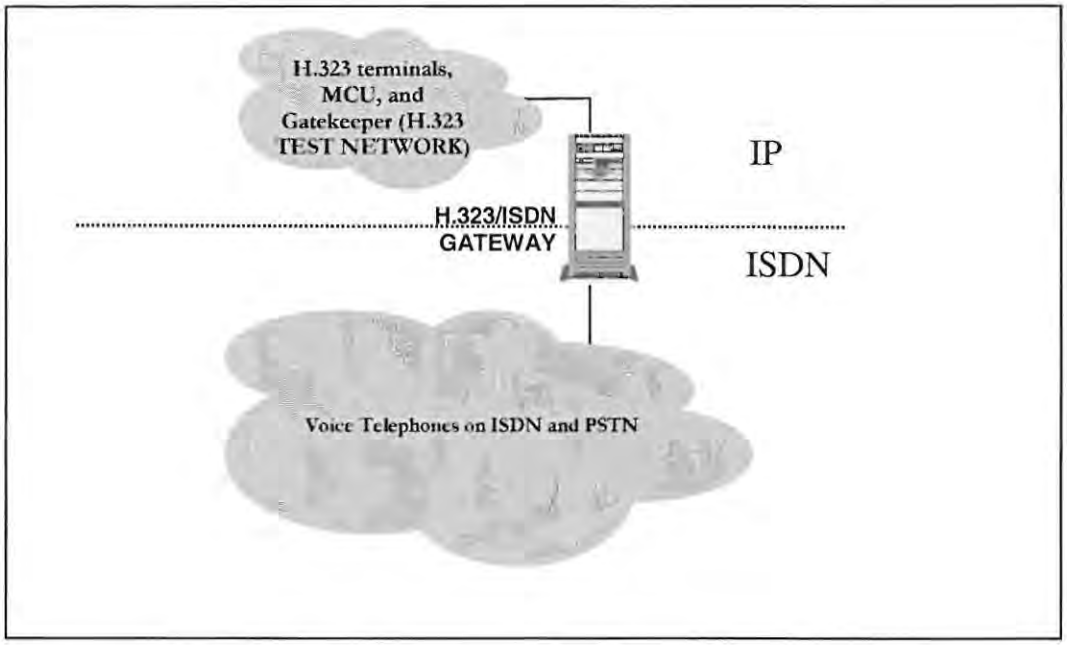

Figure 3.3 H.323-PSTN Interworking Test Network

H.323-PSTN service interworking at a basic service level is addressed by various standards. These standards specify the interoperation between H.450, the service control protocol in H.323 and QSIG, the service control protocol in enterprise circuit switched networks. However, service interworking between H.323 and the PSTN at an enhanced service level must be considered. For example how may a standard PSTN telephone interact with the EmailReader and alarm notification system services introduced in the previous section?

The H.323/ISDN gateway (Figure 3.3) is responsible for bridging call signalling and service control operations between the IP and PSTN networks. In essence, this gateway acts as a mediator between traditional telephones and H.323 terminals. This enables PSTN users to engage in calls with standard H.323 terminals (i.e., standard voice calls between a party on the PSTN and a party on the H.323 network). PSTN users wishing to utilise H.323 services have to do so using DTMF tones. This is because PSTN telephones are relatively 'dumb' network devices, capable of only interacting with core network entities using Dual Tone Multi-Frequency (DTMF) tones.

DTMF tones on the PSTN are transported from one party to another within an actual audio stream. When PSTN devices engage in calls with H.323 terminals via a gateway such as the H.323/ISDN gateway, the tones must be forwarded to the H.323 terminal for processing. Transport of the DTMF tones on H.323 networks can be achieved in a number of ways.

The first method of transport is to package the DTMF tones within the RTP stream. In this case the tones are never extracted from the PSTN audio stream, but are rather packaged with the rest of the audio for transmission to the remote H.323 terminal. This method of DTMF transport is referred to as in-band transmission (tones transported within the audio stream). This leaves the responsibility of extracting the tones from the audio stream up to the H.323 terminal receiving them. In-band DTMF transport is not feasible within H.323 networks because of the various audio compression codecs used to compress the raw audio for transmission on the IP packet network. Various compression codecs lose a certain amount of audio information at the expense of achieving higher compression rates. DTMF tones are usually transported accurately when using high bitrate voice codecs such as G.711 (64 kbps), whereas low bitrate codecs such as G.723.1 (6-8 kbps) are highly optimized for voice patterns, and tend to distort DTMF tones. These distorted DTMF tones may not be correctly recognised by the receiving terminals. For further information on why this occurs refer to [1, 6].

The solution to this problem with recognition is to remove the DTMF tones from the audio stream at the gateway and forward them to the H.323 terminal separately. This method of DTMF transport is called out-of-band DTMF transmission. Tones can be sent to the H.323 terminal in an ad hoc manner by including them in various unused fields of the H.323 signalling messages. But this is not an acceptable solution since it degrades the standard nature of H.323 and its interoperability features. However, two standard methods of carrying out-of-band DTMF in H.323 do exist as follows:

- Use of RFC 2833;

- Removal of DTMF information from the audio stream at the gateway. The data equivalent of these tones can be forwarded to the H.323 terminal via the H.225 call signalling channel.

RFC 2833 specifies the transport of DTMF information, as well as other signals and telephony events within RTP on packet-based networks [31]. These signals are not transported in-band with the audio stream but separately in the RTP payload (i.e., they are not part of the audio stream). In this way the tones are not affected by audio codecs and they can be correctly recognised at the receiving H.323 terminal.

The second solution adopts a different approach to RFC 2833; in this case, DTMF information is removed and decoded from the audio stream at the gateway. The data equivalent of the tones is encoded within standard H.225 signalling messages rather than being placed in RTP payloads. The most commonly used H.225 message for these tones is the FACILITY message because it may be sent at any stage during a call, without affecting call state (e.g., a Setup message cannot be sent with DTMF information once in a call).

H.323 recommends the use of RFC 2833 for out-of-band DTMF transmission, but the H.323 library used for this project did not support RFC 2833. The possibility of adding RFC 2833 to the library was investigated but the actual implementation was deemed not particularly significant to the study so I adopted the second approach in the services that I implemented.

3.5 Summary

This chapter of the research describes eight service categories for classifying H.323 services: basic, enhanced, standard, non-standard, signalling, non-signalling, distributed and centralised. I have shown the differences between the service categories and how various H.323 services may be classified by using four example services. Most services fall into more than one category depending on their attributes, for example, Call Divert is classified as a basic, standard, signalling, and centralised service. The manner in which a service is implemented also influences its categorisation. The exception is standard services (ITU standardised services, such

as H.450.3) because their operation is rigorously defined and may not be altered. Finally, I addressed how H.323 services can be made available to the PSTN, using QSIG for basic service interoperation and DTMF tones for enhanced service interoperation.

CHAPTER 4

TOOLS AND COMPETENCIES

"Do not wait; the time will never be 'just right'. Start where you stand, and work with whatever tools you may have at your command, and better tools will be found as you go along."

Napoleon Hill

This chapter introduces and describes the various tools and competencies required to carry out a study of service creation in H.323 version 4 networks. The tools introduced include a description of the H.323 version 4 network that was deployed for development and study, the H.323 protocol library used to develop new H.323 applications with enhanced services, and finally the Abstract Syntax Notation One (ASN.1) compiler used to create new H.450 services. A basic knowledge of ASN.1 is required for this portion of the work as well as subsequent chapters. A brief overview of the subject is provided in Appendix A.

4.1 Introduction

A study of H.323 service creation requires various tools and competencies. In this work the necessary tools included a complete H.323 version-4 compliant network with open-source components that could be easily modified, and a stable H.323 version-4 protocol library with which to create new services and then integrate them into new or existing H.323 endpoints. A range of competencies related to computer programming were also required. The most important and most relevant of these to H.323 systems is Abstract Syntax Notation One (ASN.1).

The H.323 library used in this project is open source, as developed by the OpenH323 project [39]. It is implemented using PWLib, an application library which enables the H.323 library to run transparently on both Linux and Windows operating systems. The H.323 version-4 network deployed for this study is comprised of many components that were written using the OpenH323 library. This gave substantial flexibility over the implementation of the H.323 environment due to the open-source nature of the library and the components developed using it. To create new services in H.323 networks, especially signalling services that use H.450, H.460, and Annex K, a good understanding of the ASN.1 is required. ASN.1 is an abstract syntax that requires a compiler to translate it into a specific programming language (concrete syntax) equivalent. Further information regarding ASN.1 can be obtained in Appendix A; two academic sources are John Larmouth's *ASN.1 Complete* [36] and Olivier Dubuisson's *ASN.1 Communication between Heterogeneous Systems* [8]. This chapter also introduces the ASN.1 compiler used to create the new H.450 services deployed during this study.

4.2 Test Network

Figure 4.1 illustrates the H.323 test network deployed for this research.

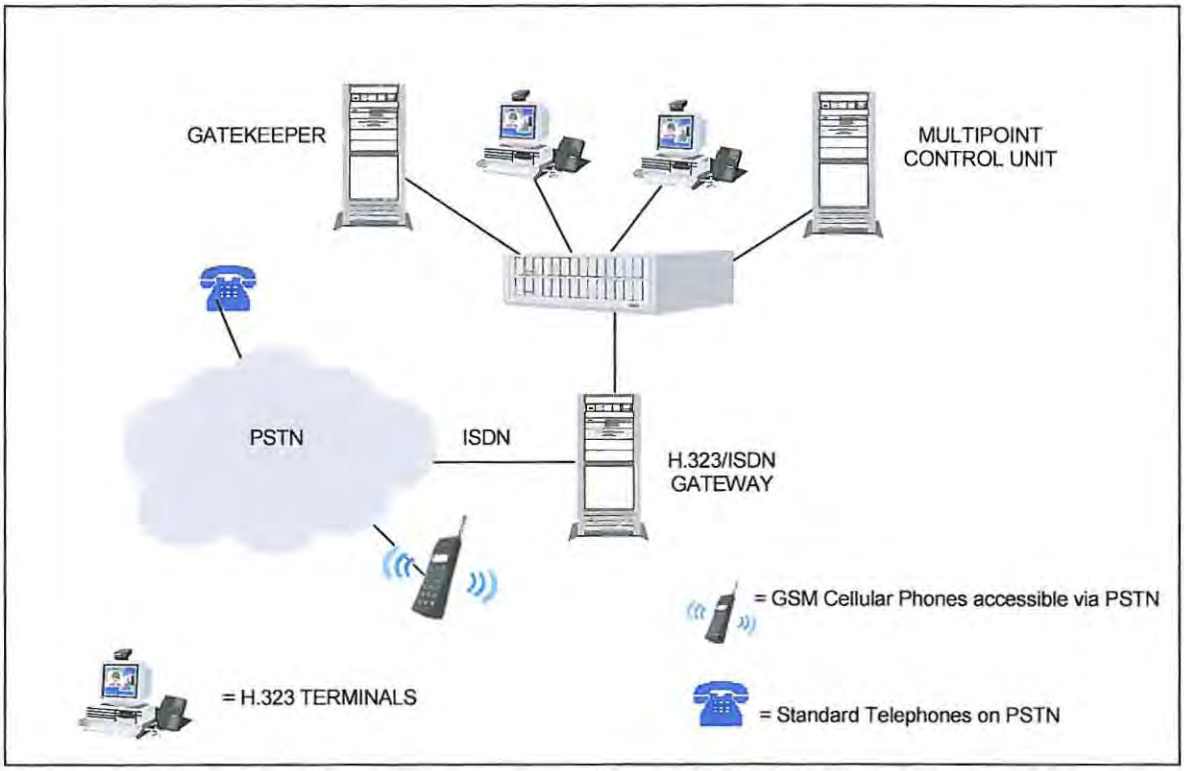

Figure 4.1 H.323 Test Network

4.2.1 H.323 Terminals

A number of H.323 terminals were deployed for users of our test environment. These terminals could be implemented using the OpenH323 library and therefore were based on the PWLib application library introduced earlier. Being based on PWLib meant the terminals could be deployed on both Linux and Windows machines around the campus.

Linux is an open-source operating system that has many advantages in research environments, two of which are mentioned here:

- Open Source software implies Open Standards; thus, it is easy to adapt software to work closely with other Open Source software and even closed protocols and proprietary applications. This solves vendor lock-in situations wherein one is tied to the products of only one vendor if you choose one's products [38].

- Although open source doesn't necessarily mean without charges, most Open Source software is freely available and doesn't demand a license fee per user/year. This allows application developers to cut costs and create more secure and adapted solutions than is possible in commercial consultancy firms [38].

Most of the services that were explored and implemented during this study have been developed on the Linux platform. The reasons for this are based on the personal opinion that Linux is a more powerful development programming system than most other operating systems. In addition, many of the computer technologies (like IMAP, TTS and STT) that were integrated into the project's H.323 services exist within the Linux open-source domain. This made it easier to include Linux-based source-code libraries for these technologies within the H.323 services that were developed.

Rhodes University's Computer Science Department has a number of PocketPC users; to cater for them I provided H.323 terminal software for the PocketPC. This extended the user base for the H.323 environment and the newly created services therein. The PocketPC H.323 software that was chosen is called *PocketBone*, which is implemented using the OpenH323 library; this offered us the advantage of being able to customise the code whenever necessary.

4.2.2 H.323 Gatekeeper

Three H.323 gatekeepers, from three different vendors, were deployed into the H.323 test environment. Two are commercial gatekeepers: Radvision's NGK-100 and VTEL's Encounter 3000. The third is an open-source application implemented using the OpenH323 library. The latter was most used during the research because of its open-source nature; for example, since I had the source code I could easily modify it, add to it, and debug it whenever necessary. The

main function of the gatekeeper in the network is to provide the address translation functions required for PSTN-H.323 interworking, discussed in Section 4.3.4.

4.3.3 Multipoint Control Unit (MCU)

The MCU was not used as a direct part of the research but merely completed the network in order to form a fully-fledged, component-rich H.323 environment. The MCU deployed for that purpose was also an open-source implementation developed using the OpenH323 project.

4.3.4 Gateways

H.323 gateways are other optional components of an H.323 environment. Their purpose is to reflect the characteristics of an H.323 network terminal to some other type of network terminal, and vice versa, in a transparent fashion. Two gateways are currently deployed in the project's test network. VTEL's Encounter 3000 H.323/H.320 gateway allows H.323 terminal users to engage in multimedia conferences with legacy H.320 videoconferencing stations connected to the Integrated Services Digital Network (ISDN). We considered that it would often be easier to call an H.320 legacy system from our PCs (or IP telephones) as opposed to using the H.320 system, which requires booking the department's conference room.

The second gateway is an open source H.323/ISDN gateway also implemented using the OpenH323 library. It provides connectivity to the Public Switched Telephone Network (PSTN) via a number of ISDN interfaces. The gateway basically allows users of the H.323 environment to make calls to traditional public telephones including cellular phones. The reverse is also possible, but the assistance of a gatekeeper is essential for reasons discussed below.

The gateway is a piece of software that resides on a PC that has an Ethernet interface and a number of ISDN interfaces. A maximum of two calls can be active per ISDN interface (2 x B channels), so by adding more ISDN interfaces to the gateway, the number of simultaneous voice connections can be increased. Presently, our gateway has three ISDN interfaces (two Gazel R841s, one Eicon 2.01) and thus supports a maximum of six voice calls.

Outgoing calls (from H.323 terminals to PSTN terminals) do not require the use of an H.323 gatekeeper (although use of a gatekeeper is strongly recommended, especially for admission control and billing purposes), while incoming calls require the use of an H.323 gatekeeper to identify the correct endpoint within the H.323 IP network.

Each telephone on the PSTN has a unique telephone number, and similarly, each H.323 terminal in an H.323 network has a unique IP address. As mentioned earlier, the gateway reflects the characteristics of an H.323 network terminal to another type of network terminal, in this case a Switched Circuit Network (SCN) terminal. This means that the gateway essentially has an IP address, for example 146.231.121.143, to identify it on the IP network as well as a telephone number, for example 046 622 2465, to identify it on the PSTN. To complete an outgoing call, the calling H.323 terminal need only know the IP address of the gateway and the number of the PSTN telephone it wishes to call. Now consider the reverse: To complete an incoming call the PSTN terminal dials the telephone number of the gateway, 046 622 2465. At this stage the gateway has no knowledge of which H.323 terminal is the intended callee. Here the gatekeeper performs its role; an H.323 terminal may register with the gatekeeper with an E.164 (numeric alias) number and a corresponding IP address. When the connection between the gateway and the PSTN telephone is established, the telephone user sends a sequence of numbers as Dual Tone Multi Frequency (DTMF) tones identifying the H.323 terminal to be called. The gateway accepts these DTMF tones and requests the gatekeeper to provide the corresponding IP address of the given extension number (alias). At this point the gateway has the IP address of the called H.323 endpoint and can forward the call accordingly.

This gateway is a fundamental component of the environment as it enables communication, particularly voice, between the PSTN and our H.323 network. This gateway is also the fundamental component in our network that makes our H.323 voice services available to PSTN users. Fortunately, this gateway is open source and the code has been accessible for modification whenever necessary (for example, for PSTN-H.323 enhanced service interworking).

4.3 H.323 Protocol Library

H.323 is a large and complex protocol; to develop a new stack from the ground up would require a substantial amount of time. A number of stacks have already been developed by various organisations, some commercial and some open source. However, the available commercial stacks were relatively expensive and using them was never contemplated. Rather, the possibility of acquiring demo versions at discounted rates to explore the functionality and performance of the stacks was investigated. Unfortunately, this was an unsuccessful ploy and instead I turned my attention to the open-source domain. Here I found a high-quality H.323 protocol stack developed by the OpenH323 project. The OpenH323 project was founded by Equivalence, an Australian development company in 1998. Since then the project has attracted many developers, telcos, and other corporates throughout the world. The OpenH323 protocol library officially supports version 3 of H.323 but it is in the migratory phase to version 4 and already supports many of the version-4 features. In some instances during the project, it was necessary to add version-4 features to the library in order to explore them completely.

Figure 4.2 illustrates a simplified entity relationship (ER) diagram of the OpenH323 library.

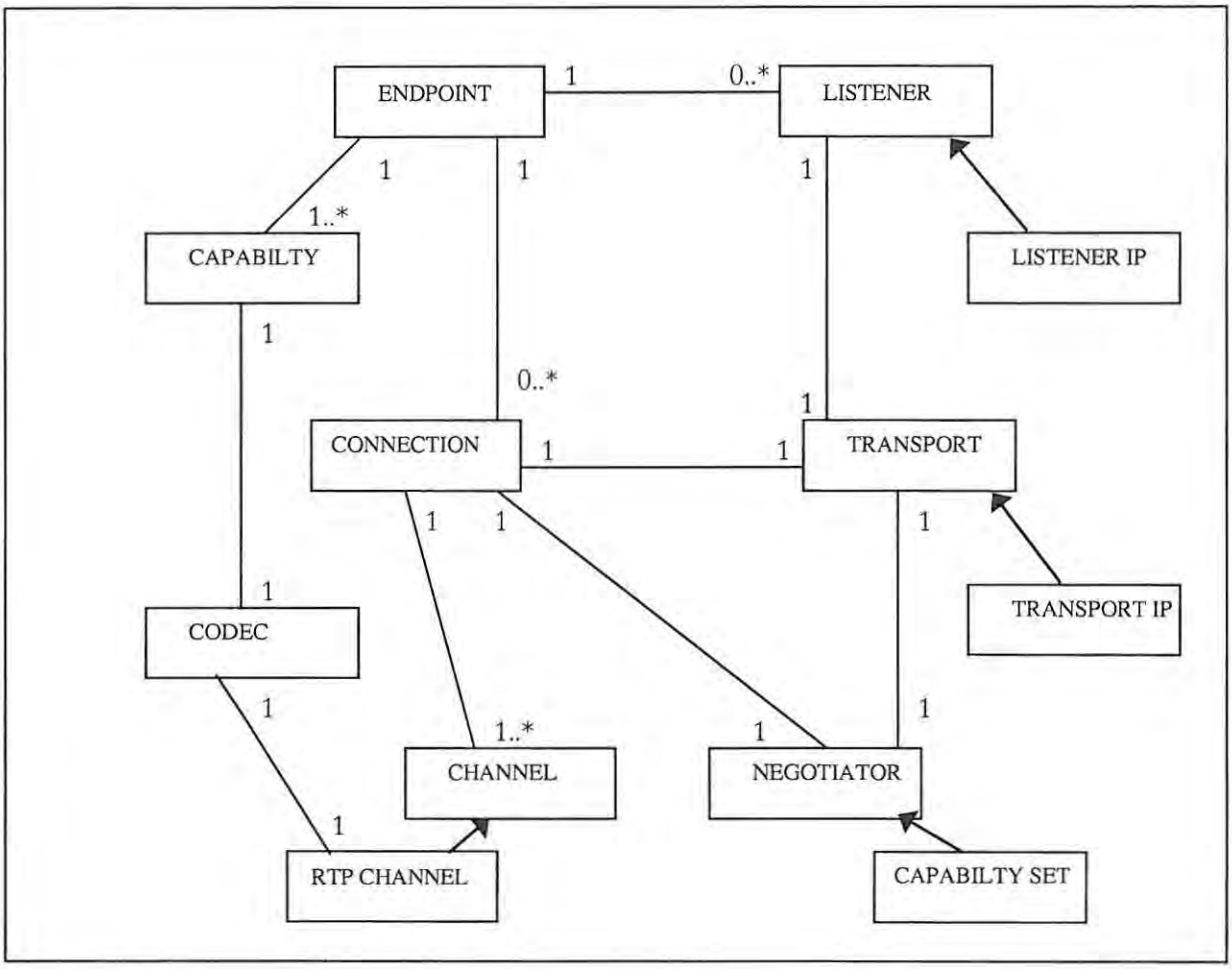

Figure 4.2 OpenH323 Library Architecture

The fundamental object of the ownership hierarchy is the *H323Endpoint* class. An application would typically have one instance of a descendant of this class. The application defined as the descendant would set up defaults for various H.323 parameters (timeouts, etc.), the most important being the capability table that defines the codecs and channel types that the application is capable of handling.

Also created by the application in the *H323Endpoint* would be instances of one or more descendants of the *H323Listener* class. There is a descendant of this class for each protocol that is supported. For instance, *H323ListenerIP* is for Internet use. Each listener spawns a thread that monitors its protocol and when a new incoming call is detected, and then creates an instance of an *H323Transport* class descendant. As for the *H323Listener* class, there is a descendant for each protocol supported (e.g. *H323TransportIP*).

When the first Protocol Data Unit (PDU) arrives on an *H323Transport* using the Q.931 and H.225 protocols, there is a call reference that identifies the connection that has been made. These connections are embodied by the *H323Connection* class, which contains all the state information for a connection between H.323 endpoints. The *H323Endpoint* instance keeps track of these active connections. If there is no connection for the call reference number, a new one is created and H.323 signalling negotiations begin.

An application would often have the system create an instance of a descendant of the *H323Connection* class, rather than the class itself. This is so that any of a large number of virtual methods may be overridden. These virtual methods are *callback* functions of the library to allow the application to either obtain information or modify the behaviour at various phases of the protocol negotiations. For example, there is a *callback* when an incoming call is in progress and the application user should be *alerted*. The code this callback executes is highly application specific and depends on the implementer of the application, for example, alerting the user with anything from simple audible beeps (resembling a ringing telephone) to displaying a pop-up window.

The *H323Negotiator* classes are used to maintain the state and functionality of each command or variable defined by the H.245 protocol. Their main reason for existence is actually to reduce the scope of the h225.h and h245.h files (C++ header files), which define many hundreds of classes. A user of the *H323Connection* class need not include all of these classes on every compilation unit.

During some of the H.245 negotiations, logical channels may be created, both by the remote endpoint and by the local application. The *H323Channel* class descendants represent these logical channels that are opened for various media formats. A typical use of one of these classes is to open a stream of encoded audio data. The *H323Channel* class would create a *H323Codec* using the *H323Capability* that was passed during the protocol negotiations [40]

This description of the OpenH323 library is a broad outline of the general operation of an H.323 application. When considering the service creation mechanisms of the library one must focus on the *H323Connection* class.

