Technical Report

UCAM-CL-TR-141
ISSN 1476-2986

Number 141

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Reliable management of voice in a distributed system

Roy Want

July 1988

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted December 1987 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Churchill College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

Summary

The ubiquitous personal computer has found its way into most office environments. As a direct result, widespread use of the Local Area Network (LAN) for the purposes of sharing distributed computing resources has become common. Another technology, the Private Automatic Branch Exchange (PABX), has benefited from large research and development investment by the telephone companies. As a consequence, it is cost effective and has widely infiltrated the office world. Its primary purpose is to switch digitised voice but, with the growing need for communication between personal computers, it is also being adapted to switch data. However, PABXs are generally designed around a centralised switch in which bandwidth is permanently divided between its subscribers. Computing requirements need much larger bandwidths and the ability to connect to several services at once, thus making the conventional PABX unsuitable for this application.

Some LAN technologies are suitable for switching voice and data. The additional requirement for voice is that the point-to-point delay for network packets should have a low upper-bound. The $10Mbs^{-1}$ Cambridge Ring is an example of this kind of network, but its relatively low bandwidth gives it limited application in this area. Networks with larger bandwidths (up to $100Mbs^{-1}$) are now becoming available commercially and could support a realistic population of clients requiring voice and data communication.

Transporting voice and data in the same network has two main advantages. Firstly, from a practical point of view, wiring is minimised. Secondly, applications which integrate both media are made possible, and hence digitised voice may be controlled by client programs in new and interesting ways.

In addition to the new applications, the original telephony facilities must also be available. They should, at least by default, appear to work in an identical way to our tried and trusted impression of a telephone. However, the control and management of a network telephone is now in the domain of distributed computing. The voice connections between telephones are virtual circuits. Control and data information can be freely mixed with voice at a network interface. The new problems that result are the management issues related to the distributed control of the real-time media.

This thesis describes the issues as a distributed computing problem and proposes solutions, many of which have been demonstrated in a real implementation. Particular attention has been paid to the quality of service provided by the solutions. This amounts to the design of helpful operator interfaces, flexible schemes for the control of voice from personal workstations and, in particular, a high reliability factor for the backbone telephony service. This work demonstrates the advantages and the practicality of integrating voice and data services within the Local Area Network.

Preface

I wish to thank Roger Needham, Ian Leslie, David Tennenhouse, Stephen Ades, and Roger Calnan for their valuable advice and discussions which enabled me to formulate my own research ideas. I am very grateful to Dan Craft for his time spent teaching me Concurrent CLU and the motivation he has given me for my work.

During the preparation of this dissertation I am grateful for the comments provided by Jean Bacon, Ian Wilson, Mike Burrows and especially Peter Newman's valuable pragmatic advice concerning simulating systems.

I also wish to thank all the members of the mechanical workshop, especially John Knight who put in many hours of work in constructing telephone hardware for the ISLAND demonstration system.

Throughout the time I have been in Cambridge I have had tremendous support from many friends. I would particularly like to acknowledge Sue Cock, Farrokh Poorooshasb, Dan Craft, Michael Seckl, Ninette Premdas (and for excellent proof reading), The Adventure Society: Nick, Reb, Paul, Marion; and also Flat #42 and the Canoe Club for providing many a distracting evening's fun.

I would like to make a special note of thanks to my parents for the support and encouragement that they have given me throughout my education. I sincerely hope that my Father's current battle against his own Cancer is successfully won.

The ISLAND project has also received considerable support from the British Telecom Research Laboratory at Martlesham Heath, for which the members of the project are thankful. Bill Bunn, Bob Turner and Alastair Rogers are also acknowledged for their help and liaison.

During the period of research I received a grant from the Science and Engineering Research Council and I am grateful for that support. Throughout my fourth year of research I am extremely grateful for the financial support arranged by Andy Hopper — Director of research at the Olivetti Research Laboratory, Cambridge.

Except where otherwise stated in the text, this dissertation is the result of my own work and is not the outcome of work done in collaboration.

I hereby declare that this dissertation is not substantially the same as any I have submitted for a degree or diploma or any other qualification at any other university.

I further state that no part of my dissertation has already been, or is being currently submitted for any such degree, diploma or other qualification.