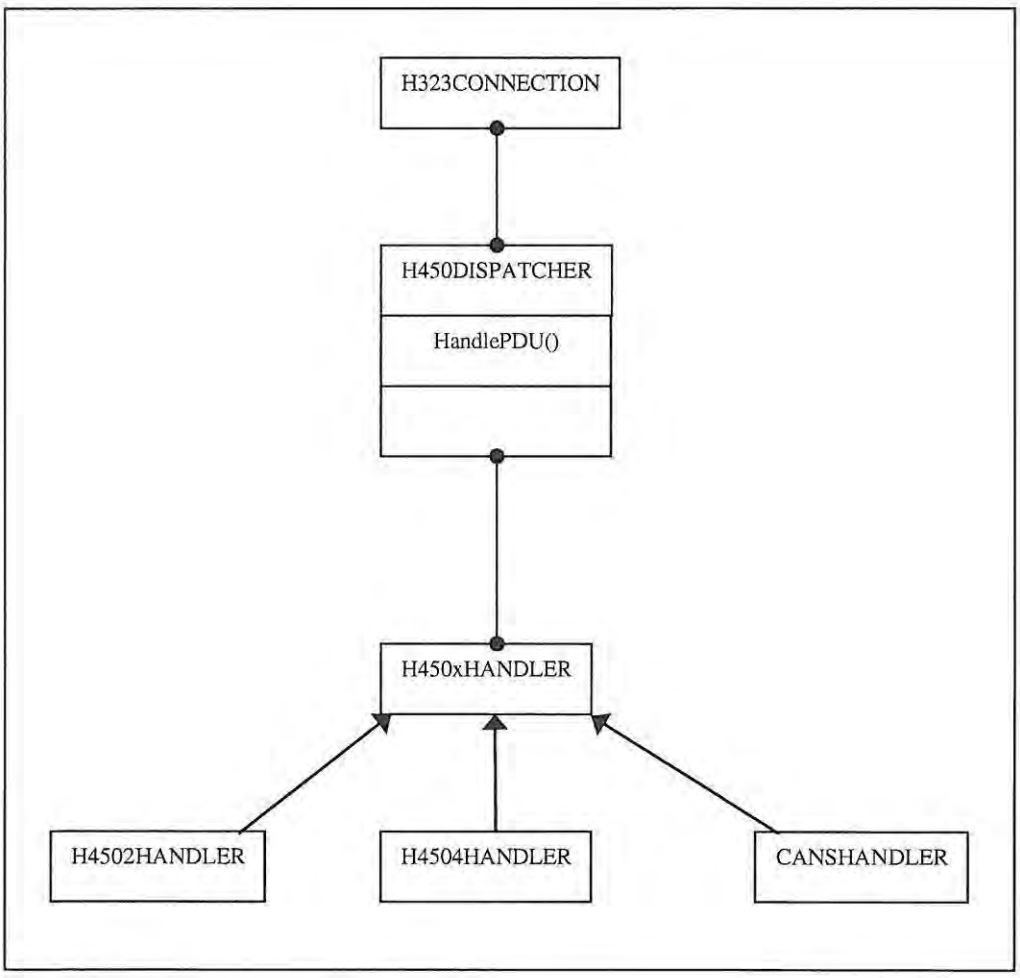

Figure 4.3 OpenH323 Library's H.450 Service Architecture

The *H323Connection* Object creates a single instance of the *H450Dispathcer* class. The *H450Dispatcher* class is responsible for handling all H.450 information within the H.225 call signalling messages. It then identifies the H.450 information according to the supplied operation codes, stored in a dictionary object. The information identified is finally dispatched to the appropriate *H450xHandler* object. The *H450xHandler* is an abstract class that specifies the base class for the specialised H.450 classes, like *H4502Handler*, *H.4504Handler* and *CANSHandler* in (Figure 4.3). *H.4502Handler* encapsulates all the operations and attributes that support the Call Transfer service, while *H4504Handler* does the same for the Call Hold service. The *CANSHandler* encapsulates all the operations and attributes that support the Customisable Alarm Notification System (CANS), an original H.450 service written during this research. CANS is the focus of Chapter 7.

So far I have covered the basics of the OpenH323 library's H.450 architecture. Currently the OpenH323 library does not support H.460. This meant that in order to investigate H.460 using OpenH323, I had to implement it. This was done in a relatively ad hoc manner whereby the H.460 operations were implemented on a per application basis. Rather than add H.460 support to the library, H.460 support was hard coded into the service applications manually. Further details of the H.460 implementations and the service implemented to explore its use are given in Chapter 7.

4.4 Portable Windows Library (PWLib)

The entire OpenH323 H.323 library is based on the Portable Windows Library (PWLib).

The OpenH323 project website describes PWLib as "a moderately large class library that has its genesis many years ago as a method to produce applications to run on both Microsoft Windows and Unix X-Windows systems." [41]

The library is a complete portability library with classes for Windows GUI portability, I/O portability, multi-threading portability, and support for creating both windows and Unix daemons and Windows NT services. It also supports a range of Internet protocols, like HTTP,

POP, FTP, etc. The HTTP Internet service was used to write a specialised web server for the CANS H.450 service, described in Chapter 7.

To use OpenH323's H.323 library I had to become experienced with PWLib first. This was a difficult task as it required me to become familiar with an entirely new language. Everything I wrote using the library was different to what I was used to when writing standard C++ applications, from string handling operations to primitive data types, like int, char, etc. The most difficult part of getting to terms with PWLib was the lack of documentation. The only documentation available is a single "hello world" example using the code on the OpenH323 website. Thus my knowledge of PWLib is largely self taught.

4.5 ASN.1 Compiler

An introduction to ASN.1 and further discussion of this compiler are provided in Sections 4.1 and 5.2, respectively.

The ASN.1 compiler was an essential tool for completing this research, because ASN.1 notations specify the structure and format of all H.323 messages. Additionally, information required for various H.450 and H.460 operations is often transported as ASN.1-encoded bit streams. Therefore, use of H.450, H.460, and other service features and frameworks in H.323 required extensive use of ASN.1 notations.

An initial investigation into obtaining a high-quality ASN.1 compiler proved unsuccessful for several reasons:

- Most were commercial implementations that were excessively expensive;
- Very few supported C++ (which is essential for use with OpenH323 and PWLib libraries), and
- Some did not support the ASN.1 Packed Encoding Rules (PER) that are otherwise required for H.323 messages (specified by ITU recommendation H.323 itself).

Tedious browsing through the complex OpenH323 and PWLib libraries to see how the H.323 messages were encoded resulted in a fortunate discovery. I understood that for the OpenH323 library to be complete there had to be some sort of ASN.1 encoding and decoding of messages. I initially believed that this support was hard coded within the numerous message parsing functions, but I was proved different: The OpenH323 library utilises its own fully-functional, complete ASN.1 compiler. This compiler is open source and developed using PWLib.

Using the compiler for our own system's messages proved relatively simple. Our own ASN.1 modules, which specified the operations for newly written services, were simply passed to the compiler at the command line as follows:

```
asnparser -c my_module.asn
```

This command could generate the appropriate C++ header and source files to be used within our H.323 applications in order to build, encode and decode the necessary information, as described with ASN.1 specifications for our new services.

4.6 Summary

In this chapter I introduce and describe the test environment deployed for the research. The H.323 version-4 test environment is complete and includes all possible H.323 components: Terminals, gatekeepers, gateways, and an MCU. The environment is H.323 version-4 compliant, allowing one to explore the service creation mechanisms supported only by version 4 of H.323. The simplified description omits minor details of the various components that were deployed.

In order to develop new H.323 applications and services a complete and stable H.323 software protocol stack is required. The stack I chose for this work was an open-source implementation provided by the OpenH323 project. The chapter provides an introductory description of this stack as well as the underlying PWLib library it has been implemented with. Using PWLib, the H.323 applications and services that were developed for the project could be run transparently on Windows and Linux.

To create new H.450 services, and in some cases H.460 services, an ASN.1 compiler is required to translate the ASN.1 modules, which are themselves used to describe the message format and operations of new services, into a particular programming language (C++ for use in the OpenH323 library). This chapter concludes by introducing and describing the use of such a compiler. Although ASN.1 is not introduced in detail in this chapter, interested readers are referred to Appendix A for an example-driven description.

CHAPTER 5

H.323 V.4 SIGNALLING SERVICE CREATION AND MANAGEMENT MECHANISMS

"There are two ways of constructing a software design; one way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."

C.A.R Hoare

H.323 service creation requires solid, mature knowledge of the protocol and its underlying message structures. Here, I address the H.323 version-4 service creation mechanisms (H.450, H.460) in greater detail than in Chapter 2. Thereafter I describe the interworking of these service mechanisms, which reside in the service control layer, with the H.323 call control layer (H.225). A new service may be implemented using either H.450 or H.460, depending on its complexity, and I discuss guidelines that can be used when trying to decide which mechanism to use. Finally, I introduce H.323 Annex K, a service management feature that is now included in H.323 version 4. Service management in H.323 provides users with a means of administering and customising the services they use, using a standard HTTP connection.

5.1 Introduction

To create new services in H.323 one needs to add functionality to the H.323 entities. Often this new functionality requires peer-to-peer communication between H.323 entities; and therefore we have to add new functionality to the H.323 call signalling messages in order to support the service. The addition of such functionality in H.323 entities and their communications messages can be achieved in three ways:

- Extension of the base H.225 ASN.1 base definition;
- Definition of a new H.450 Supplementary Service (SS);
- Definition of a new H.460 Generic Extensible Framework (GEF) module.

Extension of the base H.225 ASN.1 base definition requires ratification by the ITU. Being an international standards body, the ITU is only concerned with the ratification of services deemed important to the operation of all H.323 applications. Alternatively, one's own implementation-specific extensions can be added to the base specification without ITU standardisation. But services created in this manner compromise the standard nature of H.323 and are detrimental to H.323 interoperability.

The other two methods, involving H.450 and H.460, allow one to create and negotiate services without affecting the underlying H.225 ASN.1 base specification. H.450 provides advanced tools for service creation that reach beyond a simple information passing mechanism. H.460 provides mechanisms for information transfer between H.323 entities and may also be used for service creation. However, H.460 does not provide the advanced tools available in H.450 and it can only be used for services that require little information transfer. H.460's simplicity and ease of use has led to its adoption as the H.323 feature negotiation mechanism, introduced in section 2.4.2.

One of the innovations of H.323 version-4 is Annex K, *HTTP Based Service Control Transport Channel*. Annex K allows third-party control of an H.323 service based on a separate control channel (using HTTP) for user interaction. This service control channel is

intended to be used by a wide range of services, some of which may require the use of H.450 for invocation or execution [24].

To understand how H.450, H.460, and Annex K work and interact with H.225, we need to understand Abstract Syntax Notation Number One (ASN.1). ASN.1 is a formal data structure description tool used in many application domains (i.e., it is not bound to telecommunications applications). ASN.1 is used to specify the structure of all H.323 protocol messages responsible for call signalling, call control, and service control. The use of ASN.1 is part of an ongoing effort by the ITU to ensure that all implementations of H.323 are interoperable. Appendix A presents an example-driven introduction to ASN.1, which is useful to understand the remainder of this chapter.

5.2 H.450 Protocol Revisited

Using ASN.1 we can explore H.450 in more detail. This section of the work aims to supply enough detail on H.450 to ensure a smooth reading of Chapter 7, where a unique H.450 service, CANS, is described. Interested readers may obtain more information about H.450 from the ITU H.450.1 recommendation, *Generic Functional Protocol for the Support of Supplementary Services in H.323* [26].

In many ITU standards, including H.450, the term Application Protocol Data Unit (APDU or sometimes just PDU) is often used. Basically, A PDU encapsulates information that is delivered as a unit between entities of a network [2]. For example, H.225 signalling messages are transmitted as PDUs (e.g., H.225 Setup PDU) from one H.323 entity to another.

In layered systems, the term PDU is used to refer to the encapsulation of data specified by a particular layer. For example, we will see later in this chapter how a ROS (a description of the remote operations service follows in the next paragraph) APDU is encapsulated within an H.450 Supplementary Service (SS) PDU, in turn encapsulated within an H.225 signalling PDU. These three PDUs represent the three protocol layers responsible for H.450's operation: the H.225 layer, the H.450 SS layer and the ROS layer. Each layer strips its

information from the PDU it receives and passes the remainder of the PDU information to the underlying layer until all layers have received the required information.

H.450 uses the services of the Remote Operations Service (ROS), which is based on ITU recommendation X.880. Olivier Dubuisson, in his book *ASN.1 Communications between Heterogeneous Systems* [8], describes ROS as:

"a very general client-server mechanism, which hides from the application programmer the existence of a communication between processes; it can ask the remote application to execute operations or to collect results and errors; each interface's operation is described with ASN.1 as an information object of the OPERATION class; ROS provides a common and standardized method for carrying requests and answers laying by specific gaps in the APDU to be filled in dynamically during communication."

ROS is therefore a generic standard method of inter-process communication of the request-response paradigm that is used for interactive communication between objects regardless of where they are located. The basic interaction type is the invocation of an operation by one ROS-object (the *invoker*) and its execution by another (the *performer*). Completion of the execution of an operation, successful or not, may lead to the return of a report of its outcome (from the performer to the invoker). A report of the successful completion of an operation is a *result* while a report of unsuccessful completion is an *error*.

ROS was released as a final recommendation by the ITU in July 1994. Since then other technologies have emerged in the computer science arena that address remote operations. These newer technologies include Java's Remote Method Invocation (RMI) and Common Object Request Broker Architecture (CORBA), both of which provide a means of invoking operations on remote hosts.

H.450.1 Supplementary Service (SS) APDUs encapsulate information for the operation of an H.450 service transported between H.323 entities. SS APDUs are transported within the H.225 call signalling channel and included in the User-User Information Element of H.225 messages (see section 5.4.2). Their purpose is to convey one or more ROS APDUs and an optional *Interpretation* APDU. The Interpretation APDU is used by an H.323 sending entity

to convey instructions to the receiving entity regarding the handling of unrecognised ROS APDUs.

Although each SS APDU can carry more than one ROS APDU, every ROS APDU must be delivered to the same target entity (i.e., ROS APDUs within the same SS APDU may not be delivered to different H.323 entities). If such a situation is required, then the ROS APDUs must be included in different H.450 SS APDUs. For example, if a terminal wants to invoke two H.450 services at a single remote terminal, it can include both ROS invoke APDUs within the same SS APDU. However, if an H.323 terminal with two connected hosts, for example an MCU, intends to invoke two H.450 services on the two separate terminals, it must send each ROS invoke APDU within a separate H.450 SS APDU.

The following fragment illustrates the ASN.1 definition of an H.450.1 SS APDU:

```
1               InterpretationApdu ::= CHOICE
2                     {
3                     discardAnyUnrecognizedInvokePdu NULL,
4                     clearCallIfAnyInvokePduNotRecognized NULL,
5                     rejectAnyUnrecognizedInvokePdu NULL,
6                     ...
7                     }
8               ServiceApdus ::= CHOICE
9                     {
10
11                    rosApdus SEQUENCE SIZE (1..MAX) OF ROS{{InvokeIdSet},
12                    {OperationSet}, {OperationSet}},
13                    ...
14
15                    }
```

Lines 1 to 7 specify the structure of the Interpretation APDU introduced earlier. Within the enclosing brackets {} of the Interpretation APDU are the three courses of action a terminal can make if it receives an unrecognised ROS Invoke APDU. Depending on what the sending terminal decides, the receiving terminal can:

- Ignore the unrecognised ROS Invoke APDU (`discardAnyUnrecognisedInvokePdu`);
- Clear the call (`clearCallIfAnyInvokePduNotRecognised`);

- Reject the ROS Invoke APDU by sending a ROS Reject APDU (`rejectAnyUnrecognisedInvokePDU`).

Lines 8 to 15 specify the structure of the H.450 SS APDU, formally called the `ServiceApdus` (line 8). The `ServiceApdus` field comprises one or more ROS APDUs (defined as SEQUENCE OF ROS). Each ROS APDU can be an alternative of ASN.1 type ROS. In accordance with X.880, there are four ROS APDU types:

- Invoke
- Return result
- Return error
- Reject

Invoke, Return result, Return error and Reject ROS APDUs may be interpreted and used differently by each individual H.450 SS. Their use therefore must be defined in each new SS by specifying the operations that control the service. These operations are defined using ASN.1 in the relevant SS specifications, and must define what an H.323 entity should do prior to sending, or on receipt of all ROS APDUs.

5.3 H.460 Protocol Revisited

As with H.450, described above, the material presented here is a limited description of H.460's operation. More information can be obtained from the ITU H.460.1 recommendation, *Guidelines for the Use of the Generic Extensible Framework* [27].

Feature negotiation in H.323 networks involves information requirements of the various services to be passed among H.323 entities. H.460 is designed for passing information between H.323 entities and it can be easily used to describe all services in H.323. For this reason H.460 has been adopted as the H.323 version-4 feature negotiation mechanism.

H.460 can also be used in H.323 version 4 networks for service implementation. In contrast to H.450, H.460 is merely a mechanism for information passing between H.323 entities. It does not provide advanced tools for service operation. This limits the nature of services implemented directly using H.460 to simple ones with relatively few parameters. On the other hand, H.460 has less overhead and is simpler to use in comparison to H.450.

All H.323 services that use H.460 for feature negotiation or service creation, do so by means of a GEF module, which can be both ITU-standardised and non-ITU-standardised (user-defined). All ITU-standardised modules are defined as part of the H.460 series of recommendations. For example, H.460.2 defines the ITU-standardised service for Number Portability Interworking between H.323 and Switched Circuit Networks (SCN). (Number Portability allows subscribers to keep existing phone numbers when changing from one service provider to another.)

Identification of GEF modules within H.323 networks is essential for supporting feature negotiation procedures and for maintaining service interoperability among H.323 entities. Identification of all ITU and non-ITU standardised modules is addressed by H.460 and interested readers may refer to Chapter 7 (section 7.5.2) for a detailed, example-driven discussion of using H.460 in non-ITU standardised services. Further information can be obtained from ITU Recommendation H.460.1 [27].

Before discussing how GEF works it is important to address the way in which we can specify services using H.460. In other words, there is a separation between the way in which we specify services and how we implement them using GEF. The specification of the service refers to the way in which the service's operation is defined. For example, in programming we define a function by identifying the information that the function requires for its operation (parameters), the operation the service performs on the parameters, and the result which is returned by the function. Using pseudo-code we can specify such functions independently of any particular programming language and then implement them in any language we wish to use.

Similarly, in H.460 we can specify services using the following two methods:

- Table-based method,
- abstract notation method.

Using these methods we specify GEF modules which represent the information that is required for the operation of a particular service (e.g., the parameters of the service).

In the table-based method, information required for a particular service or feature is described in tabular form. An instance such as shown in Table 5.1 is used to describe the feature as a whole, capturing the feature's name, its identifier and a brief description.

Feature name	The name of the feature
Feature description	Short description of the feature
Feature identifier type	Indicates whether the feature is standard, OID, or non-standard
Feature identifier value	The actual value of the identifier

Table 5.1 H.460 Service Description Table

Each parameter required for the operation of the feature is described using an instance such as shown in Table 5.2.

Parameter name	The name of the parameter
Parameter description	Short description of the parameter
Parameter identifier type	Indicates whether the parameter is standard, OID, or non-standard
Parameter identifier value	The actual value of the identifier
Parameter type	The type of the identifier, i.e. integer, bool, raw, etc.
Parameter cardinality	How many times the parameter may occur

Table 5.2 H.460 Parameter Description Table

To illustrate, one of the parameters defined in ITU Recommendation H.460.3, *Presence in H.323 Systems*, is specified using the table-based method shown in Table 5.3.

Parameter name	Priority
Parameter description	Integer value of the priority of the contact (0=highest)
Parameter identifier type	standard
Parameter identifier value	3 (3rd parameter)
Parameter type	number8
Parameter cardinality	1

Table 5.3 H.460 Example Parameter Description Table

The second method of GEF module specification uses ASN.1, XML, ABNF, or similar notation. In this case the feature is specified using a formal description language like ASN.1. This method of feature specification is particularly recommended for features that contain a large number of parameters making the table-based method of feature specification cumbersome because one table is required for every parameter. For example, the Number Portability feature (H.460.2) is specified using ASN.1. (The complete ASN.1 syntax for this feature can be found in Appendix B.3.)

Feature negotiation procedures of GEF always make use of the table-based method of feature specification because it only involves identification of the service to be passed between H.323 entities. Identification of a particular service is equivalent to the specification of a single service parameter using the table-based method (i.e., there is no complexity or abundance of parameters required for feature negotiation in H.323).

New services, implemented using H.460, can be formally specified using either of the specification methods (table-based or abstract notation methods), depending on the service's complexity and number of parameters.

Having addressed the specification of H.460 features, I address the way in which H.460 (GEF) modules are transported within the H.225 call signalling messages in the following section.

5.4 H.225–H.450 and H.225–H.460 Interaction

5.4.1 Introduction

The details of the H.450 and H.460 protocols have been explained above, but we have not yet described the way in which the information defined in these two standards is transported from one H.323 entity to another.

In the H.323 protocol, H.450 and H.460 reside within the service control layer, while H.225 exists within the call control layer. There is a functional separation between these two layers, but at the same time there is a strong interaction between them. This interaction is due to the reliance of the service control layer on H.225 for its transport among H.323 entities. The

transport of H.450 and H.460 within H.225 messages is standardised and specified in the H.225 specification itself. In the next two sections, 5.4.2 and 5.4.3, I will show where H.450 and H.460 information is embedded within H.225 messages.

5.4.2 H.225–H.450 Interaction

Within the ASN.1 specification of all H.225 call signalling messages there is a field called `h4501SupplemtaryService` and it is defined as (The complete H.225 ASN specification can be found in Appendix B.1):

```
h4501SupplementaryService  SEQUENCE OF OCTET STRING OPTIONAL
```

From this line we can see that all H.450 supplementary service (SS) information is encoded as an OCTET STRING, a string of hexadecimal bytes. Each SS APDU will be encoded as a single OCTET STRING. If more than one SS APDU is to be transported within the same H.225 message, then they shall be encoded as separate OCTET STRINGS and sent as a SEQUENCE (i.e., an array of OCTET STRINGS). The OPTIONAL identifier in the `h4501SupplementaryService` definition indicates that the presence of this field in H.225 messages is optional, that is all H.225 messages that do not contain H.450 SS information remain valid H.225 messages.

5.4.3 H.225–H.460 Interaction

All H.460 information transported via H.225 messages is encoded within the `genericData` field of the H.225 ASN.1 specification. The `genericData` type is defined as:

```
genericData                SEQUENCE OF GenericData OPTIONAL
```

From this line we see that `genericData` is defined as an optional array of type `GenericData` elements. It is identified as optional by the ASN.1 OPTIONAL keyword; this simply means that H.225 messages are not required to have the `genericData` structure to be valid H.225 messages. In other words, if H.460 information is to be transported to other H.323 entities, it can be integrated into any H.225 call signalling message using the

genericData field. If no H.460 information is required, then the entire genericData field can be omitted.

The GenericData type is defined as a SEQUENCE of the feature's identifier followed by an optional array of feature parameters that are encoded as EnumeratedParameter types.

```
GenericData ::= SEQUENCE
{
     id                    GenericIdentifier,
     parameters            SEQUENCE (SIZE (1..512)) OF EnumeratedParameter
     OPTIONAL,
     ...
}
```

The GenericIdentifier type is defined as an integer (for identifying standard H.460 features) (e.g., H.460.2 will have a value of 2), OBJECT IDENTIFIER (for a standard service not ratified by the ITU), or a GloballyUniqueID (for identifying proprietary services). *GloballyUniqueID* is a unique byte stream which identifies the service in the proprietary domain.

```
GenericIdentifier ::= CHOICE
{
     standard              INTEGER(0..16383,...),
     oid                    OBJECT IDENTIFIER,
     non-Standard          GloballyUniqueID,
     ...
}
```

The EnumeratedParamter type is defined as a SEQUENCE of id (identifying the specific parameter, 1 for first parameter, 2 for second parameter, etc.) and the actual value of the parameter encoded as type Content.

```
EnumeratedParameter ::= SEQUENCE
{
     id                    GenericIdentifier,
     content               Content OPTIONAL,
     ...
}
```

`Content` is defined as a choice of all the types encapsulated within the { } of the `Content` structure (e.g., raw, integer, bool, etc.).

```
Content ::= CHOICE
{
     raw                 OCTET STRING,
     text                IA5String,
     Unicode             BMPString,
     bool                BOOLEAN,
     number8             INTEGER (0..255),
     number16            INTEGER (0..65535),
     number32            INTEGER (0..4294967295),
     id                  GenericIdentifier,
     alias               AliasAddress,
     transport           TransportAddress,
     compound            SEQUENCE (SIZE (1..512)) OF EnumeratedParameter,
     nested              SEQUENCE (SIZE (1..16)) OF GenericData,
     ...
}
```

The `Content` structure describes how a specific feature implementation is specified. In section 5.3 I introduced the two methods of GEF module specification: the table-based and abstract notation methods. The feature negotiation mechanisms of H.460 always use the table-based method, while new features can be specified using either method. For instance, a feature parameter specification similar to the table-based specification of the priority parameter presented in Table 5.4 would be encoded as the `number8` variant of the `Content` structure, which is a number between 0 and 255.

If the feature is specified using the abstract notation method, such as the `numberPortabilityInfo` parameter given in Appendix B.3, then the raw variant of the `Content` field is used. The raw variant is of ASN.1 type OCTET STRING and carries the encoded ASN.1 information structurally specified by the ASN.1 module also provided in appendix B.3. The receiving entity recovers the OCTET STRING from the parameter and decodes the information into its corresponding data structures for processing.

When using the feature negotiation functionality of GEF, the needed, supported and desired feature sets are encoded within the `FeatureSet` type:

```
FeatureSet ::= SEQUENCE
{
     replacementFeatureSet    BOOLEAN,
     neededFeatures           SEQUENCE OF FeatureDescriptor OPTIONAL,
     desiredFeatures          SEQUENCE OF FeatureDescriptor OPTIONAL,
     supportedFeatures        SEQUENCE OF FeatureDescriptor OPTIONAL,
     ...
}
```

Each feature set is defined as an array of type `FeatureDescriptor`, which are in turn of type `GenericData` (the same type described above):

```
FeatureDescriptor ::= GenericData
```

When specifying the feature we only need to make use of the `id` field of the `GenericData` structure. For example, H.460.3, *Presence in H.323 Systems*, would be described as the standard variant of `GenericIdentifier` and given, rather naturally, the value of 3.

5.5 Choosing H.450 or H.460

From what has been said above, H.460 is a mechanism for H.323 service negotiation as well as for simple services. This means that new services can be created using H.460 or H.450. The designer of a new service is therefore faced with the decision of which mechanism to use. To assist in making the decision it may be appropriate to consider the service negotiation and the data transport aspects of the service separately. For example, it may be appropriate to implement a new service primarily as an H.450 SS, yet negotiate it using H.460's feature negotiation procedures. This is exactly how I developed the CANS service discussed in Chapter 7.

To decide which method should be used to add new service functionality, the following guidelines are suggested (taken and modified from H.460.1 [27]).

- If the feature is applicable to a very large number of applications, then it may be appropriate to add it to the H.225.0 ASN.1 base specification directly.

- If the feature requires few parameters or data to be transported in H.225 RAS or H.225 Annex G, then define it as a GEF module.

- If the feature contains numerous parameters, then define it as either ASN.1 (or similar method such as ABNF or XML) to be encoded into a raw element of a GEF parameter construct, or define it as an H.450 supplementary service. In this case the decision to use H.450 or H.460 will be based on the overall complexity of the service. Although the number of parameters is usually related to a service's complexity there is a significant distinction. As an example: the operation of fictitious services X and Y require 50 parameters for their operation. However, service X only requires a single array-based transport of the 50 parameters from the invoker to the performer of the service, while service Y requires a more complex communication involving all 50 parameters, between the invoker and performers of the service. Hence, service Y is more complex as it requires a more detailed communication that may require condition processing, timers, etc.

Figure 5.1 illustrates these considerations in the form of a flow chart.

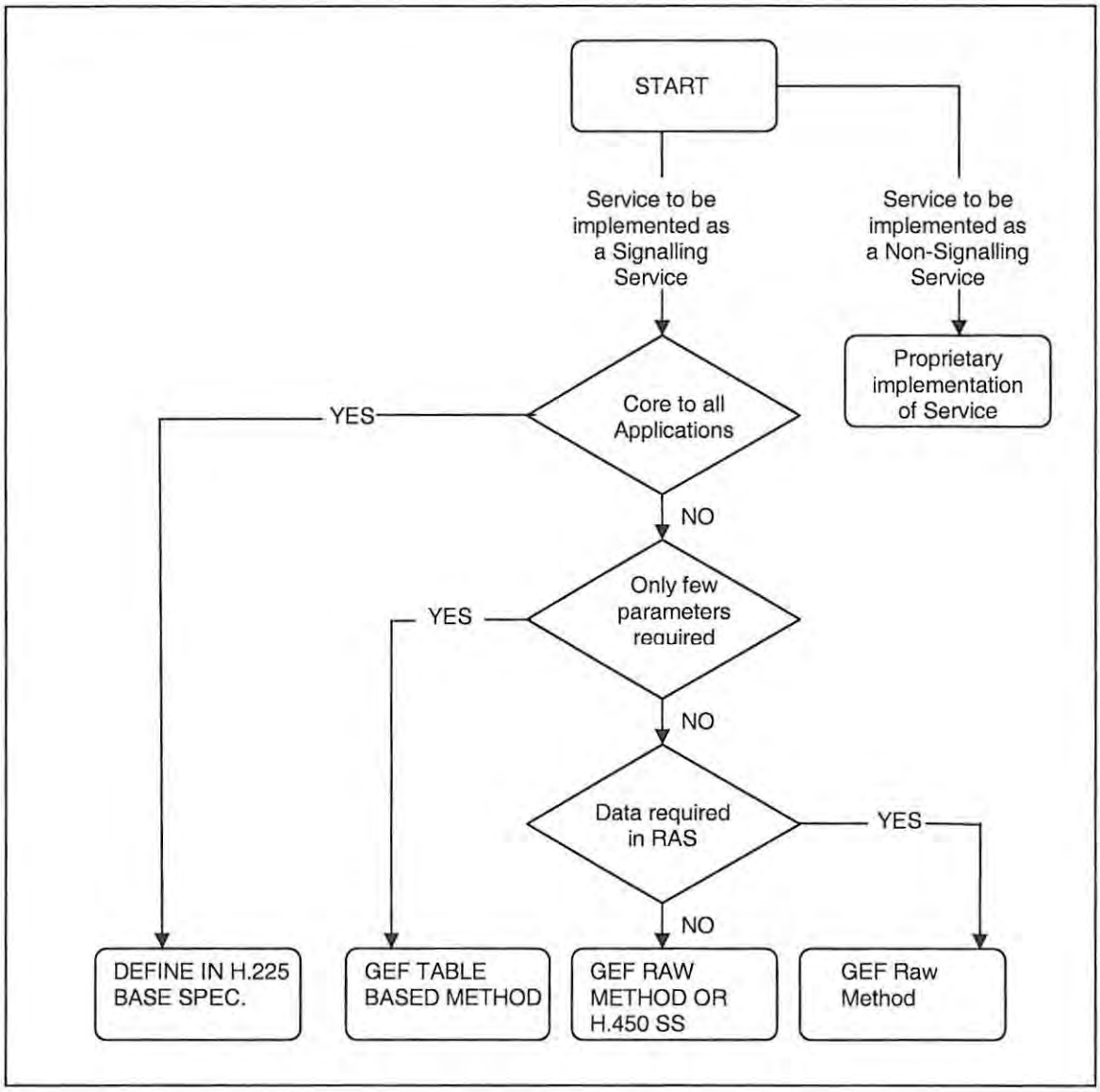

Figure 5.1 Decision Tree for Specifying H.323 Services (taken ad modified from H.460.1 [27]

From this figure we see that before deciding to use H.450 or H.460 one needs to determine whether or not the service will be implemented as a signalling service. Only signalling services are implemented using H.450 or H.460, while non-signalling services are implemented in a relatively ad hoc manner. When considering the implementation of a new

service as being signalling or non-signalling it is important to consider the need for the service to interact with other H.323 network entities, like gatekeepers.

If the implementer of the service would like the service to be used by other vendors' products, then it should be implemented as a signalling service. Once other vendors have the service specification they can integrate support for the service into their products in an H.323-compliant manner. Finally, if the service involves complex communication procedures between H.323 entities, with numerous parameters being passed back and forth, then it is ideal to implement the service as a signalling service because of the functionality offered by the H.450 and H.460.

5.6 Annex K

In some cases a service provider may want their customers to have some control over the operation of the services provided. This scenario is referred to as service management. Service management in H.323 version-4 networks is addressed by H.323 Annex K, *HTTP Based Service Control Transport Channel*. Annex K allows third-party control of an H.323 service based on a separate control channel (using HTTP) for user interaction. This service control channel is intended to be used by a wide range of services, some of which may require the use of H.450 for invocation or execution [24].

Annex K allows Uniform Resource Locaters (URL) to be conveyed in both call-related (H.225) and non call-related (RAS) H.323 messages. When an endpoint receives a service control session command, for example an OPEN command, it is responsible for opening an HTTP channel to the given URL. As a result the client endpoint is able to interact with the service provider using HTTP and so essentially control the service on offer.

It is important to distinguish between the two logical layers that exist within an endpoint that supports H.323 Annex K. Firstly, there is the conventional H.323 layer, which represents a basic H.323 call, and is defined as the Call Control Plane. The second layer, Service Control Plane, leverages itself above the call control plane and represents the HTTP control channel of Annex K (Figure 5.2).

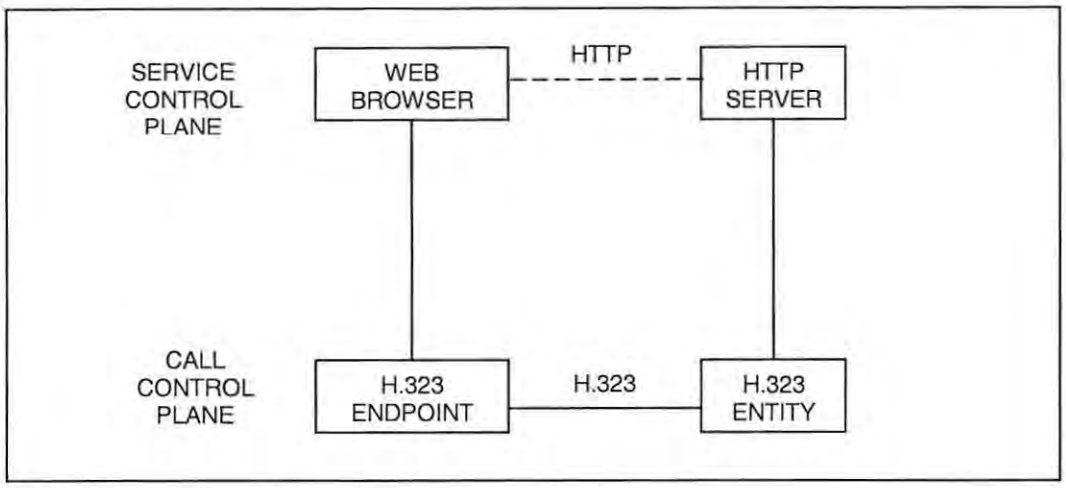

Figure 5.2 System Overview for HTTP based Service Control

Annex K does not formally define the interaction between the Call Control Plane and the Service Control Plane, leaving the vendor the responsibility of providing mapping between the two. Of course it is up to the provider of the URL (service provider) to implement the control functions and services at the URL presented.

Annex K information is transported within H.225 messages; the way in which H.225 transports this information is discussed in section 7.5.3.

Support for Annex K was included in our H.323 Signalling service called CANS (Chapter 7). Since the open-source H.323 library OpenH323 did not support Annex K, it was implemented by us and integrated into the library.

5.7 Summary

In this chapter I described the H.323 version-4 service creation mechanisms, H.450 and H.460. These mechanisms are used to create signalling services in H.323 version-4 networks. Because signalling services require interaction with the H.225 call control protocol of H.323, I have shown how H.450 and H.460 information is integrated within H.225 messages.

Both H.450 and H.460 can be used to create signalling services in H.323, which means that implementers wanting to create new H.323 signalling services are faced with the decision of which mechanism to use. A decision tree was presented that can be used to help decide which mechanism to use in the implementation of a new signalling service.

Finally, I discussed the service management functionality offered in H.323 version-4 networks via H.323 Annex K, *HTTP Based Service Control Transport Channel*. Annex K allows third-party control of an H.323 service based on a separate control channel (using HTTP) for user interaction. This service control channel is intended to be used by a wide range of services, some of which may require the use of H.450 for invocation or execution.

CHAPTER 6

H.323 NON-SIGNALLING SERVICE CREATION

"If we can dream it, we can build it."

Boeing

This chapter discusses EmailReader, an H.323 non-signalling service developed for the project. EmailReader adds mobility to conventional email by synthesizing voice messages of users' emails and playing them to users on their telephones (H.323 SoftPhones, conventional PSTN telephones, and GSM cellular telephones). Non-signalling services are usually based on standard, callable H.323 terminals. The terminals have various amounts of embedded intelligence responsible for the majority of the service's execution logic. In other words, H.323 non-signalling services are created using standard, callable H.323 terminals, implemented with additional logic tailored to carry out a particular service. Two non-signalling services were developed during this study; EmailReader, as mentioned, is based on a standard H.323 terminal, while the second, Telgo323, is based on a gateway between the PSTN and H.323 networks. Telgo323 is an H.323 non-signalling service that enables a standard PSTN user to communicate with a deaf user employing a form of text telephone, called a Teldem, via the PSTN [45]. This chapter is concerned with EmailReader only and readers interested in Telgo323 are referred to Appendix C.

6.1 Introduction

Once familiar with the general principles of H.323 and the protocol stack I had at my disposal, I embarked on an exploration into practical H.323 service creation, with an initial focus on non-signalling service creation. This empirical exploration of H.323 service creation was conducted by the implementation of our own large-scale non-signalling service, which I called EmailReader. EmailReader's design and implementation is the focus of this chapter, with an emphasis on the H.323 non-signalling nature of the service.

EmailReader enables H.323 users to have their emails read to them as voice messages on their standard H.323 terminals. To use the service users simply dial the EmailReader service terminal (using IP address, DNS name, or E.164 address) from their H.323 terminal equipment. Once connected, EmailReader will download the user's email messages from a mail server and essentially read them using speech synthesis. My implementation of EmailReader allows it to be easily extended to provide any sort of information via a speech synthesis system. For example, by replacing the backend mail server connection with a database connection, students can have exam results or account balances read to them. In fact, the system could support a number of backend interfaces offering users the ability to select the information they want read upon connection. Naturally, in that case EmailReader's name would have to change to something more appropriate like InfoReader.

EmailReader is implemented as a specialised H.323 terminal that complies with the H.323 recommendation completely. No modifications were made to the call signalling or call control messages in order to support the service. Instead, a large amount of logic was integrated into an H.323 terminal to create EmailReader. In this manner EmailReader can be used by any standard H.323 terminal, of which our test network supports the following:

- H.323 software terminals running on personal computers (Linux and Windows), and
- wireless access devices, like Compaq IPAQ Pocket PC.

Using the H.323/ISDN gateway, introduced in Chapter 4 (section 4.3.4), I extended the accessibility of the service to traditional PSTN and GSM users, thus allowing standard PSTN telephone and GSM cellular phone users to dial into the EmailReader on an H.323 terminal. A few modifications to the gateway were required and they are discussed later in the chapter.

Figure 6.1 illustrates the complete system architecture of the EmailReader service with all the components required for its operation. The heart of EmailReader is the callable H.323 endpoint (H.323 EmailReader labelled A in Figure 6.1) with its embedded intelligence.

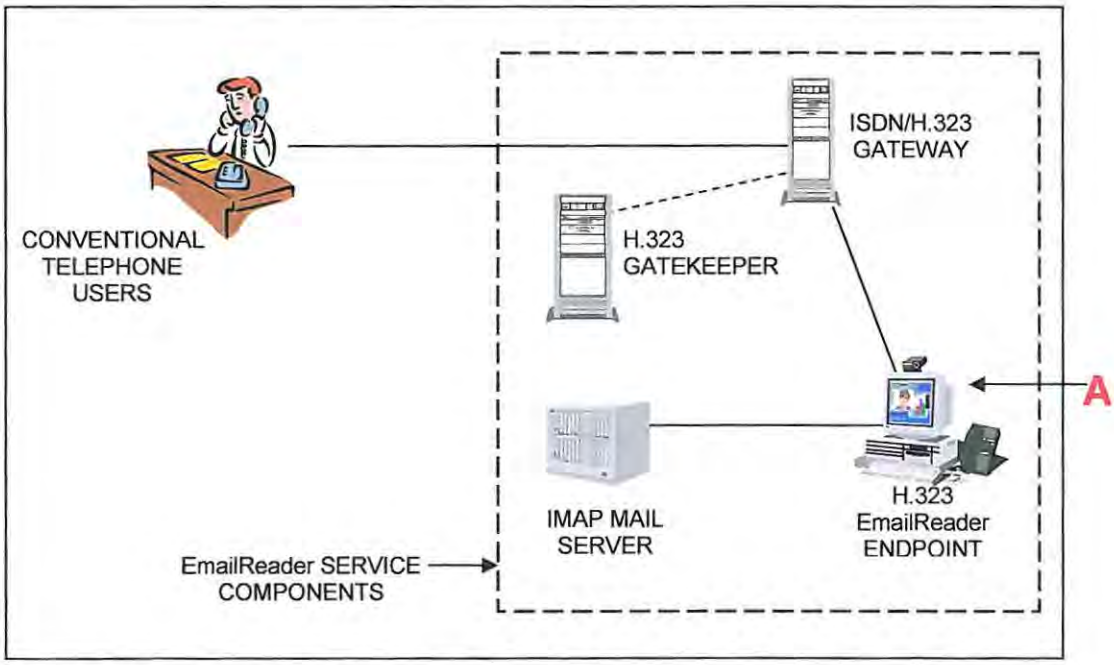

Figure 6.1 EmailReader Architecture

6.2 Design

EmailReader was designed as a callable H.323 terminal. Therefore, it can engage in an H.323 call with other H.323 callable entities, such as terminals, gateways, or MCUs. To perform its function the basic EmailReader terminal requires the aid of other computing technologies that enable it to:

- Communicate with a public mail server to access users' email messages;
- Synthesize speech from text and in so doing, generate the audio from the emails.

The public mail server at Rhodes University serves staff and students, totalling approximately 4000 users. Using the server in the EmailReader service meant that the service could be made available to the majority of the university community. To enable EmailReader to communicate with the server, I needed to integrate a mail access protocol. Two common protocols are the POP and IMAP protocols, both supported by the Rhodes University mail server. IMAP is the newer of the two protocols, and in keeping with the latest technologies I chose to integrate the IMAP protocol into EmailReader as opposed to the older POP protocol.

Email is text based; all email messages are stored as text files on the mail server's local disk. In order to have these messages read to a user via an H.323 terminal or telephone, they must be converted into voice. Text-to-speech (TTS) systems take text files as their input and synthesize audio files from them. An abundance of such systems exist and one had to be chosen for use in the EmailReader application. The University of Edinburgh's Centre for Speech Technology Research (CSTR) is highly accredited and has developed a complete TTS system called *Festival*. Some reasons for choosing this system were that it is open source, it is implemented in C++ (enabling easier integration into the EmailReader application), and it offers many unique features not offered by other systems, such as the inclusion of a Scheme-based command interpreter. Using Scheme I could control the way in which the emails are synthesised into voice messages. For example, consider the following email taken from the Rhodes mail server:

```
That is fine Jason. See you tomorrow.

Alfredo

On Thursday, June 07, 2001 1:56 PM, Jason Penton
[SMTP:g97p5142@campus.ru.ac.za] wrote:

> Hi Alfredo
>
> It looks like I won't be finished before 5 pm; will email you tomorrow as
> soon as I am done.
>
> Jason
```

This email contains correspondence between two parties, with lines of text from the previous message denoted with ">". Using Scheme I can instruct *Festival* to synthesize the two threads of conversation using different voices. This feature was incorporated in the EmailReader system and made the reading of mail messages as voice messages more understandable over the phone. IMAP and TTS constitute the embedded intelligence that is characteristic of many H.323 services, especially the non-signalling type.

EmailReader requires an extensive amount of user interaction and so the service was designed to incorporate an Interactive Voice Response (IVR) system. IVR systems continuously prompt users with voice messages that enable them to navigate and control a particular system easily. In EmailReader it is an essential component for PSTN communication as this is the only way users can interact with services of this nature from their touch-tone telephones. Most IVR systems are application specific and as a result I had to implement one for use in EmailReader.

IVR systems are two-way communication systems. They output voice messages that offer users various options to control a particular service. As their input, they take the choices the users make from the list of supplied options. Usually users indicate their choices by pressing the corresponding number on their terminal keypads. EmailReader requires more than just numeric input, needing alphabetic and punctuation characters as well. For example, users need to be able to enter user names that contain numbers, letters, and punctuation marks. In the following section I introduce the design of the user interface with which EmailReader users can input information and control the IVR system.

6.2.1 User Interface

One of the design challenges of this service was to create a user interface for a technology (email) that normally uses a full computer keyboard and a monitor, in an H.323 terminal as well as a telephone handset. I needed to design an interface that was consistent, easy to use and at the same time powerful.

A major difficulty here was obtaining users' usernames and passwords for authentication. Most mail servers allow user-defined usernames and passwords that consist of alphanumeric strings.

For example, the Rhodes University mail server's usernames are usually of the form g97p5142 (i.e., a letter followed by two digits, followed by another letter and finally another four digits).

EmailReader was developed as an H.323 service available to all possible H.323 terminals. However, because an H.323/ISDN gateway existed within the test network I had envisioned that the service could ultimately be made available to PSTN users as well. This meant that the EmailReader's user interface had to be designed to support the mapping of all the possible characters that may comprise a username and password into a telephone touch-tone sequence. The following implementation attempted to use a simple way of mapping telephone keys to keyboard characters. This mapping is based on the way the keys on most cellular phones are mapped to alphanumeric characters.

The Telephone keys are mapped as follows:

- A lower-case letter is mapped to the following:
    - The touch-tone key where the letter appears. For example: a→2, b→2, d→3.
    - "1", "2","3" or "4" are pressed to differentiate among the letters that appear in each key.

    For example: "21" will be used for "a", "22" for "b", and "23" for "c".
- An upper-case letter is represented using the first two keys used for the lower-case equivalent, followed by a "1", which will represent upper case.
- Since punctuation can be used for passwords, "." is represented by "*".
- A digit in the password is mapped to the same digit.
- All characters, numbers, etc. are terminated by the * key.

- Finally, the end of a sequence representing an alphanumeric string is terminated by the # key (i.e., usernames and passwords must be terminated with a #).

Table 6.1 summarises the mapping from keyboard characters to touch-tone telephone keys in the EmailReader system.

1	1*	m	61*	I	431*
2	2*	n	62*	J	511*
3	3*	o	63*	K	521*
4	4*	p	71*	L	531*
5	5*	q	72*	M	611*
6	6*	r	73*	N	621*
7	7*	s	74*	O	631*
8	8*	t	81*	P	711*
9	9*	u	82*	Q	721*
0	0*	v	83*	R	731*
a	21*	w	91*	S	741*
b	22*	x	92*	T	811*
c	23*	y	93*	U	821*
d	31*	z	94*	V	831*
e	32*	A	211*	W	911*
f	33*	B	221*	X	921*
g	41*	C	231*	Y	931*
h	42*	D	311*	Z	941*
i	43*	E	321*	.	**
j	51*	F	331*		
k	52*	G	411*		
l	53*	H	421*		

Table 6.1 EmailReader IVR Interaction

For example, a student with username g97p5142 would enter:

41* 9* 7* 71* 5* 1* 4* 2* #

g 9 7 p 5 1 4 2

Users' passwords can be entered in a similar manner.

6.2.2 Security

The security design goal of EmailReader is to allow users to access their own messages and to prevent anyone else from accessing them. Since mail message security is provided by the users' usernames and passwords, these must be entered at the beginning of each email-by-phone session for message retrieval from the mail server. Users are allowed three attempts to log-in, after which the system will automatically disconnect them. This is meant to prevent attempts at password guessing. When an invalid username/password pair is entered the system does not release any information as to whether the username or password was incorrect. It simply states that an invalid authentication pair was entered. Within H.323 networks audio streams can also be encrypted for security purposes. Audio stream encryption can be negotiated at call setup time using ITU's H.235 Recommendation, *Security and Encryption for H-series (H.323 and other H.245-based) Multimedia Terminals*. This prevents RTP packets from being sniffed off the network, decoded and listened to by malicious users, essentially releasing EmailReader user's personal email information [19].

With EmailReader's extension to the PSTN, there is potentially a similar problem of somebody listening to users' phone lines and thereby compromising the privacy of their mail messages. No encryption can be done for the communication in this media, since no decryption tool is available at the PSTN end terminals (telephones).

6.3 EmailReader System Operation

Once users initiate a call with the EmailReader terminal they are prompted by the IVR system for their usernames and passwords for authentication purposes. If the authentication with the IMAP mail server is successful, their mailboxes are queried for all unread messages. The headers of these messages are then downloaded using IMAP. Often the headers contain substantial information that is not essential to the user and also difficult for the speech synthesis system to read back correctly. Headers are therefore parsed to create stripped-down versions, containing only the *From* and *Subject* fields. The stripped headers are synthesized into voice messages using *Festival* and stored in audio files on EmailReader's local disk.

The audio file is played to the user through the H.323 RTP (Real-Time Protocol) channels. At this stage users indicate which of the available messages they would like to have read. After entering a valid message number the system will download the full message, synthesize it, and play it to the user in a manner similar to the playing of the headers.

At any point, users can either delete or save the message. A saved message remains in the mailbox, marked as read (many state flags exist within the IMAP protocol, of which *read* is one). A deleted message is flagged for deletion on the mail server, where it is later purged from the user's mailbox.

When a user hangs up, the system will remove all downloaded text files and synthesized audio files associated with the user in order to protect privacy. Finally, the connection to the IMAP server is closed, and the usual processes of H.323 call termination are conducted.

6.4 System Components

The EmailReader system is comprised of 4 major components:

- H.323 API
- IMAP API
- Text-to-speech API
- Interactive Voice Response (IVR) system.

The components mentioned above are modules of software that reside within the EmailReader terminal application and are responsible for its operation. Together they make up the H.323 EmailReader terminal, which can be called and used by any H.323 devices that have access to our H.323 network, including PSTN users via the H.323/SDN gateway. This requires that such terminals have admission rights from our gatekeeper, which administers the network.

In addition to the software components, there is a hardware component that exists within our H.323 version-4 test network. The H.323/ISDN gateway facilitates the use of EmailReader by public PSTN and GSM terminals. I included a description of this gateway and its operation in section 6.4.5.

6.4.1 H.323 API

EmailReader was developed as a complete H.323 terminal using the H.323 protocol stack API provided by the OpenH323 Project (see Chapter 4). Thus the H.323 component of EmailReader functions as the core of the terminal application. The other software modules (IMAP, TTS, and IVR), were integrated into it by wrapping them within specialised PWLib classes.

6.4.2 IMAP API

The IMAP API used in EmailReader is an open-source implementation of the protocol called *C-Client.* Integrating *C-Client* into EmailReader enables the service to access, query, and download messages from any IMAP mail server. The protocol API is also used for numerous other control operations of the IMAP mail server (such as marking specific messages as deleted, read, or unread).

C-Client was integrated into the H.323 EmailReader terminal application by wrapping it within a PWLib class, which I named *IMAPTalker* (see section 6.5 for greater details).

6.4.3 Text-to-Speech API

The Text-to-Speech (TTS) API I chose was developed at the University of Edinburgh's Centre for Speech Technology Research (CSTR) and is formally referred to as *Festival* [3]. It is used to synthesize voice messages from email messages, downloaded from the mail server.

Festival is written in C++ and was easily integrated into the H.323 EmailReader terminal application.

6.4.4 IVR System

I implemented my own Interactive Voice Response (IVR) system within the EmailReader terminal in order to control the systems user interaction component. The IVR system prompts users for input, guiding them through the service quickly and easily. The IVR system is an important component that provides:

- The user interface into the system;
- Control of the back-end functionality (embedded intelligence) of the EmailReader terminal application.

At each point of users' interaction with the system they are given limited options to choose from. For example, before retrieving any messages they must pass the authentication procedure. EmailReader is state-driven and therefore lends itself for modelling as a finite state machine. For example, if the machine is playing a message it is in *playing* state. This means that the system expects the user to either save or delete the currently playing message; no other action is permitted. Three high-level states were identified:

1. Authenticating
2. Playing
3. Control

Of course these states have numerous sub-states that perform operations for the high-level states. For instance, in the authenticating state there is a sub-state that controls the input of users' usernames and passwords, using the user interface introduced earlier.