Roy Want 1987

List of Chapters

Contents

List of Figures

Glossary of Terms

BBP	Basic Block Protocol
BSP	Byte Stream Protocol
BCPL	A language used to implement the Tripos operating system
CCITT	International Consultative Committee for Telephony and Telegraphy
CFSM	Call Finite State Machine
CFR	Cambridge Fast Ring
CFSM	Call Finite State Machine
CODEC	Coder–Decoder (commonly used for voice digitisation)
CLU	An object-based language developed at MIT
CMDS	Cambridge Model Distributed System
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CR	Cambridge Ring
CSMA/CD	Carrier Sense Multiple Access with Collision Detect
DAC	Digital to Analogue Converter
DES	Data Encryption Standard
DTMF	Dual Tone Modulated Frequency
ECHD	Establish, Connect, Hold, Delete interface
ECL	Emitter Coupled Logic
EPROM	Erasable and Programmable Read Only Memory
FIFO	First In First Out
ISDN	Integrated Services Digital Network
ISLAND	Integrated Services Local Area Network Development
LAN	Local Area Network
LED	Light Emitting Diode
MAN	Metropolitan Area Network
MTBF	Mean Time Between Failures
MTTR	Mean Time To Repair
PABX	Private Automatic Branch Exchange
PCM	Pulse Code Modulation
RAM	Random Access Memory
ROM	Read Only Memory
RPC	Remote Procedure Call
SSP	Single Shot Protocol
STP	State Transition Procedure
STSP	State Transition Subprocedure
TASI	Time Assignment Speech Interpolation
TDM	Time Division Multiplexing
VCO	Voltage Controlled Oscillator
VLSI	Very Large Scale Integration
VME	Bus Standard Recognised by IEEE P1014 and IEC 821

Chapter 1

Introduction

In the early stages of telecommunications engineering, communications networks were constructed from analogue circuit-switched systems. In the last decade network designs have been moving towards digital packet-switching techniques. This change has come about through the development of fast integrated technologies which have been applied to specific problems in the telecommunications industry. This revolution brings with it a host of advantages. One important advantage is flexibility: it is possible to switch digital data, voice, and video within the same network, and to offer network services which integrate these media in novel applications. If voice and data are to be integrated in real-time, then the digital network must guarantee information transport, for which the criteria of low delay and low error-rate are necessary for the respective media (real-time video has both requirements). Some Local Area Networks (LANs) have properties which satisfy both of these criteria, an example of which is the Cambridge Ring, see chapter 3.

Private Automatic Branch Exchanges (PABXs) have followed this technological trend although, in general, they have remained centralised switches. With the increasing use of Local Area Networks for the interconnection of personal computers in business, it has become viable to integrate the PABX into the same network, and hence to provide the traditional services alongside many new ones. Although technologically feasible, the design and organisation of such an integrated system presents some new and interesting problems.

1.1 Aims of this Research

This thesis investigates the design of a PABX as a distributed system, and the problems of providing the reliable service expected of telephone systems in common use today. In order to investigate these research issues, the ISLAND project was established in 1983. Its long-term goals were to investigate the provision of Integrated Services on a Local Area Network. The project was undertaken by the University of Cambridge Computer Laboratory, with financial support from British Telecom Research Laboratory.

Initial work involved the design and replication of voice digitising equipment in the form of network telephones. As a first step towards the development of an environment that allowed applications integrating voice and data, ISLAND set out to build a backbone PABX making use of the services already available within the Cambridge Distributed Computing System. One of the main criteria for the design was that it should provide a quality of service surpassing that provided by existing systems. For that reason, considerable thought was put into the 'human interface' and the mechanisms for providing control of voice from application programs. A particular consideration in this research was the attention paid to designing a reliable system.

A telephone service is expected to be highly reliable: in fact a typical private exchange supporting 200 extensions has a specified Mean Time Between Failures (MTBF), for a total loss of service, equal to 17 years (a 10% loss of service is specified to have a MTBF of 8 months) [BT 83]. A better description of the service is that it is highly available. If clients are going to consider new approaches to telephony as a serious alternative to conventional equipment, it is important to guarantee the reliability and availability of the alternative services. In a distributed PABX, voice is not switched by a centralised exchange; instead it is distributed by the addressing structure of the network. The operation of an *Exchange* in a LAN/PABX becomes a management issue. If the *Exchange* fails, existing voice connections continue to operate correctly, but changes in the connective state are no longer possible. This property allows for an inherently larger recovery time since it is only the clients who are modifying their connections that become aware of the failure. The reliability issues are discussed fully in chapters 6 to 9; these include:

- Replication of exchange components
- Distribution of control
- Distribution of state
- Fault detection
- Fault confinement
- Reconfiguration
- Recovery of state
- System re-start

The backbone PABX was designed to be usable in its own right, but could also incorporate modular functions in the form of services that could be incrementally added to the distributed system. These services could be made as reliable as the designers intended them to be, but in the event of their failure the PABX would ensure a graceful degradation down to a reliable telephony-service. Clients writing application programs should be able to control resources of the PABX,

namely telephones, and make any newly-designed services readily available by a simple update to the system directory. In this way separate voice and data services could be integrated to provide future multimedia applications.