When in the authenticating state the system gathers the user's username and password. This state also includes the procedure of authentication with the IMAP mail server. Successful authentication will result in the machine proceeding to the control state. In the control state the system reads the unread message headers to the user. It continuously loops, presenting users

with the total number of messages and the number of unread items in their mailbox. Each message number is numerically tagged enabling users to identify the message they want read by pressing the number using DTMF. The IVR system waits for the DTMF information identifying a specific message and then proceeds to download it. In the control state, the IVR system allows users to do one of the following:

1. Jump to a specific message number by typing the numeric message number followed by the # key.

2. Play the next unread message by pressing the # key.

3. Play the previous unread message by pressing the * key.

If successful, the system proceeds to the playing state and begins to read the selected message body to the user. At this stage the system only accepts one of two options:

1. Save the current message by pressing 9.

2. Delete the current message by pressing 7.

If successful, the system will return to the control state and await indication of the next message to be played.

Obviously, the IVR system plays a vital role in the control of this service. It acts as an operator by accepting commands from users (in the form of DTMF tones) and by putting the EmailReader system into the appropriate state based on each user's input.

6.4.5 H.323/ISDN Gateway

The H.323/ISDN gateway is an open-source application developed by the University of Carlos, Spain [47]. It provides connectivity to the PSTN via a number of ISDN interfaces. As such, it is an H.323 component of our test network, enabling PSTN and GSM telephone users to access the EmailReader service. Essentially the gateway extends the H.323 user base of EmailReader to

include users of the public telephone networks (PSTN and GSM), the 'natural' target networks of this service.

The gateway acts as a switchboard operator when PSTN users call the service. Next, users are prompted to dial an extension number, using touch-tones (DTMF), denoting the desired service (in this case, EmailReader). With the help of our H.323 gatekeeper, the gateway will forward the call to the EmailReader service on the H.323 network.

6.5 Implementation of the H.323 EmailReader Terminal

As mentioned, EmailReader is implemented as a non-signalling service that is in turn implemented within a callable H.323 terminal. According to the H.323 service categorisation, introduced in Chapter 3, EmailReader can be classified as a distributed service as it does not require the intervention of central H.323 network entities, such as a gatekeeper, for its operation. Its full classification is as a non-signalling, distributed, proprietary (implemented by us and not standardised) service.

This section describes the ways in which a basic H.323 terminal was adapted to perform its function as an H.323 EmailReader terminal. I do not include a complete description of how EmailReader was implemented, but instead concentrate on some of the difficulties that arose during its implementation. Among the many programming issues that did arise during implementation of EmailReader, I discuss a subset specifically related to H.323.

6.5.1 EmailReader Software Architecture

Figure 6.2, shown on page 86, illustrates a simplified entity-relationship (ER) diagram of the EmailReader application. The root application class of this service is the *EmailReaderEndpoint* class that derives from the OpenH323 API *H323EndPoint* class. This class is responsible for the initialisation of all H.323 terminal procedures, for example building the capabilities table for EmailReader and starting the Listener classes that listen for incoming connections. In this class I initialised the TTS, IMAP, and IVR systems. This ensures that initialisation of these relatively resource-intensive modules are performed on a per application basis rather than a per connection basis.

When the OpenH323 API *H.323Listener* class detects an incoming connection from a user terminal, it spawns an instance of the EmailReader H.323 Connection class (*ERH323Connection*), an EmailReader class which is a specialisation of the OpenH323 *H323Connection* class.

The ERH323Connection class houses most of the logic required for EmailReader's operation. It is in this class that all H.225 and H.245 signalling and control PDUs are handled. In addition, it maintains state information regarding the system's operational state, it controls the interaction with the IMAP mail server on a per user basis using the *IMAPTalker* class, and it provides the IVR functionality through the use of OutGoingMessageChannel (*OGMChannel* class). The *OGMChannel* class is my extension of the OpenH323 API *PIndirectChannel* class that is responsible for preparing the audio stream for transmission using RTP. The *PIndirectChannel* class usually sources the audio from the microphone, but my extension (*OGMChannel*) enables the audio to be sourced from audio files instead.

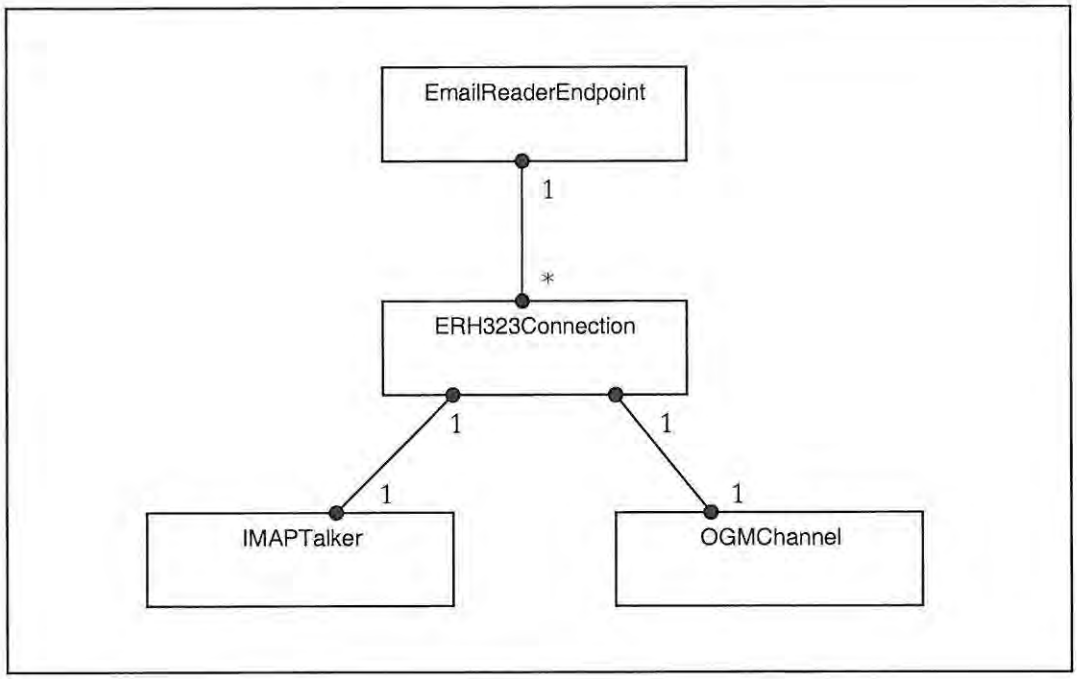

Figure 6.2 EmailReader's Software Architecture

ITU Recommendation H.323 [25] states that an H.323 terminal "is an endpoint on the network which provides for real-time, two-way communications with another H.323 terminal, gateway or MCU." Usually the real-time communication between two H.323 terminals originates from one user's microphone (audio) and/or video camera (video) and terminates at the remote user's speakers (audio) and/or display device (video). EmailReader differs in two ways.

First, the TTS-generated audio is not sourced from a microphone but from a text file that is converted into an audio file via the TTS system. Because all audio from EmailReader is stored in audio files on the system it was necessary to pipe audio from these files directly into the audio channel between the user and EmailReader service audio channels.

Secondly, there is no need for audio to be transported from the user to the EmailReader service terminal. This service is a one-way audio communication between EmailReader and the connected user.

6.5.2 Audio Channels

All standard H.323 terminals are required to open both audio channels (one receive and one transmit) during call establishment. This means that in order for the H.323 EmailReader terminal to maintain interoperability with other H.323 terminals it must open both audio channels, one receive and one transmit, on call establishment. As mentioned above, there is no need for the transmission of audio from the H.323 user terminal to the EmailReader terminal. This poses a problem since the standard forces us to open both audio channels, yet only one is required and effectively used.

I identified two solutions to this situation:

- Allow the user terminal to maintain both audio channels with EmailReader for the duration of the call. All transmitted audio from the user terminal can be ignored at the EmailReader terminal.

- Once both audio channels have been opened during call establishment, the EmailReader service terminal can close its receive channel and hence the user terminal's transmit channel.

Although both solutions have advantages and disadvantages I employed the second. Leaving both audio channels open (first option) would allow all DMTF information, used to control the IVR system, to be transported in-band (within the audio stream from user terminal to EmailReader). The drawback here regards bandwidth waste as the audio stream (RTP Packets) would be continuously transmitted across the network to the EmailReader terminal, only to be read in and then discarded. The ratio of DTMF tone to audio information is negligible, and leaving the channel open for DTMF transport appears a poor idea. As discussed in Chapter 3 (section 3.4), there are numerous ways of carrying DTMF information within H.323 networks, and using the audio channel to send DTMF in-band is not the preferred choice. The procedure of closing a single audio channel can be easily carried out, in a protocol compliant manner, using the H.245 media control channel.

6.5.3 Sourcing Audio from Files

Sourcing the audio from files as opposed to the microphone is a requirement in the implementation of EmailReader. While considering the OpenH323 API and its architecture (see Chapter 4), I noticed that each audio channel is associated with a particular codec, responsible for the encoding and decoding the audio in an H.323 application. In a standard H.323 terminal the audio is sourced from the user's microphone using the *H323Channel* class and presented to the *H323Codec* class as a raw audio stream. This raw audio is usually sampled at 8 KHz with each sample being 16 bits (signed) in length.

In EmailReader it was necessary to extend the OpenH323 *H323Channel* class to redirect the audio input from the microphone to the audio files stored on the local file system. In addition, I needed to ensure that the audio files were in the same raw PCM format (in this case, signed 16 bit audio samples at 8 KHz) expected by the *H323Codec* class. The encoding function of the *H323Codec* class takes as its input the raw audio from the *H323Channel* class and encodes it according to the codec algorithm agreed upon during the capabilities negotiation procedures of H.323 call establishment (e.g., G.723.1). The encoded audio stream is finally presented to the *RTPChannel* class responsible for packaging it into RTP packets and sending it to the H.323 user terminal via a UDP connection.

One problem I encountered was that the files generated by *Festival* were in WAV format and not the required raw format. These WAV audio files had to be transcoded into the raw equivalent before being made available to the *H323Channel* class. This was done using the popular Linux *SoX (Sound eXchange)* program from within the EmailReader terminal application. Executing a Linux shell application from within another application can be performed using the Linux C++ *system* function, by passing it the program name (e.g., sox) and arguments as standard C++ character arrays. For example, the following Linux shell command will convert a WAV file called mywav to an 8 KHz, 16 bit, signed audio file equivalent:

```
sox myWavFile.wav -r 8100 -s -w myRawFile.sw
```

With the file in raw audio format, it can be presented to the *H323Channel* class, where the audio stream is read and passed to the *H323Codec* class for encoding and finally transmission to the user terminal via the RTP audio channel. Once at the user terminal the audio stream can be extracted and decoded using the *H323Codec* class and presented to the user.

6.5.4 DTMF Control

EmailReader is controlled entirely using DTMF tones. These tones are transported within H.245 media control messages. As discussed in section 3.4, DTMF tones in H.323 networks can be transported in a number of ways. In our test network they are transported out-of-band within the H.245 *UserIndication* message of the H.245 control channel.

According to Annex B.14.6 of ITU Recommendation H.245 [23], user input messages are defined as a string of characters coded according to ITU Recommendation T.51 [20], which can be used for key-pad input, an equivalent of DTMF.

My intention was to transport the DTMF tones through the H.323 network in a reliable and protocol-compliant manner. Essentially, out-of-band DTMF tones are sent within the *UserInputIndication* message of H.245, the H.323 call control protocol. Appendix B.2 illustrates the ASN.1 notation for H.245 Indication messages and more specifically the `UserInputIndication` ASN.1 specification. Examination of this specification reveals that there is enough information to fully support the transmission of DTMF information in H.323 networks. For example, a field called `duration` exists in the specification and is used to indicate the duration of the tone (i.e., the length of time the user held the particular tone on their terminal).

Here I have noted the issue of DTMF transport on H.323 networks. At the time of EmailReader's implementation the H.323 test network featured an H.323/ISDN gateway. This gateway bridges the public PSTN and GSM voice networks with our H.323 network. Therefore, I implemented EmailReader with a vision that the service would ultimately be available to the H.323 network, in which the service would reside, as well as the public networks PSTN and GSM, via the gateway. This affected the manner in which I implemented EmailReader,

including its user interface and support for DTMF transmission. The following section addresses how PSTN users can interact with the EmailReader service.

6.6 Service Availability to PSTN Terminals

Use of the EmailReader service is dependent on the ability to interact with it using DTMF, and more specifically using out-band H.245 DTMF transmission. Traditional PSTN telephones are relatively 'dumb' terminals only capable of interacting with their network and other terminals using DTMF tones. This means that DTMF is supported on the PSTN, but it is transported differently in the PSTN as opposed to the H.323 network: PSTN DTMF is transported within the audio channels (bearer channels) and is referred to as in-band DTMF.

Interaction with EmailReader requires the PSTN DTMF information to be extracted from the audio stream at the H.323/ISDN gateway. DTMF tone extraction from an audio stream involves complex mathematical algorithms. Fortunately, such functionality existed within the ISDN/H.323 gateway I had at my disposal (actually supported by the underlying ISDN protocol API used by the gateway, called ISDN4Linux [33]). The DTMF tones extracted from the PSTN audio bearer channels had to be forwarded to the EmailReader terminal. Therefore I had to package the tones and send them to the EmailReader terminal.

When PSTN users use the EmailReader service, the H.323/ISDN gateway sits between the user's terminal (telephone or cellular) and the EmailReader service terminal. As far as the EmailReader service terminal is concerned, it is in an H.323 call with another H.323 terminal (the gateway). DTMF information can therefore be injected into the H.245 control channel using the H.245 UserIndication messages introduced earlier.

Figure 6.3 illustrates the process of converting the in-band DTMF from the PSTN to out-of-band DTMF on the H.323 network. Extracted DTMF tones are injected into H.245 PDUs and sent over the H.323 network to the awaiting EmailReader service terminal.

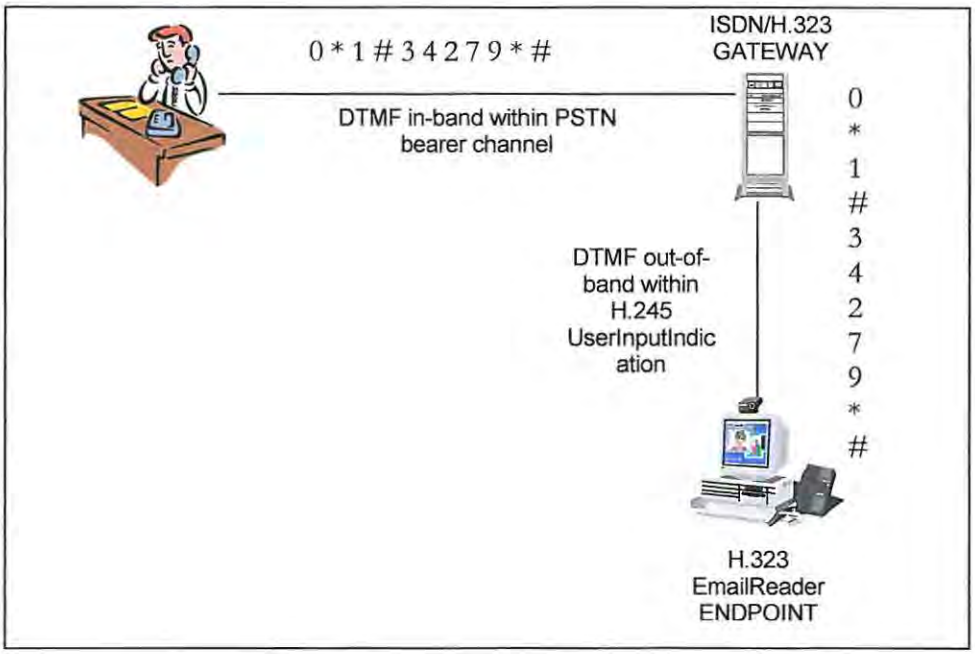

Figure 6.3 In-band DTMF to Out-of-band DTMF Conversion

These procedures ensure that PSTN users can use the EmailReader service transparently, masquerading as standard H.323 terminal users by means of the H.323/ISDN gateway. As far as the PSTN users are concerned, the EmailReader service is located within the PSTN service layer; and as far as the EmailReader service terminal is concerned, it is in an H.323 call with another H.323 terminal (in this case the H.323/ISDN gateway).

6.7 Summary

The EmailReader service is an H.323 non-signalling service developed for the research in an effort to empirically explore H.323 non-signalling service creation. The chapter describes EmailReader's functionality, its components, and its implementation as an H.323 non-signalling

service. Several important issues were discussed, such as the design of EmailReader's user interface for easy use by PSTN and GSM telephones; the software architecture of EmailReader; description of the IVR system developed for use in EmailReader; and the integration of enabling computer technologies, like TTS and IMAP, into the EmailReader service application.

CHAPTER 7

H.323 SIGNALLING SERVICE CREATION

"Knowledge is more than equivalent to force."

Rasselas

This chapter covers the signalling service I have created called CANS. CANS is a notification service that allows H.323 users to setup and customise notification calls from their CANS-enabled H.323 terminals. I begin with a description of the design of CANS, including its basic operation and user interface. To implement CANS as a signalling service I had to choose which H.323 signalling service creation mechanisms (H.450 or H.460) to use. This decision is the focus of section 7.3. The CANS service is made up of two components: the CANS client and the CANS server. Section 7.4 describes these components and their operation. Additionally, I describe two gateways that enable notification calls to be made to the PSTN and GSM networks. Actual implementation of CANS is described in section 7.5. This section discusses the implementation of CANS as an H.450 service, its feature negotiation using H.460, and service management using H.323 Annex K. Finally, the possibility of extending the CANS service to PSTN terminals for notification setup is addressed.

7.1 Introduction

The Customisable Alarm Notification Service (CANS) is a complete H.323 signalling service, newly developed in an effort to explore and demonstrate the use of the H.323 version-4 service creation and extension mechanisms [43]. The mechanisms I used to develop CANS include the H.450 Supplementary Service (SS) Framework, H.460 Generic Extensibility Framework (GEF) and Annex K (HTTP Based Service Control Transport Channel), all of which were introduced in Chapter 5.

CANS is discussed here as a signalling service although a certain amount of embedded intelligence (typical in non-signalling services) does exist within the system. This embedded intelligence exists within CANS in the form of text-to-speech (TTS) technology, similar to EmailReader. This does not mean that CANS is a non-signalling service, but rather a signalling service that requires non-signalling backend logic.

Basically, the service enables users (clients) to establish an H.323 call with a CANS server. Once connected, users can set a date and time for the service to call them (*notification call*). In addition they can include the identification of the device on which to be contacted, as well as a text or voice description of the event they wish to be reminded of. The text or voice description is presented to the user on establishment of the notification call.

Notification calls can be initiated to any device on which the user of the service can be reached via our H.323 test network. This test network supports the following contactable devices:

- PC Soft Phone (such as NetMeeting on Windows, GnomeMeeting on Linux)
- Compaq IPAQ PocketPC (via a 802.11b wireless extension of our LAN)
- PBX phones (via H.323/ISDN gateway)
- Telkom land lines (via H.323/ISDN gateway)

- Cellular telephones (voice via H.323/ISDN gateway and/or SMS via SOAP GSM SMS gateway).

The last item requires clarification. Users' notification reminders can be presented to them as voice messages or as SMS text messages on their cellular phones. Prior to initiation of the notification call, the system identifies which message is to be presented to the user. If a voice message is to be presented, the system initiates a call via the H.323/ISDN gateway to the user's cellular phone. If a text message is to be presented then the system sends it as an SMS, via a SOAP GSM SMS gateway. The H.323/ISDN gateway is an H.323 component within our test network (see Chapter 4), while the SOAP GSM SMS gateway is not an H.323 component but rather a service on our LAN that can be used by any application supporting the Simple Object Access Protocol (SOAP). A description of this particular gateway is given in section 7.4.4.

In certain instances users may supply the system with only a text message. In this case the system automatically synthesizes a voice message equivalent using TTS. This ensures that both voice and text messages are available for notification calls. If users leave only a voice message then the system is restricted to initiating a notification call to voice devices only. This is because speech-to-text (STT) systems are still not of a high enough standard to be used to create a text message equivalent.

7.2 Design

CANS is a client-server application and consists of the CANS client and CANS server terminals. CANS' operation may be broken into two distinct phases, the *notification setup* (NS) phase and the *notification call* (NC) phase.

The NS phase involves users utilising their CANS client terminal to set up and administer notification events. In this phase users interact with the system via the client terminal's user interface. The NS phase ends when a user commits the information entered into the client terminal, resulting in its transfer to the CANS server. This information is then stored within the

CANS server until the NC phase. The NC phase begins with the initiation of a notification call to a user's device and ends with successful delivery of the reminder.

7.2.1 Basic System Operation

Operation of CANS is perhaps best introduced by way of example. Consider Bob, a user who would like to use the service to be reminded of his wife's birthday. Bob makes use of his CANS client terminal application in which he supplies his authentication details (username and password), his callback number (telephone number for PSTN terminals, IP address for H.323 terminals, etc.), and a notification message (system supports both text and voice notifications). The system also allows users to optionally provide information of a particular gateway to use. For example, the system is currently configured with two automatic gateways: the H.323/ISDN gateway for initiating voice calls to the PSTN and GSM networks, and a SOAP GSM SMS gateway for sending SMS notifications. If a particular user wants to be contacted via her H.320 system on the ISDN network then she can provide the CANS system with details of the particular H.323-H.320 gateway to use for the notification call.

Once Bob is happy with the information he has supplied to his CANS terminal he can instruct it to commit the data. The CANS client does this by sending all Bob's information to the CANS server. The server processes the information beginning with Bob's authentication details. If his authentication is successful the server will store the details for the notification call.

Bob's authentication will fail if:

- he is not registered with the CANS service, or
- he supplies an invalid username/password pair.

Such failed authentication will result in an error response from the CANS server to the CANS client terminal indicating the problem to Bob. If Bob is not registered, the CANS server will follow this error response message with another message that will instruct Bob's terminal to open an HTTP connection to a specified URL where BOB can register himself as a valid CANS user. Once registered, Bob may continue to use the service as normal.

Let's assume that Bob's authentication was successful and that his notification information has been successfully stored within the CANS server. During the period between the end of the NS phase and the beginning of the NC phase, Bob may want to edit the notification he has already set up. In this case the system allows Bob to use a standard HTTP connection (web browser) to administer his notification details. In addition to being able to edit the date, time and callback number the user may:

- Delete the notification request;
- Add alternate callback numbers that can be used for the notification call should the primary device fail;
- Prioritise the devices in terms of callback choices. The CANS server will attempt the notification call to the device with the highest priority. Failure to connect to this device will result in an attempt to connect to the device with the next highest priority.

I identified the interim period (between the NS and NC phases) as a sub-phase and called it the *service management* (SM) phase. It is important to note that CANS could easily have been implemented as a web service. However, one aim of this is to investigate H.323 service creation, and, in this particular instance, using the service creation mechanisms of H.323 version 4. In this regard I have implemented the CANS service and made it customisable via an HTTP interface. This allows users to customise their notification calls using a standard web browser, or directly from their CANS-enabled H.323 terminals, using the H.323 Annex K functionality.

At the moment the system must call Bob and present his notification message (NC phase), the CANS server will attempt to establish the notification call through a connection to the device with the highest priority (if Bob has setup more than one device for possible reception of the notification call). If one device is busy or is not answered, the server will try a lower-priority device. This continues until no specified devices are left with which to establish the notification call, or until one is answered. If all devices available for the notification call are unable to accept a call, the system checks if the user has provided:

- a text message for his notification, and
- the number of a device capable of receiving an SMS.

If both conditions were satisfied in Bob's case, then he will be sent an SMS of his notification via the SOAP GSM SMS gateway. As STT systems improve, the first condition above will become unnecessary because a text message could be created using a user's voice message. Nevertheless, SMS notifications are asynchronous and the system will assume that Bob will receive notification by SMS at some time, resulting in a successful notification event. A failed notification occurs if the system fails to alert Bob of his notification.

7.2.2 User Interface

A significant amount of user information is required for the NS phase of CANS. This information is input via the graphical user interface (GUI) of the CANS client terminal. Figures 7.1 to 7.4 illustrate the user interaction interfaces of the CANS client terminal application.

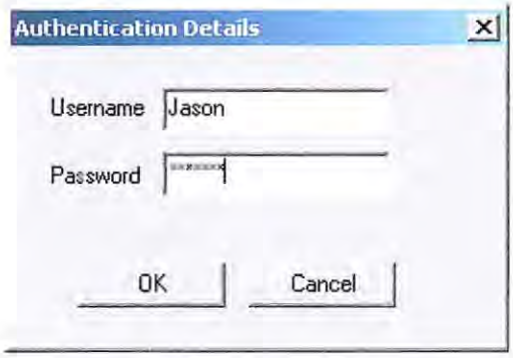

Figure 7.1 CANS Authentication

Figure 7.1 illustrates the input dialog for the user's authentication details. The user is not authenticated locally on the H.323 CANS client but rather on the CANS server (the authentication details are stored within the CANS client and sent with the notification information to the CANS server). This ensures that the user registers with the central CANS

server. It also means that a CANS registered user may use the service from any H.323 terminal that supports the CANS service.

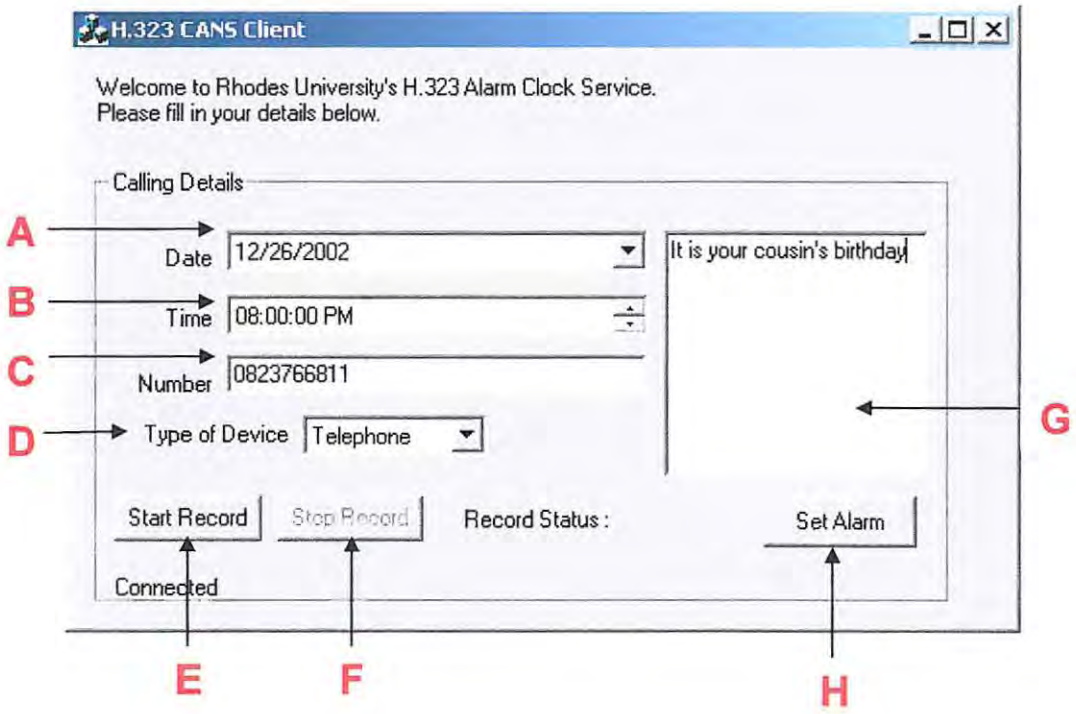

Figure 7.2 CANS H.323 User Terminal

Figure 7.2 illustrates the CANS client's main user interface that is presented following entry of the user's authentication details. Letters A through H identify the interactive components of the interface:

A. Date that the notification call will be made;

B. Time that the notification call will be made;

C. Number – primary device on which to initiate the notification call;

D. Type of device – options are Telephone, SMS, or SoftPhone;

E. Start Record – button to begin recording voice notification message;

F. Stop Record – button to end recording of voice notification message;

G. Text field in which to write text notification message;

H. Set Alarm – Transfer information to CANS server for processing;

Once a user has entered all the notification details for a notification call, the *Set Alarm* button (H) is pressed to send the information to the CANS server. At this point the CANS server can request that the CANS client to open an HTTP connection to it, allowing the user to administer the service. An HTTP open recommendation is sent to the CANS client if:

- the user is not registered with the CANS service, or
- the user is registered and the notification call has been set up successfully .

Figure 7.3 illustrates the CANS client's built-in browser that allows the user to register with the CANS server. In this illustration, the CANS client has been instructed by the CANS server to navigate to the service registration page. Here the user can register as a CANS user.

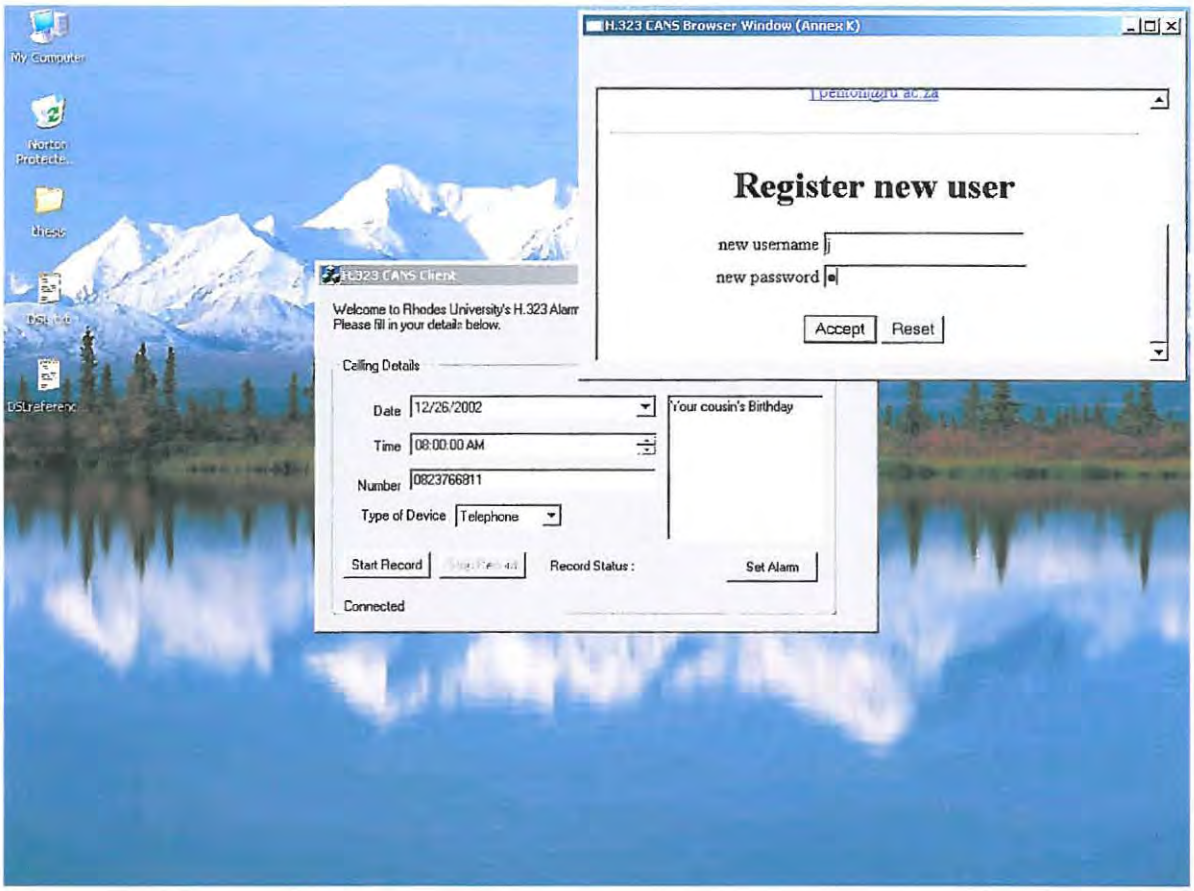

Figure 7.3 CANS User Interface for User Registration

Figure 7.4 illustrates the navigation of the CANS client's browser to the CANS server's notification call customisation pages. Here the user can add and prioritise the devices to which the notification will be sent. In this illustration, the user has indicated two devices for the notification call: A telephone with the number 0823766811 given the highest priority, and an SMS device with the same number (cellular) of lower priority. The CANS server will attempt to establish a notification call with the telephone (0823766811). Failure of this notification call will result in an SMS of the notification being sent to the SMS device 0823766811.

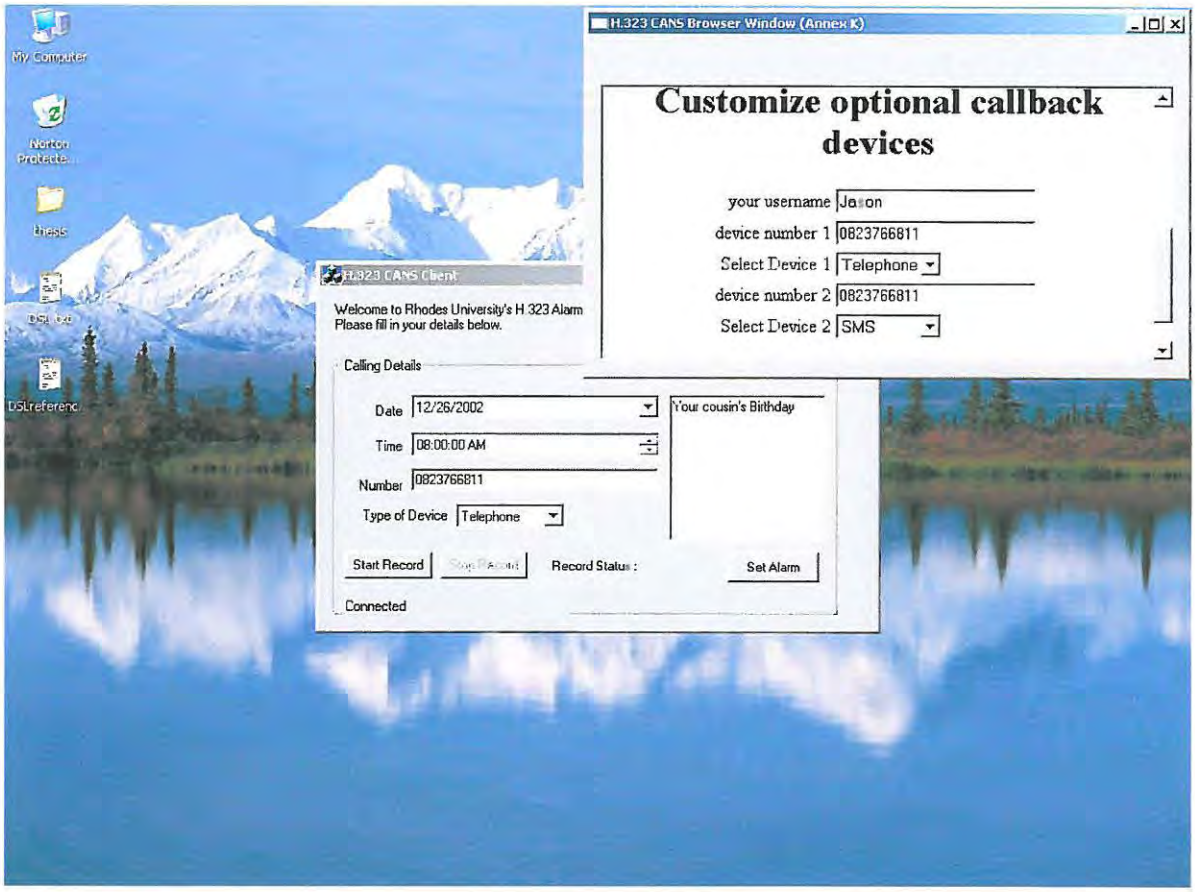

Figure 7.4 CANS User Interface for Customising Callback Devices

7.3 Choosing H.450 or H.460

The simplest way to build the CANS service would be to add the necessary intelligence into a standard H.323 endpoint (i.e., a non-signalling service). In this case the user would be required to interact with the system using Dual-tone Multi-Frequency (DTMF), in the same manner as the EmailReader service discussed in Chapter 6. Although this is a viable approach, my intentions with this service (CANS) were to investigate signalling service creation. Thus CANS has been implemented primarily as a signalling service with all the service information transported within H.225 messages.

In order to implement CANS as signalling service, it was necessary to decide which H.323 service mechanisms to use. Two H.323 mechanisms introduced in Chapter 5 are the H.450 Supplementary Service (SS) and the H.460 Generic Extensibility Framework (GEF). Both H.450 and H.460 can be used for service creation while H.460 can be used for service negotiation as well. To choose a mechanism to use for the primary implementation of the service I turned to the decision tree presented in Chapter 5 (Figure 5.1). Working through this tree and considering the nature of CANS I decided to use H.450. The attributes that aided the decision are as follows:

- The CANS service is not core to all H.323 applications,
- it has a significant amount of parameters (notification details),
- information is not required to be transported in RAS messages (i.e., no necessary gatekeeper interaction). This is because the CANS service is a point-to-point service that exists between two H.323 terminal devices. Service information is not required to be relayed between the terminals and their controlling gatekeeper. The gatekeeper does however still play its usual role as far as establishment and administration of the call is concerned.

Naturally, I wanted to ensure that H.323 endpoints that do not support the CANS service also would not engage in a call with the CANS server. This constraint was neatly enforced using the feature negotiation mechanism of H.460 (GEF). When the CANS server receives an H.323 call set-up message from an H.323 client, it checks that the supported feature set includes CANS. If not, the server will send an H.323 Release Complete message with the Release Complete reason set to *unsupported feature*. CANS is thus implemented as an H.450 service and negotiated using H.460.

To enable CANS users to control and administer their notifications, I integrated H.323 Annex K into the system. H.323 Annex K provides service management functionality to H.323 version-4 networks. It enables users to interact with a service via a separate HTTP connection from their

H.323 terminal to the provider of the service. There are two ways that a user may interact with the service such as CANS using Annex K.

First, to use CANS a user must register with the system. This can be done directly from the H.323 CANS client using Annex K functionality. Second, users may customize the devices (adding, changing priority, etc.) for their notification calls. Once a notification is successfully set up, the CANS server instructs the CANS client terminal to present users with an URL (e.g., http://IPofCANSServer/customise?username=Bob&servicecallid=1), which allows them to customize the devices on which they can be contacted. The system will attempt to establish the notification call to each device until the notification is successfully supplied to the user, or until no other notification devices are left on which to contact the user (failed notification call).

7.4 System Components

The CANS system consists of the following components:

- CANS Client
- CANS Server
- H.323/ISDN Gateway
- SOAP SMS GSM Gateway.

7.4.1 CANS Client

The CANS client provides the CANS service to an H.323 user. It is essentially a standard H.323 terminal in which I have integrated support for the CANS service. Supporting the CANS service means that the CANS client terminal can communicate with any CANS server.

7.4.2 CANS Server

The CANS server processes all user notification requests originating from an H.323 CANS-enabled terminal. Similar to the CANS client, the CANS server is a standard H.323 terminal that

has CANS support integrated into it. Support for the CANS service enables the server to service all CANS-enabled H.323 clients.

7.4.3 H.323/ISDN Gateway

The H.323/ISDN gateway is used to bridge the H.323 test network to the PSTN. This enables CANS notification calls to terminate on devices accessible via the PSTN (telephones and GSM cellular phones).

7.4.4 SOAP GSM SMS Gateway

The SOAP GMS SMS gateway is used to send SMS notifications to a user's cellular telephone and other SMS-capable devices via the GSM network. This gateway is implemented as a web service and is made accessible via a Simple Object Access Protocol (SOAP) interface. By integrating support for SOAP into the CANS server we could send SMS notifications to terminals on the GSM network. This gateway service was implemented by Guy Halse, one of my colleagues at Rhodes University's Department of Computer Science; further discussion of it can be found in a paper by Halse and Wells [16].

7.5 CANS Implementation

In this section I discuss the practical implementation of CANS. Previously I mentioned that both the CANS client and CANS server terminals were implemented as standard H.323 terminals, using the H.450 SS and H.460 GEF frameworks. I now describe how the CANS service was integrated into these terminals, allowing them to support CANS.

7.5.1 Service Information Transfer using H.450

CANS is modelled on the request/response paradigm. As such, the client initiates a request (e.g., notification setup) to the server, which processes the request and returns a result indicating the outcome of the processing. The CANS client initiates its request once users have entered all the details required for the notification. This information includes:

- User authentication details (username, password pair);

- Device details (number to be called on, type of device);
- The time the notification call must be established;
- Text message for the notification;
- Voice message for the notification.

The following is an extract of the CANS ASN.1 data structure that was written to support the operation of the service. This structure encapsulates the information required for a notification setup and accompanies the CANS `SetNotification` operation (i.e., it is an argument of the `SetNotification` CANS operation). Various CANS operations are discussed later in this section.

```
1       CANSSetNotificationArgument ::= SEQUENCE
2       {   authDetails AuthDetails,
3           deviceDetails DeviceDetails,
4           year Year,
5           month Month,
6           day Day,
7           hour Hour,
8           minute Minute,
9           second Second,
10          txtMessage GeneralString OPTIONAL,
11          messageFormat CHOICE
12          {
13                voice_only NULL,
14                text_only NULL,
15                both NULL,
16          },
17          argumentExtension CHOICE
18          {
19                extensionSeq ExtensionSeq,
20                nonStandardData NonStandardParameter
21          } OPTIONAL,
22      }
```

This structure specifies the notation of the argument `CANSSetNotificationArgument` transported within a CANS H.450 invoke operation from the CANS client to a CANS server. We will return to `authDetails` and `deviceDetails` (lines 2 and 3) later. The time the notification call must be initiated is defined as the types: `Year`, `Month`, `Day`, `Hour`, `Minute`,

Second (lines 4 to 9). All of these user-defined types are of ASN.1 built-in type INTEGER, defined later in the specification.

Line 10 specifies the optional presence of a txtMessage, defined as type GeneralString. This field houses the textual notification message that users may supply.

Next is the field messageFormat (line 11), which is of ASN.1-type CHOICE. This field is used to specify the format of the message supplied by users for their notification calls. Options for this field are:

- voice_only (line 13) for a single voice notification;
- text_only (line 14) for a text notification;
- both (line 15) for a voice and text notification.

The field argumentExtension (lines 17 to 21) is provided for vendor-specific extension of the CANSetNotificationArgument structure.

The fields authDetails and deviceDetails (lines 2 and 3) are defined as user-defined types AuthDetails and DeviceDetails, respectively. These types are specified as follows:

```
1       AuthDetails ::= SEQUENCE
2       {
3           username OCTET STRING,
4           password OCTET STRING
5       }
```

AuthDetails (line 2) is defined as a SEQUENCE of the user's username followed by the corresponding password.

DeviceDetails (line 3) is a slightly more complex structure, and consists of the field contactNumbers, which is defined as a SEQUENCE OF (array) of DeviceNumbers:

```
1   DeviceDetails ::= SEQUENCE
2   {
3      contactNumbers SEQUENCE OF DeviceNumbers,
4      ...
5   }

6   DeviceNumbers ::= SEQUENCE
7      number OCTET STRING (SIZE(1..100)),
8      requiresGW BOOLEAN,
9      gatewayInfo GatewayInfo OPTIONAL
10  }
```

Referring to the extract above, `DeviceNumbers` is a SEQUENCE of the device number (line 7), whether or not the device requires an access gateway from the H.323 network (line 8), as well as the optional gateway information (line 9).

The number field (line 7) is defied as an OCTET STRING that can be used to identify all possible devices on which a particular CANS notification call can terminate. These identifiers include telephone numbers, IP addresses, and DNS names.

The `requiresGW` (line 8) is a BOOLEAN field that can either be TRUE or FALSE, indicating whether or not a gateway is required to access the corresponding device.

The optional field `gatewayInfo` is of user-defined type `GatewayInfo`, as specified below:

```
1       GatewayInfo ::= SEQUENCE
2       {
3       ipaddress OCTET STRING (SIZE(1..100)) OPTIONAL,
4       gatewayIdentifier CHOICE
5       {
6                 sMS NULL,
7                 voice NULL,
8                 ...
9       }
10      }
```

The structure consists of the `ipaddress` field, which represents the IP address of the gateway to be used by the CANS server on initiation of a notification call. This is followed by the field `gatewayIdentifier`, defined as an ASN.1 CHOICE with the following options:

- `sMS` (line 6) representing an SMS gateway
- `voice` (line 7) representing a voice gateway
- `...` (line 8) – the ASN.1 extension marker catering for future gateways.

The structure `CANSSetNotificationArg` used to carry CANS notification setup information from the client to the server. This information is transported within an X.880 ROS Invoke operation. The operation `setNotification` is one of six possible operations defined by CANS. All possible CANS operations are specified in the following ASN.1 structure:

```
1       CANSOperation ::= ENUMERATED
2       {
3           setNotification (20),
4           deleteNotification (21),
5           NotificationIsSet (22),
6           NotificationNotSet (23),
7           startRecord (24),
8           stopRecord (25),
9       }
```

The operation `CANSOperation` is an enumeration of the six CANS operations. Each operation is tagged with a number in brackets used to identify the operation within the scope of the CANS service. For example, the operation `setNotification` (line 3), discussed in detail above, is an operation initiated by the CANS client on the CANS server indicating that a user has requested a notification setup.

The operations `startRecord` and `stopRecord` are used to instruct the CANS server to open or close its audio channels with the connected CANS client. These operations are used for the recording of users' voice message for their notification calls, and map to the user interface buttons "Start Record" and "Stop Record" in Figure 7.2 (components E and F, respectively).

There are a number of errors that may occur in the service. For example, the information supplied by a user for a notification setup may include a time for the notification call that has already expired. In this case the CANS server terminates its processing of the supplied information and returns an X.880 ROS ReturnError indicating the error. The various CANS errors are defined as:

```
1       CANSErrors ::= ENUMERATED
2       {
3           unspecified (1),
4           invalidDay (2),
5           invalidMonth (3),
6           invalidYear (4),
7           invalidHour (5),
8           invalidMinute (6),
9           invalidSecond (7),
10          invalidDate (8),
11          failedAuthentication(9),
12          unregisteredUser(10)
13      }
```

On receipt of an error the CANS client alerts its user and may take appropriate action. If no processing errors occur and the CANS server successfully completes a notification setup, an X.880 ROS ReturnResult message is sent from the CANS server to the client, as opposed to a ReturnError in the previous case. Accompanying this message is the structure `CANSSetNotifcationResult`, defined as:

```
1       CANSSetNotificationResult ::= CHOICE
2       {
3           notificationSet NotificationIsSet,
4           notificationNotSet NotificationNotSet
5       }


6       NotificationIsSet ::= SEQUENCE
7       {
8           date GeneralizedTime
9       }

10      NotificationNotSet ::= SEQUENCE
11      {
12          reason GeneralString
13      }
```

The CANSSetNotificationResult leads to one of two options. If the notification setup was successful, NotificationIsSet is used; this returns the date and time of the requested notification to the CANS client for final acknowledgement by the user. In the case that the notification was not set, the structure NotificationNotSet is used, which returns a string to the CANS client indicating why the notification setup was unsuccessful. For example, if the server was scheduled to go down for maintenance on the day the user is requesting a notification, then it would use the structure NotificationNotSet to inform the user of such an event.

7.5.2 Service Negotiation using H.460

Features and services are negotiated in H.323 using H.460. Negotiation is required to ensure that H.323 terminals can specify the services they require, support, and would like to use during an H.323 call. H.460 specifies the way in which a service is identified in the negotiation procedure as well as the procedures required to conduct the negotiations. H.460 was described in section 2.4.2.

Using H.460, services are described using the three feature sets (needed, desired and supported) that were introduced in Chapter 2. These feature sets are transported within the type FeatureSet of the H.225 ASN.1 specification:

```
FeatureSet ::= SEQUENCE
{
     replacementFeatureSet    BOOLEAN,
     neededFeatures           SEQUENCE OF FeatureDescriptor OPTIONAL,
     desiredFeatures          SEQUENCE OF FeatureDescriptor OPTIONAL,
     supportedFeatures        SEQUENCE OF FeatureDescriptor OPTIONAL,
     ...
}
```

Each feature set is defined as an array (SEQUENCE OF) FeatureDescriptor types, which is in turn defined as type GenericData:

```
FeatureDescriptor  ::= GenericData
```

Let us reintroduce the structure GenericData, previously described in Chapter 5. The type GenericData is defined as a SEQUENCE of the feature's identifier followed by an optional array of feature parameters that are encoded as EnumeratedParameter types:

```
GenericData ::= SEQUENCE
{
     id                  GenericIdentifier,
     parameters          SEQUENCE (SIZE (1..512)) OF EnumeratedParameter
     OPTIONAL,
     ...
}
```

The two fields within this structure are id and parameters. The field parameters is optional (indicated by the ASN.1 OPTIONAL identifier) and is only required when implementing a service using H.460. Because we rather want to identify a service, we use the field id of the structure GenericData. This field is defined as type GenericIdentifier.

```
GenericIdentifier ::= CHOICE
{
     standard            INTEGER(0..16383,...),
     oid                  OBJECT IDENTIFIER,
     non-Standard        GloballyUniqueID,
     ...
}
```

Above, we see that the type GenericIdentifier can be one of three choices. It can be defined as an INTEGER for identifying standard H.460 features (e.g., H.460.2 will have a value of 2), an OBJECT IDENTIFIER for a standard service not ratified by the ITU, or as GloballyUniqueID for identifying proprietary services.

CANS was developed as a proprietary service which limits its method of H.460 identification to the use of the fields oid (OBJECT IDENTIFIER) or non-standard (GloballyUniqueID) of the structure GenericIdentifier.

Use of object identifiers for identifying services is based on a tree-structure, usually drawn as in Figure 7.5.