1.2 Background

The work behind this thesis has been carried out in an environment shaped by two major projects undertaken at the University of Cambridge Computer Laboratory:

- The Cambridge Model Distributed System (CMDS)
- Project Universe (in collaboration)

In order to explain the philosophy behind some of the design decisions made (and the terminology that has grown up with this computing culture), the following background sections have been provided.

1.2.1 The Cambridge Model Distributed System

In 1978 the University of Cambridge Computer Laboratory began a research project to design and build a distributed computing system centred around a Local Area Network. The LAN design was also a laboratory project and became known as the Cambridge Ring [Wilkes 79]. The raw data rate for the original ring implementation had an upper limit of 10Mbs^{-1} and was quite suitable for the load and applications envisaged at that time.

The resulting implementation is usually referred to as the Cambridge Model Distributed System (CMDS) [Needham 82]. This system is characterised by using the LAN to separate a client from the machines he or she is working on, and is in direct contrast to a popular alternative model which uses the network to link together a large number of client workstations. In general, the latter approach does not use the network for sharing computing power, but instead for inter-workstation communication and the sharing of expensive resources, such as a filing system or printer. In the Cambridge model a client's interface is a terminal attached to a *Terminal Concentrator* [Ody 84] that can handle eight terminal connections while only requiring a single network-interface. The concentrator establishes a virtual connection between it and remote processors using a protocol known as Byte Stream Protocol (BSP) [Johnson 81]. The remote processors are unusual in that they have a network interface as their only peripheral. Processors available for general use make up a heterogeneous collection of machines called the *Processor Bank*. Many of the processors used are based around a 68000 [Motorola 82] microprocesser utilising an intelligent network interface called a MACE [Garnett 83].

A number of services were identified to be essential for managing the system, for instance to provide Naming, Resource Management, Authentication, and Loading

functions across the network. These services were established as modular system-components in dedicated machines known as *Small Servers*. These *Servers* were constructed around Z80 microprocessors and resulted in a cost effective and reliable implementation, primarily due to their lack of complexity. This kind of system partitioning allowed the development of modular and reliable software, enforced by the rigid network service-interfaces.

Naming

Naming is an important issue in network communication. The CMDS uses a static *Name Service* which maps names to network addresses and *vice versa* (a function referred to as 'Reverse Lookup'). This kind of indirection allows machines to be detached from fixed network addresses. In addition, it later became an important part of the mechanism by which a *bridge*[1] [Leslie 83] could be used to extend the topology and addressing capabilities of the Cambridge Ring.

Resource Management

Resource Management is a necessary strategy for allocating limited heterogeneous-resources amongst a large population of clients [Craft 85]. In CMDS a client does not request a particular machine for an application, but instead requests the name of a system that he or she requires. A set of attributes may also be given (for example, memory size and type of processor) to ensure that the needs of the application are met. The request is sent to the *Resource Manager* (RM) and, in response, the RM finds the most suitable machine to meet the needs of the request, or denies it if the request cannot be fulfilled. The RM then initiates the loading of the required system into the allocated machine. A client gains control of the machine by a virtual connection set up between the two by RM.

Loading

In order to hide the loading characteristics of a particular processor type, a server called an *Ancilla* is associated with each type. The *Resource Manager* will know which *Ancilla* to use in association with any particular type of processor-bank machine. An additional server called the *Boot Server* is also available for general loading but its primary task is to provide a mechanism for re-booting the static services.

There are many other services available which include: the *Active Object Table* (AOT) [Girling 83], the *Logger*, *Date Server*, various *Print Servers*, and the *File Server* [Dion 81], descriptions of which are not appropriate in this text.

---

[1] The original Cambridge Ring needed to be partitioned into two separate rings when the number of stations had risen in excess of 60. The partition was motivated by the large number of stations leading to a low point-to-point bandwidth, and an increasing instability in the clock phase-locking mechanism of each station. A ring-ring *bridge* was designed to allow the two separate networks to be connected together.

1.2.2 Project Universe

In 1982 the Universe (Universities Extended Ring Satellite Experiment) project [Leslie 84] was established as an experiment to connect a number of Local Area Networks at geographically different sites. Universe was a collaborative effort between British Telecom Research Laboratory, Cambridge University, GEC-Marconi Research Laboratories, Logica Ltd, Loughbourgh University of Technology, Rutherford Appleton Laboratories and University College London. The LANs used in this venture were exclusively $10Mbs^{-1}$ Cambridge Rings and the inter-site connection was a $1Mbs^{-1}$ satellite broadcast-channel.