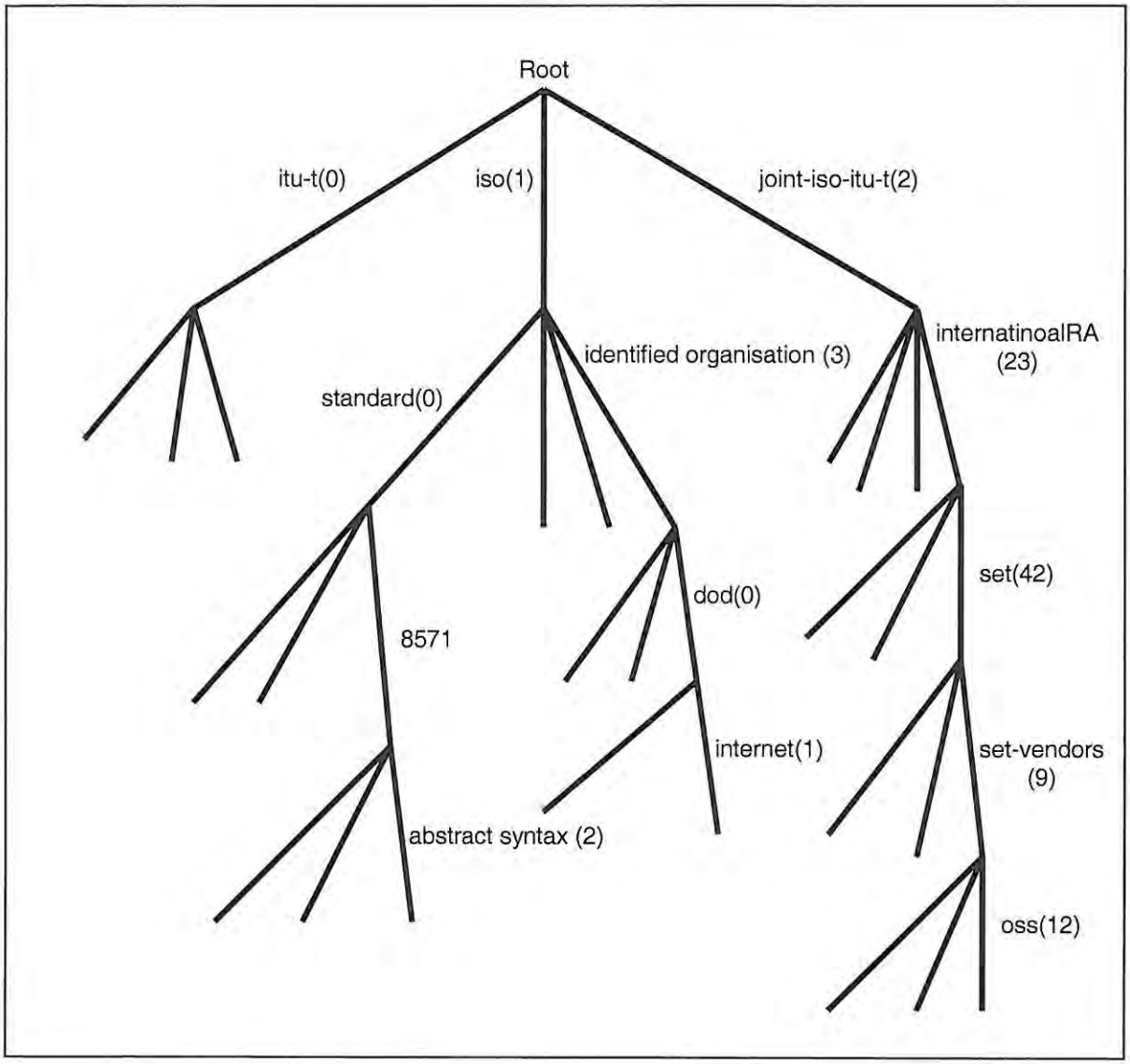

Figure 7.5 Subsection of Object Identifier Tree (from John Larmouth's ASN.1 Complete [36])

Each object identifier value corresponds to precisely one path from the root down to a leaf (or possibly an internal node), with each component of the value notation identifying one of the arcs

traversed on this path. The value notation consists of a series of components, one for each arc leading to an identified object.

From Figure 7.5 we can identify the object oss(12) at the bottom of the figure by:

```
{joint-iso-itu-t internationalRA (23) set (42) set-vendors (9) oss (12) }
```

Or equivalently by:

```
{2 23 42 9 12}
```

This method of service identification ensures that CANS' identification is unique among all possible service definitions, worldwide. The problem with this method is that it requires us to acquire a namespace in the object identifier tree, such as itu-t(0) or iso(1). As an academic institution with little interest in dedicated development of H.323 services, this was not undertaken. Instead we used the field *non-standard* field of the type `GenericIdentifier`.

The field *non-standard* of `GenericIdentifer` is of type `GloballyUniqueID`, which is in turn defined as ASN.1 type OCTET STRING. This means that we can use a unique hexadecimal byte stream to identify the service. The question we now asked was: what ID should be used to identify CANS?

The string of characters I chose that would uniquely identify the CANS is:

RhodesCANSService

When encoded using the ASN.1 PER rules, the above string of characters equates to:

52 68 6f 64 65 73 43 41 4e 53 53 65 72 76 69 63 65

This hexadecimal string represents the identification of CANS and can be transported within any of the three feature sets for H.460 feature negotiation.

Once I had addressed the identification of CANS I had to consider the negotiation procedures of the service using H.460.

I wanted to ensure with the service that standard H.323 terminals, not supporting the CANS service, would not engage in an H.323 call with any H.323 CANS server. This is because a standard H.323 terminal that does not support the CANS service is not capable of communicating with the CANS server in an appropriate manner.

To engage in a call with a CANS server, a CANS-enabled H.323 terminal must indicate its support for the CANS service. Failure to do so will result in the CANS server disconnecting the call before completion. Therefore, identification of the CANS service, introduced above, must be inserted into the *supported* feature set of the H.225 Setup message sent from the H.323 client terminal to the H.323 CANS server terminal. On receipt of the Setup message, the CANS server ensures that the CANS service identification exists within the *supported* feature set. If not, the CANS server will initiate call tear-down procedures by sending an H.225 Release Complete message to the H.323 client terminal. The CANS server sets the field `ReleaseCompleteReason` field of the Release Complete message to `neededFeatureNotSupported`, indicating to the H.323 terminal that it does not support a required service. Additionally, it includes identification of the CANS service within the *needed* feature set of the Release Complete message. As a result, the CANS client terminal is made aware of the service it requires (in this case, CANS).

7.5.3 Service Management using Annex K

H.323 Annex K allows CANS users to interact with the service using an HTTP connection to the CANS server. Through this channel users can customise the devices for their notification calls. Such customisation includes the ability to edit a list of devices on which they can receive notification calls, for example by adding or deleting devices.

Also, users are required to register with the CANS server before using it. They can do so using a standard web browser and navigating to the appropriate registration page on the CANS server. If unregistered users attempt to use the CANS service from their H.323 terminals, the CANS

server will instruct the client terminal to open the HTTP channel to the appropriate URL, in parallel with the standard H.323 call. This allows users to register directly from their Annex K-enabled H.323 terminals.

In H.323, Annex K information is contained within the H.225 protocol layer, in particular the call signalling messages, such as Setup, Release Complete, Facility, etc. An extract of the H.225 Facility message below shows the field `serviceControl` (line 9).

```
1       Facility-UUIE ::= SEQUENCE
2       {
3           protocolIdentifier          ProtocolIdentifier,
4           alternativeAddress          TransportAddress OPTIONAL,
5           alternativeAliasAddress     SEQUENCE OF AliasAddress OPTIONAL,
6           conferenceID                ConferenceIdentifier OPTIONAL,
7           reason                      FacilityReason,
8           .
            .
            .
9           serviceControl              SEQUENCE OF ServiceControlSession
            OPTIONAL
            .
            .
            .
10      }
```

The field `serviceControl` is defined as an optional array (SEQUENCE OF) `ServiceControlSession`, which is in turn defined as follows:

```
1       ServiceControlSession ::= SEQUENCE
2       {
3           sessionId                   INTEGER (0..255),
4           contents                    ServiceControlDescriptor OPTIONAL,
5           reason CHOICE
6           {
7                 open              NULL,
8                 refresh           NULL,
9                 close             NULL,
10                 ...
11          },
12          ...
13      }
```

The most important fields of the structure `ServiceControlSession` are the fields `contents` (line 4) and `reason` (line 5). The field `reason` is defined as a choice of either, `open`, `refresh`,

or `close` (lines 7, 8, and 9, respectively). These fields represent the actions to be taken regarding the HTTP connection. For example, a reason set to `open` indicates that a new HTTP connection must be opened to the URL specified by the field *contents*, `refresh` indicates a refresh of the URL, and `close` indicates a closure of the HTTP connection.

The `contents` field is defined as `ServiceControlDescriptor` of built-in ASN.1 type IA5String (International Alphabet 5). The field is used to specify an URL to be used by the H.323 client terminal. For example, the CANS server will send a Facility message to a CANS client if the user is not registered with the CANS system. The service control field within the Facility message contains Annex K information that requests the CANS client terminal to open an HTTP connection to the CANS server in order to complete registration. For example, the structure `ServiceControlSession` may have its field `contents` populated with the HTTP registration URL (http://IPorDNSofCANServer/register.html, for example). Since the CANS server is requesting the CANS client terminal to open a new HTTP connection to this URL, the field `reason` is set to `open`. A similar sequence of events is followed for device customisation.

To support Annex K, the CANS server needed to have a web server to serve the registration and device customisation pages. Instead of using a stand-alone web server like Apache or IIS, I chose to write one and integrate it into the CANS server application. Thus my web server runs within the CANS-enabled H.323 server terminal, eliminating difficulties that could arise with the interaction between the CANS server and stand-alone web servers. Another advantage of integrating my own web server into the CANS server is that I was not restricted by the APIs provided by stand-alone web servers for interaction. Instead the interaction between the CANS server and its embedded web server can be as flexible as I program it to be, with few limitations.

7.5.4 Audio Channels

Similar to EmailReader, CANS does not require audio channels to be open for the entire duration of the call between the CANS client and the CANS server. However when CANS users wish to record voice messages for their notification calls, an audio channel between the client and server must be open. This implementation includes buttons within the user interface that let

users control when the audio channels must be opened or closed. When users want to start recording their message, the button "Start Record" (component E in Figure 7.2) is pressed. This causes the H.323 logical channel to be opened between the CANS client terminals and the CANS server terminal. The channel will use the audio codec negotiated during call setup for encoding and decoding the audio between the two terminals. Users provide their own voice messages; these are transported via the logical channel (as RTP packets) to the CANS server, where they are recorded to a file and saved to disk.

In addition to opening and closing the audio channel from the CANS client to the CANS server, the CANS server must be instructed to process the audio it receives (i.e., save it to disk until initiation of the notification call). Without any indication that it should process incoming audio, the CANS server simply discards the audio as it is received. Before the H.245 procedures are initiated to open the audio channel the CANS H.450 operation `startRecord` is invoked on the CANS server by the CANS client. This process is initiated immediately once the user has pressed "Start Record" (component E in Figure 7.2) on the CANS client.

7.6 Service Availability to PSTN Terminals

As mentioned at the beginning of this chapter, CANS consists of two phases: the Notification Setup (NS) and the Notification Call (NC). The NS phase represents the period during which a user sets up a particular notification and saves the information to the CANS server. The NC phase represents that actual callback initiated by the CANS server to the user's device for presentation of the reminder. Currently the NC phase of CANS supports PSTN and GSM terminals. The set of devices on which the notification call can terminate includes PSTN and GSM telephones via the H.323/ISDN gateway (see section 7.4.3). If a textual notification is to be sent the system utilises the SOAP GSM SMS gateway (see section 7.4.4) to send an SMS to the user. The possibility of making the NS phase of CANS available to the PSTN and GSM networks is discussed in the next paragraph.

CANS is implemented as an H.450 service with all CANS-enabled H.323 terminals interacting with the H.323 CANS server using H.450 operations and data structures. Because the underlying

hardware of terminals on the H.323 network is PC-based, the user interface is relatively advanced and graphical in nature (GUI). (The CANS client user interface was described in section 7.2.2.) The PSTN only supports voice for interface presentation and not GUIs. PSTN and GSM users therefore require the user interface to be presented to them as voice prompts i.e. via an interactive voice response (IVR) system. Such an IVR system can be integrated into the CANS server. PSTN and GSM users interact with the IVR system using DTMF as this is the only form of user input supported by the PSTN. This means that DTMF tones from the PSTN and GSM networks need to be translated into the appropriate CANS H.450 operations. Translation of this service information can be performed directly at the H.323/ISDN gateway by integrating a new service-control translation module (STM) into the H.323/ISDN gateway. Alternatively, the STM could be integrated into the H.323 CANS server itself.

7.7 Discussion

The service creation and extensibility frameworks of H.323 are adequately flexible, and once understood, allow for easy implementation of standard services. Being able to use these mechanisms provides implementers with a large amount of flexibility in the creation of their products, allowing H.323 networks to become more intelligent and more useful, resulting in state-of-the-art communications networks.

Currently there is plenty of industry support for H.323 and new services are being investigated as a result of this interest. Such services currently undergoing standardisation include H.323 Mobility, both user and terminal, as well as presence systems that allow users to see which of their contacts are online and as a result, callable. Essentially all of these services make use of some or all of the above-mentioned H.323 extensibility and service creation mechanisms. It is therefore imperative that implementers understand these mechanisms in order to provide standard and reliable services that offer a high standard of interoperability with other H.323 systems.

When creating new H.323 version-4 services, specifically signalling services, implementers are faced with various decisions, including which H.323 service creation mechanisms to use. For

example, CANS is implemented primarily as an H.450 service and negotiated using H.460. I chose to implement it in such a fashion so as to explore both the H.450 and H.460 service creation mechanisms. CANS could have been implemented entirely as an H.460 service, although this approach would have been tedious considering the number of parameters required for CANS. In fact CANS may also have been implemented as a non-signalling service. As a non-signalling service CANS would require an IVR system to present users with a user interface and allow them to input information using DTMF tones.

Finally, I consider the possibility of offering a signalling service, like CANS, to PSTN and GSM users. In this case, an IVR system would be required to present users with an interface in the form of voice prompts on their PSTN and GSM telephones. To use an IVR system, PSTN and GSM users need to control the navigation of the system. In addition they may be required to provide service terminals with information necessary for their operation, for example, entering a date and time for a CANS notification call. User input in the PSTN is only supported via the use of DTMF tones. This means that a service-control translation module (STM) is required to convert the DTMF tones from the PSTN into H.450 operations and data structures. The STM can either be integrated in a gateway that bridges the H.323 and PSTN networks (e.g., the H.323/ISDN gateway discussed in section 7.4.3), or it could be integrated into the actual H.323 service terminal itself. The choice of location of such STMs ultimately depends on processing power. Gateways can often be heavily loaded in terms of the amount of processing power required, especially when there are numerous concurrent calls. This is due to the protocol signalling translation and media transcoding required for bridging two distinct networks. It is thus better to integrate the STM into the service terminal whenever possible.

7.8 Summary

In this chapter I discussed an H.323 service newly implemented during this research, called Customisable Alarm Notification System (CANS). The discussion includes a description of CANS' operation, its user interface and individual components. CANS is implemented as an H.323 signalling service, thus I introduced and discussed the H.323 version-4 service creation mechanisms H.450 and H.460 in an example-driven manner based on CANS. Guidelines which

aid the decision of which mechanism to use when creating a new service were presented along with discussions of the decisions I made in the implementation of CANS. CANS is implemented using a combination of H.450 and H.460; the service itself is implemented using H.450 while the H.460 feature negotiation mechanism is used to negotiate the CANS service. My discussion of the CANS service therefore includes a description of the implementation of CANS using H.450 and the way it is negotiated using H.460. I also discussed the use of H.323 Annex K that facilitates service management in CANS via a separate HTTP connection. Finally, I considered how the NS phase of CANS could be made available to PSTN and GSM telephones users.

CHAPTER 8

DISCUSSION AND FUTURE WORK

"What we call the beginning is often the end

And to make an end is to make a beginning.

The end is where we start from."

T.S.Elliot: Four Quarters 'Little Gidding'

A major trend in modern telecommunications is to transport voice on IP networks as opposed to the PSTN. Advancements in improved quality of service for real-time traffic on the IP network have been a major factor contributing to this trend over the past few years. Another contributing factor is the possibility of creating a new range of innovative, state-of-the-art telephony services that take advantage of the underlying IP network's intelligence and flexibility. The latter has yet to be fully explored and the work reported in this thesis is, hopefully, a contribution to the exploration. Various protocols exist that facilitate the transport of voice and other media, such as video, on IP networks. The best known, and widely supported, is H.323, with which this thesis is concerned. In this chapter I summarise and discuss the work done and suggest extensions that could break new ground surrounding the H.323 protocol.

8.1 Research Questions Revisited

The questions that guided the research, outlined in Section 1.4, are revisited and answered based on the findings of my study.

1. *What is an H.323 service?*

I have defined an H.323 service as an entity that enhances the functionality of an H.323 network, or adds value to users' H.323 multimedia calls. Such enhanced functionality of H.323 networks includes services like user and terminal mobility [28, 29], number portability [30], and user presence [21]. Value-added services that add functionality to users' H.323 calls include the set of basic supplementary services we have become accustomed to on the PSTN (such as call hold, call divert, call intrusion, etc.), as well as new services developed during this project (such as EmailReader, Telgo323, and CANS).

2. *Can H.323 services be logically categorised in terms of their functionality, design, and implementation?*

One of the practical applications of the research is a method of categorising H.323 services (Chapter 3). By investigating the properties of various H.323 services, based on their design, implementation, and functionality, the research identified eight service categories: basic, enhanced, standard, proprietary (non-standard), signalling, non-signalling, centralised, and distributed. These categories are non-orthogonal meaning that a single H.323 service may fall into more than one category; for example, CANS is classified as an enhanced, non-standard, distributed, signalling service. I have shown that categorisation of H.323 services often depends on the manner in which they are implemented. For example, EmailReader is implemented as a non-signalling service; all information used to control the service is transported using DTMF tones. Alternatively, EmailReader may have been implemented using H.450 or H.460, whereby service information would be transported within H.225 call signalling messages. Such an implementation would result in the change of EmailReader's

classification from that of a non-signalling service to a signalling one. Although the implementation of EmailReader as a signalling service would be feasible, it would appear a poor idea because of the small amount of information transfer required for its operation. The mechanisms H.450 or H.460 would add a significant amount of overhead to the service with no advantages to its operation.

3. *What service architectures and mechanisms does H.323 provide that support service creation and protocol extension?*

H.323 version 4 supports service creation and extensibility through the umbrella recommendations H.450, *Generic Functional Protocol for the Support of Supplementary Services in H.323* and H.460, *Generic Extensibility Framework*. Both of these mechanisms were studied during the course of the research and they are discussed throughout the work. Both H.450 and H.460 can be used for service creation; H.460 can additionally be used for H.323 feature negotiation. In chapter 5, I provided guidelines that facilitate a decision regarding when to use H.450 or H.460 to implement new H.323 services. The use of these guidelines was demonstrated in regard to CANS, which is implemented as an H.450 service and negotiated using H.460. H.450 and H.460 are complete mechanisms that can be used to create powerful signalling services in H.323 environments. Services implemented using H.450 or H.460 automatically fall into the signalling service category because of their requirement to transport service control information using H.323 signalling messages.

4. *If possible, how can H.323 voice services accommodate traditional PSTN users?*

Voice calls can be quite easily established between H.323 users and PSTN and GSM users alike. The essential H.323 component required to bridge the PSTN and IP network is a gateway. Our H.323 test network features an H.323/ISDN gateway that bridges our H.323 network to the PSTN via numerous ISDN interfaces. Using this gateway in conjunction with our gatekeepers, voice calls can be made from H.323 terminals in our network to PSTN and GSM terminals on the PSTN.

The standard H.323/ISDN gateway addresses the bridging of call signalling between our H.323 test network and the PSTN, but what about service interworking? H.323 is designed to interoperate with traditional telephony services (call forward, call hold, etc.). This is evident in the fact that the underlying call signalling messages of H.323 are based on Q.931 and Q.932, the call signalling protocols found in the PSTN. In addition the H.450 SS framework in H.323 is based on QSIG, the service control protocol found in private switched-circuit voice networks, or PBXs. This facilitates the use of basic H.323 services by PSTN and GSM users. The research addresses the ability to interact with enhanced H.323 services, in particular EmailReader and CANS, from PSTN and GSM terminals. However, PSTN and GSM terminals lack the intelligence available in H.323 terminals and can only interact with network entities using DTMF tones.

DTMF information can be transported in various ways in H.323 networks in a protocol compliant manner; I used out-of-band DTMF transport, using the H.245 media control channel of H.323. I modified the basic H.323/ISDN gateway to extract the DTMF tones received from PSTN terminals and GSM cellular phones and repackaged them within H.245 call control messages for transport to the H.323 service terminals. This allows PSTN and GSM telephones to use certain services, such as EmailReader, as if they were residing on the PSTN.

To accommodate this method of service control, EmailReader was appropriately implemented to accept the same method of control from H.323 terminals and PSTN telephones alike (i.e., out-of-band DTMF transmission using the H.245 media control channel). With DTMF, users can control the service and provide input, yet they still require a user interface. When developing H.323 services with the intention of making them available to the PSTN, the user interface must be carefully considered. Terminals on the PSTN are not normally capable of providing graphical user interfaces (GUI) that are otherwise common on PC-like devices. On the other hand, all terminals on the PSTN and GSM networks support voice communication and the results of the research suggest that the user interface is best provided as voice prompts using Interactive Voice

Response (IVR) systems. Such an IVR system was developed and integrated into EmailReader, but can be used anywhere as needed.

5. *Does H.323 provides service management?*

H.323 Annex K, *HTTP Based Service Control Transport Channel,* is an addition to H.323 that facilitates service management in H.323 networks. It is a powerful mechanism that can be used by a variety of services. With Annex K users can turn services on or off, customise the services they use, etc. The project explored Annex K functionality by integrating it into CANS, the service discussed in Chapter 7, to allow users to customise it (such as, by registering with the service and listing their callback devices). We propose that Annex K will become a vital component for enabling users to directly control third-party telephony services.

6. *How difficult/easy is it to create new H.323 services?*

When considering the degree of difficulty of creating new H.323 services it is important to distinguish between signalling and non-signalling services. With basic knowledge of the H.323 protocol and the protocols it specifies for its operations, creating new non-signalling services is relatively straight forward. Good programming skills combined with a basic knowledge of H.323 will enable developers to create new and exciting non-signalling services. Programming knowledge is especially useful for integrating various computer technologies into H.323 applications. (For example, STT, IMAP and IVR technologies were integrated into the EmailReader service. Importantly, one must know in which H.323 components the service logic can be hosted, and how the service logic can interact with call signalling and control operations.

H.323 signalling services are more complex than non-signalling ones. To create a signalling service in H.323, a developer must be familiar with the H.450 and H.460 service creation mechanisms. In addition, a good understanding of ASN.1 and how it can be used to specify various data structures is required. When creating signalling

services many decisions may need to be made, such as when to use H.450 or H.460 for service implementation or H.460, or a combination of the two. Making such decisions becomes easier with experience and the more experienced the developers the more advanced and efficient their services become.

8.2 Summary of Work Covered

Substantial experience with service creation within H.323 networks was gained during the course of the research and the project has made several useful contributions:

Service Categorisation:

- The research identified eight non-orthogonal categories in which H.323 services can be classified based on their functionality, design, and implementation. This was the result of an investigation into the mechanisms provided by the H.323 protocol that facilitate protocol extensibility and service creation; this provided sufficient background for understanding the various ways in which new services can be created. With this knowledge I suggest a categorisation method of H.323 services.

Service Management:

- The research has contributed to the ongoing migration of OpenH323's protocol stack from version 3 to version 4 by integrating support for H.323 Annex K. H.323 Annex K, *HTTP Based Service Control Transport Channel*, facilitates service management in H.323 version-4 networks. This is an important feature of H.323 that enables users to customise the services they use. At the time of the research Annex K was not supported by the OpenH323 project's H.323 protocol stack, which was used for H.323 development. To explore Annex K functionality it was necessary to implement and integrate it into the protocol library.

Actual development of real, innovative H.323 services:

- Three enhanced H.323 services were developed as an outcome of the study: EmailReader, Telgo323, and CANS. Much of the practical work was based on the design and implementation of the two major categories of H.323 version-4 services: signalling and non-signalling. Both EmailReader and Telgo323 are examples of non-signalling services while CANS is an example of a signalling service. In chapters 6 and 7 I offer an introduction to and a relatively detailed discussion of the functionality, design and implementation of EmailReader and CANS, respectively. Telgo323 is not discussed in this thesis since it is a non-signalling service, like EmailReader. However, it is included as Appendix C for interested readers.

Other contributions of the research that do not offer material deliverables (like the previous three contributions) instead offer valuable insight into the various issues that were encountered during the study.

Requirement of STMs in H.323-PSTN Service Interworking:

- The user interface (UI) is an important component of any H.323 service. The more advanced and more graphical the UI of a service is, the easier it is for users to interact with the service. One factor limiting the capabilities of a UI is the intelligence of the terminal on which it will run. As mentioned, most of our services were implemented to cater for use by PSTN terminals. Such terminals are relatively dumb, only capable of sourcing and terminating audio streams. Services implemented for the PSTN, therefore, have to provide a UI in the form of voice prompts (i.e., using interactive voice response or IVR systems). Users can then control the service, navigating it and providing user input, via DTMF tones. This makes it relatively easy for non-signalling H.323 services to be made available to the PSTN, but what about signalling services? Signalling services are more difficult to extend to the PSTN and they require an entity to

translate user interface and service control information between the H.323 network and the PSTN. I offer the notion of a Service Translation Module (STM) employed to do exactly this: bridge environments or networks that differ at the service control layer. The STM is therefore required to translate, for example, DTMF from the PSTN into the appropriate H.450 or H.460 data structures, necessary in H.323 signalling services. The STM must also be able to interact with an IVR so that it is aware of the IVR system's state and the information it is anticipating from the user. An STM can either interact with an IVR system within the H.323 network or it can have its own generically, built-in system. Discussion of STMs and their location within an H.323 network was presented in section 7.6.

Service Control:

- One issue addressed by the work concerning H.323 service creation is the ways in which users can interact with the services they use (service control). In H.323 networks services can be controlled in many ways depending on their implementation. I concluded that most non-signalling services are currently best controlled using DTMF tones. In section 3.4 I introduced a number of ways to carry DTMF tones in the H.323 network in a protocol compliant manner. Of these the project adopted out-of-band transport of DTMF using the H.245 UserInputIndication messages for transport of DTMF in our H.323 development environment. On the other hand, signalling services are most often controlled using a particular H.323 service creation mechanism, H.450 or H.460, to implement the service. CANS, discussed in Chapter 7, is implemented and controlled using H.450.

PSTN Service Interworking:

- Throughout this study and particularly with regard to implementation of our H.323 services we thought about H.323/PSTN service interworking. The existence of the H.323/ISDN gateway in our test network kept us thinking of ways in which our services could be made available to PSTN users. For example, EmailReader (Chapter 6) was designed from the start with the intention of offering it as a service to the PSTN, a natural target for a service of this nature. Telgo323 (Appendix C), as another example, was developed specifically for the PSTN user-base. Finally, CANS (Chapter 7) was developed as a service aimed at the H.323 user-base only, although notification calls could terminate on any device available via our H.323 network, including the PSTN and GSM terminals. In section 7.6 I looked at possible ways to extend CANS to the PSTN. Practical implementing of these ideas was not carried out but is left as a possible extension to the study.

8.3 Future Research

The focus of this study was on the categorisation and implementation of H.323 version-4 services. The project's implementation of these services is based on the service architecture provided by the H.323 recommendation (i.e., H.450, H.460, etc.). This service architecture requires the terminals of an H.323 network to individually support a service. This means that when a new service is created it must be integrated into all the H.323 terminals of an H.323 environment. For example, let us assume that we have just developed a new service called X. If we want all terminals in our H.323 network to support X then we have to upgrade every endpoint in the network. Upgrading the endpoints requires us to integrate the software module into the terminal software applications via plug-ins, or by upgrading the software application itself.

Future research may consider the possibility of creating a more open and flexible provisioning of advanced H.323 services. For example, by using technologies like Extensible Markup Language (XML), Common Object Request Broker Architecture

(CORBA) and Enterprise JavaBeans (EJB), it may be possible to dynamically invoke services located in specialised network nodes as opposed to the H.323 terminals themselves. This means that specialised servers could perform services on behalf of the H.323 terminals. Such architecture would enable service providers to deploy H.323 services within specialised network nodes, making the service accessible to all H.323 terminals without upgrading the terminal software.

Issues to be addressed in such a study should include: the subscribing and unsubscribing of services, locating the specialised nodes that offer the services, location of service code, identification of services within directory listings, mobility of service code (from specialised network nodes to the H.323 terminals that require service logic), and service management that allows users to customise services.

Consider an example of the advantage of such an architecture. In Chapter 7, I mentioned that H.323 terminals that do not support CANS can not engage in an H.323 call with the CANS server. I enforced this restriction using the H.460 feature negotiation functionality, discussed in section 7.5.2. If the H.323 terminal does not include the identification of the CANS service in the H.460 *supported* feature set of the Setup message to the CANS server, the call will fail. The call fails by the CANS server sending a Release Complete message to the H.323 terminal with the field `ReleaseCompleteReason` set to `neededFeatureNotSupported`, indicating to the H.323 terminal that it does not support a required service. Additionally, it includes identification of the CANS service within the *needed* feature set of the Release Complete message, making the H.323 terminal aware of the service it requires (i.e, CANS). The H.323 terminal in this case will not be able to engage in an H.323 call with the CANS server unless a CANS plug-in is installed or an entirely new H.323 application that supports CANS is installed. However, using the new service proposed here, the H.323 terminal could dynamically locate and make use of the service. The following presents an example design for this new architecture. (Please note that this is

a simplified proposal for the new service architecture in order to illustrating a possible approach).

Figure 8.1 illustrates example architecture for H.323 dynamic service invocation. This illustration assumes that the H.323 terminal (not supporting the CANS service) has already attempted an H.323 call with the CANS server, and has been unsuccessful. The terminal is therefore aware that it does not support the appropriate service (CANS) to engage in a call with the CANS server. It is also aware of the identification of the service it requires (RhodesCANSService) from the *needed* feature set returned to it by the CANS server.

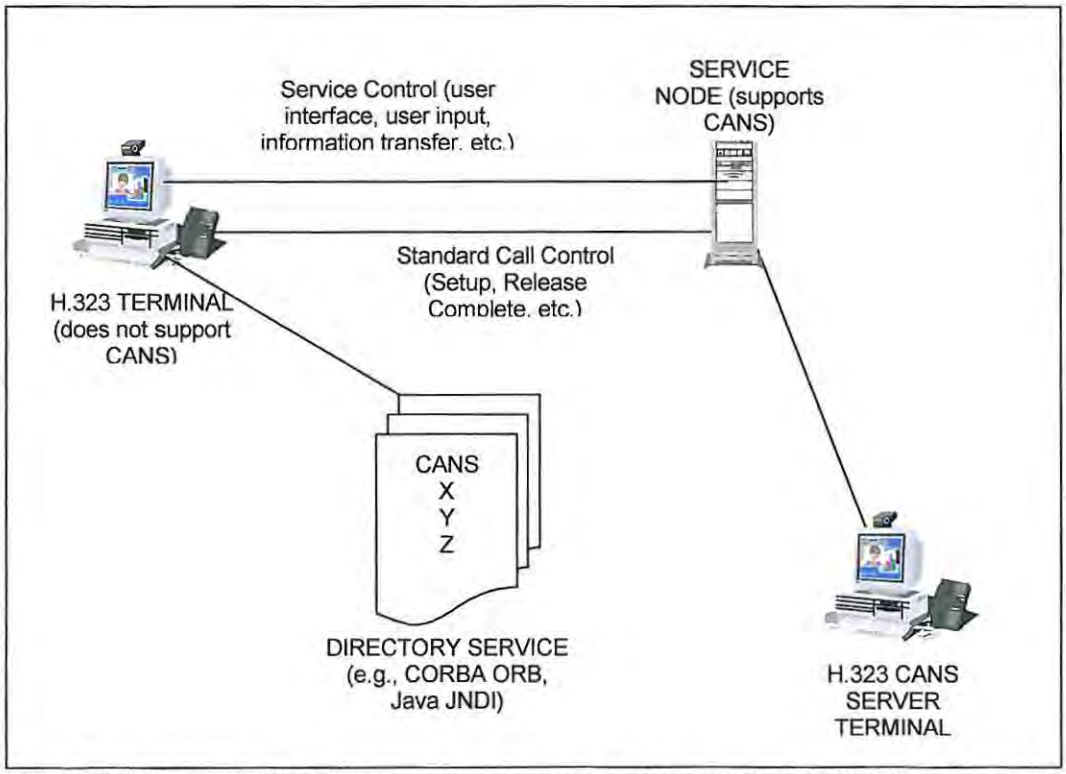

Figure 8.1 Example Architecture for Dynamic and Extensible H.323 Services

The H.323 terminal needs to locate the specialised service node that will perform the CANS service on its behalf. It does this via a directory service or interface repository that returns the location of a CANS service node. Once the terminal knows the location

of the CANS service node it can attempt a call to the CANS server. For this call the CANS service node will act as a proxy (it will relay all call signalling and control messages to and from the H.323 terminal and CANS server via the CANS service node). This will allow the service node to alter the protocol messages necessary for operation of the service. For example, the H.225 Setup message from the H.323 terminal will not specify support for the CANS service in the *supported* feature set. This is because the terminal still does not support the service on its own. Instead the service is performed by the service node on its behalf and as a result the H.225 Setup message is altered, by adding CANS support into the *supported* feature set of the H.225 Setup message before passing it onto the CANS server.

Now that the call has been successfully established between the H.323 terminal and the CANS server, via the CANS service node, we must consider how users may interact with the service. CANS requires a significant amount of user information, like the callback time, date, device number, etc. Applications that require a significant amount of user interaction usually require a graphical user interface (GUI). There are a number of ways that a GUI can be provided. For example, by downloading a standard description of the GUI from the CANS service node, the H.323 terminal can generate an interface using this description and take input from the user, whereby it can be returned to the service node. Alternatively, a standard HTTP service channel could be established, possibly using H.323 Annex K, between the H.323 terminal and the CANS node. Further study will determine which of numerous methods will prove most usable and efficient.

A study into creating a new H.323 service architecture of this nature will not be a trivial task. There are many technologies that support dynamic invocation of distributed objects, code mobility using mobile agents, etc., but I suggest that the technologies used to support the architecture are abstract, limiting any binding to programming language and operating system dependence. For this reason, existing technologies like XML and CORBA may play a vital role in the new architecture.

REFERENCES

[1]. Agilent Technologies. 2000. *DTMF Tone Analysis.* (http://onenetworks.comms.agilent.com/VQT/VQT_DTMF.asp).

[2]. American National Standard for Telecommunications. 2000. *Telecom Glossary (Protocol Data Unit).* (http://www.atis.org/tg2k/_protocol_data_unit.html).

[3]. Centre for Speech Technology Research. 2002. *Festival Speech Synthesis System.* University of Edinburgh. (http://www.cstr.ed.ac.uk/projects/festival/).

[4]. Chatras, B. and Chapron, Jean-Emmanuel. 2001. *An Analysis of the IN call model suitability in the context of VoIP.* Computer Networks. 35(5):521-535.

[5]. Chatzipapadopoulosa, F., De Zenb, G., Magedanzc, T., Venierisa, I. S. and Zizzab, F. 2000. *Harmonised Internet and PSTN Service Provisioning,* Computer Communications. 23:731-739.

[6]. Cisco Systems Incorporated. 2002. *Cisco Unity with Call Manager Problem with DTMF.* (http://www.cisco.com/warp/public/788/AVVID/unity_cm_dtmf_prob.html).

[7]. Davidson, J. and Peters, J. 2000. *Voice over IP Fundamentals.* Cisco Press. Indianapolis.

[8]. Dubuisson, O. 2000. *Communication between heterogeneous Systems.* Morgan Kaufmann Publishers. San Francisco.

[9]. Gbaguidi, C. and Hubaux, J-P. *An Architecture for the Integration of Internet and Telecommunication Services.* Institute for computer Communications and Applications. Swiss Federal Institute of Technology. Switzerland.

[10]. Gbaguidi, C., Hubaux, J-P. and Hamdi, M. *A Programmable Architecture for the Provision of Hybrid Services.* Institute for computer Communications and Applications. Swiss Federal Institute of Technology. Switzerland.

[11]. George, H.W. *ASCII, Baudot and the Radio Amateur.* (http://www.comunicacio.net/digigrup/ccdd/rtty.htm).

[12]. Glaser, M. 2001. *A Field Trial and Evaluation of Telkom's Teldem Terminal in a Deaf Community in the Western Cape.* Proceedings of the South African Telecommunications Networks and Applications Conference. Wild Coast Sun. South Africa. (CD-ROM publication).

[13]. Glaser, M. and Tucker, W.D. 2001. *Web-based Telephony Bridges for the Deaf,* South African Telecommunications Networks and Applications Conference. Wild Coast Sun. South Africa. (CD-ROM publication).

[14]. Glasmann, J., Kellerer, W. and Müller, H. *Service Architectures in H.323 and SIP – A Comparison.* Munich University of Technology (TUM). Germany.

[15]. Glitho, R.H. *Emerging Alternatives to Today's Advanced Service Architectures for Internet Telephony: IN and beyond.* Computer Networks. 35(5):551-563.

[16]. Halse, G.A. and Wells, G. 2002. *A Bi-directional SOAP/SMS Gateway Service.* South African Telecommunications Networks and Applications Conference. Drakensburg. South Africa. (CD-ROM publication). Available online at http://coe.ru.ac.za/papers/halse/halse-satnac-2002.pdf.

[17]. Hamdi, M., Hubaux, J-P. and Verscheure, O. 1999. *Voice Service Interworking for PSTN and IP Networks.* Institute for computer Communications and Applications. IEEE Communications Magazine. May 1999.

[18]. Hao, You-chao., Wang Wen-dong, Lu., Mei-lian, and Cheng, Shi-duan. 2001. *Interworking between H.323 network and intelligent network and analyzing of its evolution trend.* Journal of Beijing University of Posts and Telecommunications. 24(2):46-50.

[19]. Hersent, O. and Petit, J. 2000. *IP Telephony Packet-based Multimedia Communications Systems.* Pearson Education Limited. Great Britain.

[20]. International Telecommunications Union. 1992. *Recommendation T.51 Packet-based Multimedia Communications System.*

[21]. International Telecommunications Union. 2002. *Recommendation Draft H.460.3 Presence for H.323 Systems.*

[22]. International Telecommunications Union. 2000. *Recommendation H.225 Call Signalling Protocols and Media Stream Packetization for Packet-based Multimedia Communications Systems.*

[23]. International Telecommunications Union. 2001. *Recommendation H.245 Control Protocol for Multimedia Communication.*

[24]. International Telecommunications Union. 2000. *Recommendation H.323 Annex K HTTP Based Service Control Transport Channel.*

[25]. International Telecommunications Union. 2000. *Recommendation H.323 Packet-based Multimedia Communications Systems.*

[26]. International Telecommunications Union. 1998. *Recommendation H.450 Generic Functional Protocol for the Support of Supplementary Services in H.323.*

[27]. International Telecommunications Union. 2002. *Recommendation H.460 Guidelines for the Use of the Generic Extensibility Framework.*

[28]. International Telecommunications Union. 2002. *Recommendation H.501 Protocol for Mobility Management and Intra/Inter-domain Communication in Multimedia Systems.*

[29]. International Telecommunications Union. 2002. *Recommendation H.510 Mobility for H.323 Multimedia Systems and Services.*

[30]. International Telecommunications Union. 2001. *Recommendation H.GEF.2 Number Portability Interworking between H.323 and SCN Networks.*

[31]. Internet Engineering Task Force. 2000. *RFC 2833 RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals.* Available online at http://community.roxen.com/developers/idocs/rfc/rfc2833.html.

[32]. Internet Engineering Task force. 1996. *RTP: A Transport Protocol for Real-time Applications.* Available online at http://www.ietf.org/rfc/rfc1889.txt.

[33]. ISDN4Linux Project. 1998. *ISDN Protocol API.* (http://www.isdn4linux.de).

[34]. Jeffries, M. and Tucker, W.D. 2001. *An Interoperable Signalling Solution between SIP and H.323.* South African Telecommunications Networks and Applications Conference. Wild Coast Sun. South Africa. (CD-ROM publication).

[35]. Kulathumani, V. 1999. *Voice over IP: Products, Services and Issues.* Ohio State University. (ftp://ftp.netlab.ohio-state.edu/pub/jain/courses/cis788-99/voip products/index.html).

[36]. Larmouth, J. 1999. *ASN.1 Complete.* Morgan Kaufmann Publishers. San Francisco. Available online at http://www.nokalva.com.asn1/larmouth.html.

[37]. Licciardi, C. A., Canal, G. and Andreetto, A. 2001. *An architecture for IN hybrid services.* CSELT (Centro Studi e Laboratori Telecomunicazioni). International Journal of Computer and Telecommunications Networking. 35(5):537-549.

[38]. Linux.be. 1999. *Open Source Advantages.* (http://linux.iguana.be/open-source/).

[39]. OpenH323 Project. 1998. *H.323 Open-Source Project.* (http://www.openh3232.org).

[40]. OpenH323 Project. 1998. *Project documents.* (http://www.openh323.org/docs/).

[41]. OpenH323 Project. 1998. *Welcome to the Wonderful World of PWLib.* (http://www.openh323.org/docs/PWLib/Introduction.html).

[42]. Penton, J.B. 2000. *Study and Deployment of the H.323 Standard for Videoconferencing on Packet Switched Computer Networks.* Honours Thesis. Rhodes University. Grahamstown. South Africa.

[43]. Penton, J.B., Terzoli, A. 2002. *CANS: Customisable Alarm Notification System, an H.323 Signalling Service*. South African Telecommunications Networks and Applications Conference. Drakensburg. South Africa. (CD-ROM publication).

[44]. Penton, J.B., Terzoli, A. and Wentworth, E.P. 2001. *Deploying a Feature-rich H.323 Environment in which to Practice the Creation of Additional Services*. South African Telecommunications Networks and Applications Conference. Wild Coast Sun. South Africa. (CD-ROM publication).

[45]. Penton, J.B., Tucker, W.D. 2002. *Telgo323: An H.323 Bridge for Deaf Telephony*. South African Telecommunications Networks and Applications Conference. Drakensburg. South Africa. (CD-ROM publication).

[46]. Radvision Incorporated. 2002. *The H.323 Revolution: A Technology Review*. Available online at (http://www.h323forum.org/papers/TheH.323Revolution-May-2002.pdf).

[47]. Sevilla, C. 2000. *OpenISDNGw*. (http://www.gae.ucm.es/~openisdngw/home_en.php).

Appendix A: ASN.1

A.1 Introduction

In his book *ASN.1 Communication between Heterogeneous Systems* [8], Olivier Dubuisson describes Abstract Syntax Notation One (ASN.1) as:

- A notation that is used in describing messages to be exchanged between communicating application programs. It provides a high-level description of messages that frees protocol designers from having to focus on the bits and bytes layout of messages.
- Supported by sets of standardized encoding rules that describe the bits and bytes layout of messages as they are in transit between communicating application programs.

Another source [36] describes the ASN.1 notation as:

- An internationally-standardised, vendor independent, platform independent and language independent notation for specifying data structures at a high level of abstraction.

Olivier Dubuisson, representing France Telecom Research and Development, presented a number of uses of ASN.1; a few of these are:

- Intelligent network (IN)
- GSM
- UMTS (3G cellphones)
- Interactive television.

From this we can see that ASN.1 is widely used and is a good notation to have experience in, especially in telecommunications circles.

A.2 Encoding Rules

ASN.1 defines three syntaxes for communication protocols:

- Concrete Syntax
- Abstract Syntax
- Transfer Syntax

The concrete syntax is a representation of the data structures to be transferred in a given programming language. It is a syntax because it respects the lexical and grammatical rules of a language (C++ for instance); it is called concrete because it is actually handled by applications (implemented in this very language) and it complies with the machine architectures' restrictions.

In order to break free of the diversity of concrete syntaxes mentioned above, the data structures to be transmitted should be described regardless of the programming languages used. This description should also respect the lexical and grammatical rules of a certain language (e.g., ASN.1) but should remain independent from programming languages and never be directly implemented on a machine. Such syntax is defined as an abstract syntax and, Abstract Syntax Notation or ASN, the language whereby this abstract syntax is described.

The transfer syntax represents the actual bits on the wire. The transfer syntax is obtainable from the abstract syntax by following a sequence of rules on how the data is represented and formatted on the wire. Such rules are defied as encoding rules and describe the links between the abstract syntax and the transfer syntax.

According to John Larmouth: "The encoding rules say how to represent with a bit-pattern the abstract values in each basic ASN.1 type, and those in any possible constructed type that can be defined using the ASN.1 notation" [36].

There are several standard encoding rules used to convert ASN.1 data structures into bit strings:

- Basic Encoding Rules (BER)

- Distinguished Encoding Rules (DER)
- Canonical Encoding Rules (CER)
- Packer Encoding Rules (PER).

The Basic Encoding Rules (BER) were the first standardised encoding rules and they use type-length-value encoding (TLV). They provide two options for encoding data values depending on whether or not the length of the data value is known beforehand. This gave rise to the canonical encoding rules (CER) and the distinguished encoding rules (DER) which are variants of BER that encode data values each using one of the options available in BER [36].

"The Packed Encoding Rules were designed to provide very compact encodings of ASN.1 data structures. They are used in bandwidth-constrained environments where the protocol messages must be as small as possible." [36] These are the encoding rules specified for use in H.323 systems. Reasons for this choice are obviously based on the smaller more compact bit streams of all H.323 protocol messages, resulting in less bandwidth required for H.323 communication on IP networks. In addition, the use of PER can also minimise the call set-up time in H.323 calls.

A.3 Using ASN.1

ASN.1 provides a protocol designer with a rich set of elements with which to construct messages in a protocol. In H.323 however, all protocol messages are standardised and already defined. Abstract syntaxes of these messages may not be altered or added to without affecting the standard. This means that new H.323 service information cannot be readily added to the protocol messages unless formally ratified by the ITU. Fortunately the standard provides placeholders in the abstract syntax that allows for extensibility within the H.323 messages. The location and contents of these placeholders is specific to the service mechanisms being used (e.g., H.450, H.460, Annex K, and others). If we want to create signalling services using H.450, H.460 or any other standard service creation mechanism in H.323 we must understand how to use ASN.1, since all information embedded within the given placeholders of the H.323 messages are encoded using ASN.1.

Implementing the service information and interaction using this notation requires code in a particular programming language to represent, encode and decode the message structures according to a set of encoding rules (e.g., PER for H.323). Fortunately there are tools available that accept an ASN.1 specification and automatically generate all the programming language code necessary to represent the message structures and encode and decode messages during protocol and service interaction. Such tools are referred to as ASN.1 compilers (introduced in Chapter 4).

A.3.1 Our Experience

The entire H.323 call signalling and service interaction protocols (H.225, H.245, H.450, H.460) have their message structures and operations specified with ASN.1. The research therefore required extensive knowledge of ASN.1 before the necessary service information could be added to the appropriate message in the correct placeholders within the ASN.1 specifications. This was a daunting task that became easier with practice.

I fully explored the functionality of ASN.1 as the services that were created and investigated became more complex and by the end of the study I could arguably say that it is powerful language that may still endure for some time. I also agree with its reputation for complexity, especially its encoding. A new encoding rule that has recently been addressed is that of the XML encoding rules (XER). These rules specify a means of converting ASN.1 structures into the Extensible Markup Language (XML). XML is text based and makes the encodings human-readable for easier interpretation. This encoding rule may gain wider use in the future resulting in a less complex ASN.1

A.3.2 ASN.1 Syntax

ASN.1 is complex with entire books dedicated to its syntax, use and compilation. Here I briefly introduce the syntax, its most basic structures and principles, so that material presented in later chapters is clearer and more concise. Much of the material in this section is taken from John Larmouth's book called *ASN.1 Complete* [36].

The easiest way to learn ASN.1 is to try it, and so I present an example to introduce it. The example is a simple one in which the goal is to describe some basic information about all students at our university. We want this information to be transported between two machines connected via a low bandwidth link. The purpose of this exercise involves a central database that keeps records of all

students registered at the various tertiary institutions throughout South Africa. How can this information be communicated by the various institutions' databases? The answer is by using ASN.1. The information must be formatted according to the ASN.1 notation and transported via the link to the central machine hosting the database.

For this example, information chosen to describe student information includes:

- Name and surname
- Age
- ID Number
- Address
- Faculty

The following code fragment illustrates the ASN.1 notation, written for this simple scenario.

```
1               Students ::= SET OF Student
2               Student ::= SEQUENCE {
3                name    GeneralString,
4                age     INTEGER,
5                id      NumericString,
6                address ENUMERATED
7                {
8                 eastlondon(0),
9                 grahamstown(1),
10                capetown(2),
11                ...
12               },
13              faculty CHOICE
14               {
15                science NULL,
16                commerce NULL,
17                arts NULL
18               } OPTIONAL,
19              }
```

In the above notation, "Students" is the lop-level field or element. This field is of ASN.1 type SEQUENCE OF, that is to say, an arbitrary number of repetitions (or zero) of what follows the SEQUENCE OF. In this case "Students" is defined as a SEQUENCE OF "Student" (i.e., a sequence of elements of type student). Notice how we can add semantic meaning to the syntactic notation of our ASN.1 structures by choosing appropriate field names.

"Student" (line 2) is defined as being a SEQUENCE (not SEQUENCE OF) of elements (an ordered list of types whose values will be sent on the line to the central DB, in the given order). The first field or element in the "Student" structure is called "name" (line 3) and is defined as a GeneralString. GeneralString is an ASN.1 type that identifies all general string values.

The next field or element is the "age" field (line 4) and is defined as an INTEGER, a number representing the age of the student. This is followed by the "id" field (line 5) which is of type NumericString.

"Address" (line 6) is of type ENUMERATED. Use of this type name requires that it be followed by a list of names for the enumerations (the possible values of the type). In ASN.1, you will usually find the name followed by a number in round brackets, as in this example. These numbers were required to be present up to 1994, but can now be automatically assigned if the application-designer so desires. They provide the actual values that are transmitted down the line to identify each enumeration, so if the "address" is "grahamstown", what is sent down the line in this field position is a '1'.

Last, the field called "faculty" (line 13) is of type CHOICE. The content of this field-position is one of a number of possible alternatives. In this case there are three possibilities: the "science" (line 15), "commerce" (line 16), and "arts" (line 17) alternatives.

The code fragment illustrated here is a not a complete ASN.1 module for the sake of clarity. The full ASN.1 module for this example can be found in section A.4.

Using the *asnparser* compiler, introduced in section 4.5, we generated the C++ header (.h) and source (.cxx) files that were included in a test application to build, encode and decode messages for this example. The C++ program that was used to build, encode and decode messages for this example can also be found in A.5.

The following three fictitious students' information was built and encoded using the test application:

Information	Student 1	Student 2	Student 3
Name	Jason Penton	Dominic Parry	Dylan Swales
Age	24	22	23
ID	7712295135084	7712295135084	7712295135084
Address	Grahamstown	East London	Cape Town
Faculty	Science	Commerce	Arts

Table A.1 Student Information for ASN.1 Encoding

This information was encoded using ASN.1 PER (Packed Encoding Rules). If these were the only three students at Rhodes University, the resulting octet string of representing this information, transported to the central DB would be a mere 80 bytes long and look as follows:

```
03 80 0c 4a 61 73 6f 6e 20 50 65 6e 74 6f 6e 2e
30 88 23 3a 62 46 19 54 04 01 00 80 0d 44 6f 6d
69 6e 69 63 20 50 61 72 72 79 2a 30 88 23 3a 62
46 19 50 04 01 40 80 0c 44 79 6c 61 6e 20 53 77
61 6c 65 73 2c 30 88 23 3a 62 46 19 58 04 01 40
```

The central DB machine would be able to decode the bit stream using the same ASN.1 specification described above. All the information regarding our three students would be rebuilt and inserted in to the central DB for later use. As you can see ASN.1 is a powerful notation that has use in a wide range of communication applications.

A.4 ASN.1 Example Specification

```
DEMO DEFINITIONS AUTOMATIC TAGS ::=
BEGIN

IMPORTS
Comment FROM COMMENT;

Rhodes-students ::= SET OF Card

Card ::= SEQUENCE {
 name    GeneralString,
 age     INTEGER (1..100),
 id      NumericString (SIZE (1..100)),
 address ENUMERATED
   {
    eastlondon(0),
    grahamstown(1),
    cape_town(2)
   },
   ...,
 faculty CHOICE
   {
    science NULL,
    commerce NULL,
    arts NULL
   } OPTIONAL
}
END
```

A.5 Example PWLib Application Demonstrating the Use of ASN.1

```
#include <ptlib.h>

#ifdef __GNUC__
#define H323_STATIC_LIB
#endif

#include "main.h"
#include "version.h"
#include "demo.h"


#define new PNEW

PCREATE_PROCESS(SimpleH323Process);



//////////////////////////////////////////////////////////////////
```

```
MyPWLibProcess::MyPWLibProcess()
  : PProcess("ASN.1 Demo Application", "Demo",
             MAJOR_VERSION, MINOR_VERSION, BUILD_TYPE, BUILD_NUMBER)
{
}

MyPWLibProcess::~MyPWLibProcess ()
{
}

void MyPWLibProcess::Main()
{
 PPER_Stream myStream;


  DEMO_Card::DEMO_Card myCard;
  DEMO_Card::DEMO_Card myCard1;
  DEMO_Card::DEMO_Card myCard2;

 DEMO_Rhodes_students rhodesStudents;


  DEMOOLD_Rhodes_students oldRhodesStudents;

  myCard.m_name = "Jason Penton";
  myCard.m_age = 24;
  myCard.m_id = "7712295135084";
  myCard.m_address.SetValue(DEMO_Card_address::e_grahamstown);
  myCard.m_faculty.SetTag(DEMO_Card_faculty::e_science);
  myCard.IncludeOptionalField(DEMO_Card::e_faculty);

  myCard1.m_name = "Dominic Parry";
  myCard1.m_age = 22;
  myCard1.m_id = "7712295135084";
  myCard1.m_address.SetValue(DEMO_Card_address::e_eastlondon);
  myCard1.IncludeOptionalField(DEMO_Card::e_faculty);
  myCard1.m_faculty.SetTag(DEMO_Card_faculty::e_commerce);

  myCard2.m_name = "Dylan Swales";
  myCard2.m_age = 23;
  myCard2.m_id = "7712295135084";
  myCard2.m_address.SetValue(DEMO_Card_address::e_cape_town);
  myCard2.IncludeOptionalField(DEMO_Card::e_faculty);
  myCard2.m_faculty.SetTag(DEMO_Card_faculty::e_commerce);

  rhodesStudents.SetSize(3);

  rhodesStudents[0] = myCard;
  rhodesStudents[1] = myCard1;
  rhodesStudents[2] = myCard2;

  myStream.BeginEncoding();
  rhodesStudents.Encode(myStream);
```

```
  myStream.CompleteEncoding();

  //here I am going to start the decoding procedures

  myStream.ResetDecoder();

  if (!rhodesStudents.Decode(myStream)){
     cout << "error with decode" << '\n';
     return;
  }
  cout <<endl << endl << endl << "encoded stream is as follows : "<< endl <<
endl << endl << endl;;
  myStream.PrintOn(cout);
  cout << endl;
  cout << endl << endl << endl << "Decoded asn structures are as follows : "<<
endl << endl;

  rhodesStudents.PrintOn(cout);

}
```

Appendix B: Miscellaneous ASN.1 Specifications

B.1 Complete ASN.1 Specification of H.323-UU-PDU

```
H323-UU-PDU ::= SEQUENCE
{
h323-message-body   CHOICE
{
setup            Setup-UUIE,
callProceeding   CallProceeding-UUIE,
connect          Connect-UUIE,
alerting         Alerting-UUIE,
information      Information-UUIE,
releaseComplete  ReleaseComplete-UUIE,
facility         Facility-UUIE,
...,
progress         Progress-UUIE,
empty            NULL,
status           Status-UUIE,
statusInquiry    StatusInquiry-UUIE,
setupAcknowledge SetupAcknowledge-UUIE,
notify           Notify-UUIE
},
nonStandardData  NonStandardParameter OPTIONAL,
...,
h4501SupplementaryService    SEQUENCE OF OCTET STRING OPTIONAL,

h245Tunneling    BOOLEAN, h245Control   SEQUENCE OF OCTET STRING
OPTIONAL,
nonStandardControl           SEQUENCE OF NonStandardParameter
OPTIONAL,
callLinkage                  CallLinkage OPTIONAL,

tunnelledSignallingMessage   SEQUENCE
{
tunnelledProtocolID          TunnelledProtocol,
messageContent               SEQUENCE OF OCTET STRING,
tunnellingRequired           NULL OPTIONAL,
nonStandardData  NonStandardParameter OPTIONAL,
...
} OPTIONAL,
provisionalRespToH245Tunneling          NULL OPTIONAL,
stimulusControl              StimulusControl OPTIONAL,
genericData      SEQUENCE OF GenericData OPTIONAL
}
```

B.2 ASN.1 H.245 IndicationMessage for DTMF Transport

```
IndicationMessage       ::=CHOICE
{
        nonStandard     NonStandardMessage,
        functionNotUnderstood   FunctionNotUnderstood,
        masterSlaveDeterminationRelease
        MasterSlaveDeterminationRelease,
        terminalCapabilitySetRelease
TerminalCapabilitySetRelease,
        openLogicalChannelConfirm       OpenLogicalChannelConfirm,
        requestChannelCloseRelease      RequestChannelCloseRelease,
        multiplexEntrySendRelease       MultiplexEntrySendRelease,
        requestMultiplexEntryRelease
RequestMultiplexEntryRelease,
        requestModeRelease      RequestModeRelease,
        miscellaneousIndication MiscellaneousIndication,
        jitterIndication        JitterIndication,
        h223SkewIndication      H223SkewIndication,
        newATMVCIndication      NewATMVCIndication,
        userInput       UserInputIndication,
        ...,
        h2250MaximumSkewIndication      H2250MaximumSkewIndication,
        mcLocationIndication    MCLocationIndication,
        conferenceIndication    ConferenceIndication,
        vendorIdentification    VendorIdentification,
        functionNotSupported    FunctionNotSupported,
        multilinkIndication     MultilinkIndication,
        logicalChannelRateRelease       LogicalChannelRateRelease,
        flowControlIndication   FlowControlIndication,
        mobileMultilinkReconfigurationIndication
        MobileMultilinkReconfigurationIndication

}

UserInputIndication     ::=CHOICE
{
        nonStandard     NonStandardParameter,
        alphanumeric    GeneralString,
        ...,
        userInputSupportIndication      CHOICE
        {
                nonStandard     NonStandardParameter,
                basicString     NULL,
                iA5String       NULL,
                generalString   NULL,
                ...
        },
        signal                  SEQUENCE
        {
                signalType      IA5String (SIZE (1) ^ FROM
                ("0123456789#*ABCD!")),
                duration        INTEGER (1..65535) OPTIONAL,
                rtp                     SEQUENCE
                {
                        timestamp       INTEGER (0..4294967295) OPTIONAL,
                        expirationTime  INTEGER (0..4294967295)
```

```
                                        OPTIONAL,
                        logicalChannelNumber
LogicalChannelNumber,
                        ...
                } OPTIONAL,
                ...,
                rtpPayloadIndication    NULL OPTIONAL
        },
        signalUpdate    SEQUENCE
        {
                duration        INTEGER (1..65535),
                rtp             SEQUENCE
                {
                        logicalChannelNumber
LogicalChannelNumber,
                        ...
                } OPTIONAL,
                ...
        },
        extendedAlphanumeric    SEQUENCE
        {
                alphanumeric    GeneralString,
                rtpPayloadIndication    NULL OPTIONAL,
                ...
        }
}
```

B.3 ASN.1 Specification for H.460.3 Number Portability Feature

```
NUMBER-PORTABILITY DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
 IMPORTS
  PublicTypeOfNumber,
  PrivateTypeOfNumber,
  AliasAddress
 FROM H323-MESSAGES

NumberPortabilityInfo ::= CHOICE
{
 numberPortabilityRejectReason  NumberPortabilityRejectReason,
 NUMBERPORTABILITYDATA SEQUENCE
 {
  addressTranslated NULL OPTIONAL,
  portedAddress    PortabilityAddress OPTIONAL,
  routingAddress   PortabilityAddress OPTIONAL,
  regionalParams   RegionalParameters OPTIONAL,
  ...
 },
 ...
}
NumberPortabilityRejectReason ::= CHOICE
{
 unspecified   NULL,
 qorPortedNumber NULL,
 ...
}

PortabilityAddress ::= SEQUENCE
{
 aliasAddress    AliasAddress,
 typeOfAddress    NumberPortabilityTypeOfNumber OPTIONAL,
 ...
}

NumberPortabilityTypeOfNumber ::= CHOICE
{
 publicTypeOfNumber  PublicTypeOfNumber,
 privateTypeOfNumber PrivateTypeOfNumber,
 portabilityTypeOfNumber  PortabilityTypeOfNumber,
 ...
}

PortabilityTypeOfNumber ::= CHOICE
{
 portedNumber   NULL,
 routingNumber   NULL,
 concatenatedNumber NULL,
 ...
}
```

```
RegionalParameters ::= SEQUENCE
{
 t35CountryCode   INTEGER(0..255),
 t35Extension   INTEGER(0..255),
 variantIdentifier INTEGER(1..255) OPTIONAL,
 regionalData   OCTET STRING,
 ...
}

END
```

Appendix C: Telgo323 – An H.323 Bridge for Deaf Telephony

Telgo323 was developed during the research as an H.323 prototype bridge that can relay text and speech between a Teldem, a text telephone for the deaf, and a standard telephone or H.323 endpoint. Telgo323 uses modified H.323 media gateways and open-source text-to-speech (TTS) and speech-to-text (STT) software. The approach allows for easy integration of new tools as the technologies mature. This appendix presents the design of the implementation prototype, discusses Teldem tone decoding, and suggests directions for future work. Telgo323 provides evidence that an automated relay bridge for the deaf community might not be too far in the future.

C.1 Introduction

In South Africa, Telkom provides a device called Teldem (Figure C.1) that enables deaf people to communicate with one another over the Public Switched Telephone System (PSTN). The Teldem is essentially a text telephone, an asynchronous device that allows two communicating parties to type to each other in real-time. The major drawback of this system is that there is no mechanism for a deaf person, using a Teldem, to communicate with any other party who does not have a Teldem. This means that if a deaf person wishes to call a hearing person, the hearing person is required to have a Teldem, which allows the hearing person to converse with the deaf user using text.

Telgo323 is a prototype that will bridge the standard PSTN telephone and the Teldem, thereby allowing users of the telephone (source and sink of audio) to communicate with users of the Teldem (source and sink of text). This bridge would be responsible for

enabling the two devices to communicate by providing media conversion between the two.

Figure C.1 Telkom's Teldem

I have partially implemented a prototype consisting of two H.323/ISDN (Integrated Services Digital Network) gateways extended to provide support for Teldem communication. The name Telgo323 derives from **Tel**dem **go**es H.**323** after the H.323 IP signalling layer [22, 23, 25] responsible for the communication between the two gateways on the IP network.

Essentially, this bridge is required to convert voice, originating from the telephone, into a textual equivalent that can be presented to the Teldem user. Similarly, the text originating from the Teldem needs to be converted to the voice equivalent and presented to the telephone user. These two scenarios can be implemented using speech-to-text (STT) and text-to-speech (TTS) technologies, respectively. However, STT technology is not yet mature enough to be used in this system because:

- The users of the system are unable to train the STT software.
- The audio output from the telephone may not be of a high enough quality for successful speech recognition. Reasons for this include a considerable amount of noise, like echo, in telephone conversations; and frequency limitations.

As a result, I decided to implement the system in such a way that more improved STT systems could be integrated into the bridge as they emerge. This means that the quality of the bridging system would improve as STT systems become more mature. The same holds for TTS tools. Thus, the system is designed to allow for *plug and play* of both STT and TTS tools.

The bridge remains a work in progress. However, enough was accomplished during the project to justify a report and this appendix provides technical information regarding the system's implementation to date.

Two research papers that discuss the work done that lead to the implementation of Telgo323 are [12] and [13].

C.2 Telgo323 Architecture and Operation

The bridge is implemented using two modified H.323/ISDN gateways. The system may have been implemented using a single, more sophisticated gateway capable of determining the nature of the originating and terminating devices (Teldem or telephone), but for simplicity it has been implemented using two.

Using two gateways means that there are two separate numbers to dial for the service, depending on the nature of the originating device. For example, if the deaf caller were using a Teldem, s/he would dial 6038000, the telephone number of the Teldem gateway. Once connected, the Teldem user is prompted for the number of the remote telephone to dial. After the user has entered the number, the Teldem gateway establishes a call to the telephone via the second gateway, called the telephone gateway. This gateway relays the voice call to the hearing user who may use a PSTN telephone, a cell phone or an H.323 terminal such as NetMeeting.

A similar sequence of events occurs if the caller is a telephone user, with the exception that the gateway number is different (6038001) and that the hearing user is presented

with a voice message requesting her/him to enter the number of the remote Teldem with which s/he wishes to communicate. This number is entered on the user's telephone and transmitted to the telephone gateway as a series of DTMF tones. On receipt of the number, the telephone gateway establishes a call with the Teldem via the Teldem gateway. From this point onwards, until the connection is dropped by either party or by the network, the gateways are working together to provide the communication link between the Teldem and the telephone. The architecture of the bridging system is illustrated in Figure C.2.

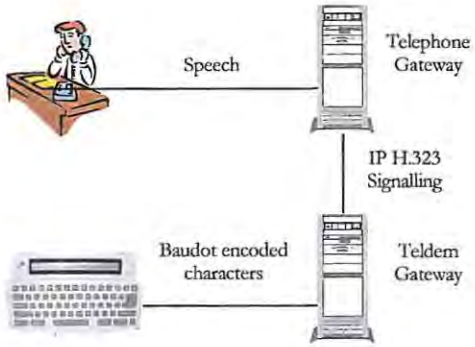

Figure C.2 Telgo323 Architecture

In addition to providing the telephone user with an interface into the IP network, the telephone gateway converts the audio it receives from the user into its textual equivalent. The text message is sent to the Teldem gateway using the H.323 *UserInput* capability. This mechanism allows the telephone gateway to send a string of characters, in this case the text message to be sent to the Teldem, to the Teldem gateway via standard H.323 messages.

In the reverse direction, the telephone gateway expects G.711 encoded audio from the Teldem gateway via an H.323 real-time protocol (RTP) channel. This encoded audio is decoded into raw pulse-code modulated (PCM) audio and sent via the PSTN to the user's telephone.

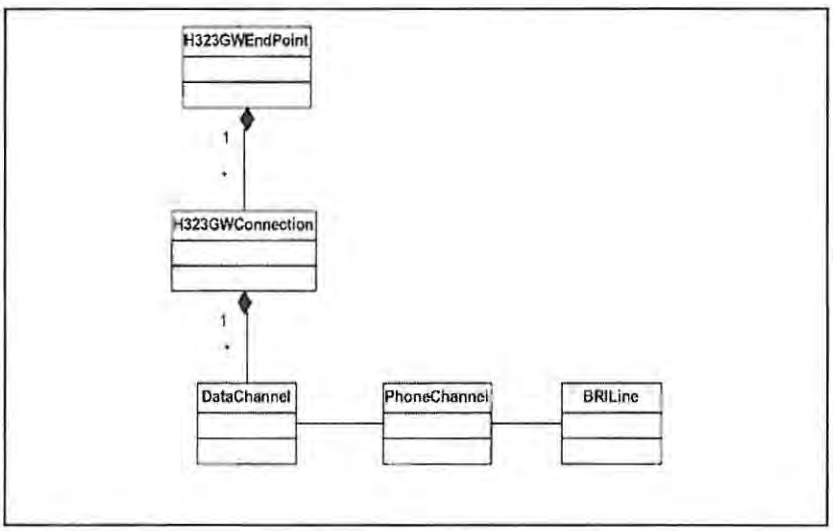

Figure C.3 UML Representation of Unmodified H.323-ISDN Gateway (For simplicity, the attributes and operations of the various classes are omitted unless relevant to the discussion.)

C.2.1 Telephone Gateway

To test the performance of the latest STT systems, I conducted a simple experiment that accepts voice from a microphone connected to a soundcard, streams the voice to a STT engine and sends the resulting text across an IP connection to a remote endpoint. One user dictated to the terminal with the microphone and STT engine while another user sat at the remote terminal and attempted to read what the dictating user was saying. The

STT quality was disappointing, and so I turned my attention to the Teldem gateway in anticipation of improvements in the STT quality. I expected the telephone gateway implementation to be quite straightforward, as the architecture closely resembles the Teldem gateway, without the complication of tone decoding. The eventual implementation should allow for *plug and play* of various STT tools.

C.2.2 Teldem Gateway

The Teldem gateway provides Teldem devices with an interface into the IP network. In addition, the gateway converts the incoming tones, representing text from the Teldem, into the voice message equivalent. The voice is encoded as a G.711 audio stream and sent to the telephone gateway via an H.323 RTP audio channel. The telephone gateway relays this audio to the telephone user on the PSTN.

In the reverse direction, the Teldem gateway awaits text messages, transported within H.323 *UserInput* messages, from the telephone gateway. These text messages are finally converted to a Baudot-encoded binary format and modulated for transmission to the Teldem via the PSTN.

C.3 Telgo323 Software Architecture

I chose the H.323 signalling protocol to build this bridge because H.323 is a mature standard and it integrates well with the PSTN. In addition, we already had a reliable H.323/ISDN gateway [47] that could be extended with Teldem communication functionality. As shown in Figure C.2, H.323 signalling occurs between the two gateways, each one hosting the service logic pertaining to its function.

The gateways used in this system were built using the open-source H.323 library provided by Equivalence PTY, called OpenH323 [39]. These gateways were not built from the ground up, but are instead modified H.323/ISDN gateways. Originally, the H.323/ISDN gateway enables telephones on the PSTN and H.323 terminals on the IP network to engage in voice communications with each other. By modifying two

separate instances of this gateway, I hoped to create a system that would enable telephones and Teldems (both on the PSTN) to communicate with each other via the IP network. The IP network is responsible for housing the logic that converts the communications into appropriate forms compatible for each of the two devices (voice for telephones and text for Teldems).

The Unified Modelling Language (UML) representation of the standard voice gateway without the Teldem modifications is illustrated in Figure C.3.

The root class of this gateway is *H323GWEndPoint*,c which represents the main thread of execution for the gateway. A single instance of this class is created when the gateway is started. Within the *H323GWEndPoint* object there may be any number of *H323GWConnection* objects, limited of course by the number of hardware interfaces into the PSTN. The class *H323GWConnection* encapsulates all the data and operations pertaining to a single connection to the gateway. When a Teldem connects to the gateway, an instance of this class is instantiated, including the instantiation of the audio channels used to carry the audio data. These audio channels are encapsulated within the DataChannel class that is responsible for sending and receiving audio data to or from H.323 terminals on the IP network. Two instances of the DataChannel class exist for each connection, one for outgoing or encoded audio (from the telephone to the H.323 terminal) and one for incoming or decoded audio (from the H.323 terminal to the telephone). These channels are attached to a specific codec class, (e.g., G.711) that is responsible for encoding or decoding the audio at the gateway and H.323 terminal, respectively.

The BRILine class represents the actual ISDN Basic Rate Interface (BRI) line. This class embodies all the data and operations required to interact with the PSTN via an ISDN connection, for example reading and writing raw PCM audio to or from the line or detecting an incoming connection. The PhoneChannel class provides the link between the DataChannel and BRILine classes. The PhoneChannel class maintains the state of the BRILine (connected, dialing, ringing, etc.) as well as provides the

DataChannel class access to read or write audio to or from the ISDN line via the BRILine class.

Let's consider a simple example scenario. Assume that the gateway has established a call between a PSTN telephone and an H.323 terminal on the IP network and that all the necessary objects have been instantiated. Consider audio originating at the telephone and terminating at the H.323 terminal.

First, the PhoneChannel object will know that it is connected and that audio is being transmitted between the gateway and H.323 terminal. When audio arrives on the ISDN BRI line, it is stored in a buffer within the BRILine object. The PhoneChannel object continuously removes audio from this buffer and passes it to the DataChannel object. The audio at this stage is in raw PCM format and it is encoded by the codec object attached to the DataChannel object. The encoded audio is then transmitted within an RTP stream to the H.323 terminal on the IP network where it is decoded by the codec object attached to the DataChannel object in the H.323 terminal. Once decoded, the audio can be presented to the user by sending it to the output port of the soundcard.

Standard H.323-ISDN Gateway Modification

Incoming audio from the Teldem on the ISDN BRI line is not standard voice data, but rather a sequence of characters encoded and modulated as a series of tones. These tones must be demodulated and decoded into the series of characters they represent before being sent to the telephone gateway.

Text messages arriving from the telephone gateway need to be encoded character by character, using Baudot encoding, and modulated to produce the tones that can be transported via the PSTN to the terminating Teldem device.

Figure C.4 illustrates the decoding portion of the Teldem gateway. This portion of the gateway was implemented with the addition of two specialized classes, the ToneDecoder class and the TTSSynthesizer class. The ToneDecoder class is defined as a thread that continuously attempts to fetch audio tones from the PhoneChannel (audio received from the Teldem via the BRILine object). The TTSSynthesizer class encapsulates the TTS system, which currently is the Festival TTS system [3]. The TTSSynthesizer class wraps standard functions around the TTS system, allowing us to substitute easily other TTS tools in the future. The TTSSynthesizer class also has methods that allow the ToneDecoder class to send it text strings that are synthesized to create voice message equivalents.

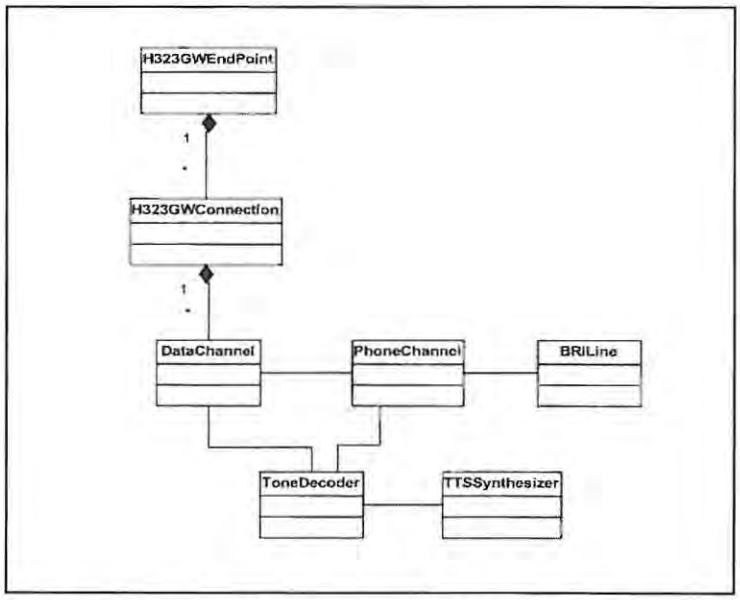

Figure C.4 UML Description of the modified H.323/ISDN Gateway

The tones received in the ToneDecoder object are demodulated and decoded to obtain the characters sent by the connected Teldem. A series of these characters is collected in the ToneDecoder object until a sentinel is received. I defined the sentinel as the uppercase character sequence "GA" for 'Go Ahead' (as this is how turn-taking is traditionally regulated during text telephony). Once the sentinel is received, the

ToneDecoder object calls on the TTSSynthesizer object to synthesize a voice message equivalent of the received text string. This voice message is passed to the DataChannel object where it is ultimately sent to the telephone user via the telephone gateway.

This part of the overall bridge has been fully implemented. The encoding portion of the Teldem gateway will be similar to the decoding portion just described. The differences are that the TTSSynthesizer class is replaced with a STTRecognizer, and the ToneDecoder class is replaced by a ToneEncoder class. The STTRecognizer class wraps STT tools just as the TTSSynthesizer class wraps TTS tools. The ToneEncoder class is defined as a thread that continuously checks for text strings arriving from the telephone gateway (the text equivalent of the speech received from the telephone user). These strings are then encoded character by character using Baudot encoding before they are modulated as a series of tones that can be sent to the Teldem via the PSTN. As stated earlier, the poor state of TTS technology to date limits the usability of this portion of the bridge.

C.4 Decoding Teldem Tones

The Teldem communicates across the telephone lines using frequency shift keying (FSK) modulated signals. These signals can be found in the high-frequency (HF) bands and are commonly known as Radio-Tele-TYpe (RTTY). In RTTY signals, a given frequency is used to transmit a logic '1', and a different frequency is used for a logic '0'. We call these frequencies MARK & SPACE, respectively. In the Teldem, the MARK frequency is 1388.25 Hz and the SPACE frequency is 1815.4 Hz. The data format is similar to those found in asynchronous serial communications: every character starts with a START bit with '0' logic value, followed by the data bits (least significant bit first), and finally one STOP bit with a logic level '1'. Inter-character time is filled with logic '1'. The character data bits are 5 bits long, and are obtained using an old method of encoding called BAUDOT [11].

The decoding process of these signals can either be done using hardware or software. Naturally, software decoding was used within the Teldem gateway. Software decoding of these signals is complex and beyond the scope of this appendix.

C.5 Future Work

The Telgo323 prototype opens the way toward a fully functional automated voice/text relay for the Deaf. The most important future work will be to complete the implementation of the telephone gateway for the bridge.

C.5.1 Complete Implementation of the Telephone Gateway

I have built a pre-prototype for the telephone gateway that can accept voice from a microphone connected to a soundcard, then streams the voice to a STT engine and sends the resulting text across an IP connection to a remote endpoint. Because the STT quality was, so far, disappointing, my attention turned to the Teldem gateway in anticipation of improvements in the STT quality. The telephone gateway implementation is relatively straightforward, since the architecture closely resembles the Teldem gateway, without the complication of tone decoding. The eventual implementation of Telgo323 will allow for "plug and play" of various STT tools.