A European Space Agency Orbital Test Satellite (OTS) provided the broadcast channel, each site having access to a ground station. In the Universe network a connection between two subnets was called a *bridge*. If the ring subnets existed at the same site it was referred to as a ring-ring bridge, but if the connection was through a satellite, it was simply termed a satellite bridge. Since all inter-site communication had to travel through the same satellite bridge there were no routing problems to consider.

The principal aim was to design a composite network in such a way that a host's view of communication across the subnets was identical to communication over a single ring. A noteworthy feature was the lack of an inter-network protocol layer. The protocols supported by Universe were lightweight and included: Universe datagrams, an extended version of the Cambridge Single Shot Protocol (SSP) [Gibbons 80], and lightweight virtual-circuits with no flow-control or error checking. The resulting flexibility also made it a test bed for the development of real-time voice protocols which have their own particular requirement of low transport-delay. Before Universe it had been claimed that LANs were capable of supporting integrated service traffic (for example, real-time voice and video, combined with data file-transfer). Universe was one of the first experiments to provide a practical demonstration.

Work on Universe finished in December 1983. To a large extent it was the inspiration of the ISLAND project. However, a more direct descendant was the Unison Project.

1.2.3 The Unison Project

Unison [Clark 86] the 'Child of Universe' was inspired by the work of Universe and that of the Cambridge Fast Ring (CFR) [Hopper 86]. The project set out to provide a land-based network suitable for the provision of integrated services over a number of remote sites and, for this reason involved many of the collaborators originally involved in Universe. Unison is part of the Alvey research programme which is still actively pursuing its research.

Unison adopted a similar virtual-connection strategy to that of Universe but the physical architecture was based on a new model. The long-promised $100Mbs^{-1}$ CFR had been seen in a number of different lights. It could be used as a service

ring in its own right or a fast packet switch confined within a 19-inch card-rack. To that end, Unison saw it as a means of interconnecting a variety of lower bandwidth networks, such as 10Mbs^{-1} Rings and 10Mbs^{-1} Ethernets, and using it as the centre of a hub configuration providing centralised packet-switching for a particular site. In order to connect the hub rings together 2.048Mbs^{-1} links (e.g. British Telecom's Megastream) were used to provide the terrestrial inter-site connections. This kind of link is very cost effective and is in direct competition with satellite technology for providing high-bandwidth communication. As fibre technology becomes more pervasive, it is increasingly likely that satellite technology will be less widely used in national communications.

Megastream divides up its bandwidth into time-slots that can be allocated to client services. This kind of bandwidth sharing is known as Time Division Multiplexing (TDM). Time-slots are chosen so that standard 64kbs^{-1} voice streams can be sent 8-bits per slot, with the next time-slot available after 125μs. At 2.048Mbs^{-1} this allows 32 x 64kbs^{-1} streams to be multiplexed on to the link. The network is therefore suitable for data and voice, and other synchronous traffic requiring a larger bandwidth division, since a number of consecutive slots could be allocated to a single application. As a result of the CCITT 'I-Series' recommendations [ITU 85], this class of network link has become the basis for ISDN (Integrated Service Digital Network) communications in Europe. The 'I-Series' recommendations go further to define the use of particular time-slots in a timing frame, and a signalling standard.

The CFR forms the basis of the *Unison Exchange* [Tennenhouse 87]. Connections of Megastream links are performed by components called *Ramps* and the local client networks are connected through *Portals*. Currently, most of the basic infrastructure for the network has been built.

Implications for the future include the ability to use services situated at remote sites as if they were available locally. These services may be data-oriented, relating to conventional distributed computing requirements, or else combined with audio-visual applications. The Unison project could, in the future, be used to share integrated services of the type investigated by this thesis in the environment of the ISLAND project.

1.3 Overview

This thesis is structured according to the following plan.

Chapter 2 is a survey of systems in the research and commercial world. A categorisation has been made which follows the advancing trends in PABX design into the field of distributed computing.

Chapter 3 is a description of the approach taken by the ISLAND project to provide the required functionality of a PABX integrated into the well established Cambridge Model Distributed System.

Chapter 4 describes the design issues in building a packet-voice telephone and interfacing it with a $10Mbs^{-1}$ Cambridge Ring. Lessons have been learned from this work concerning the division of hardware and software for the processing of real-time voice. The chapter concludes with a proposed design for future network-interfaces for voice.

Chapter 5 discusses a model for the ISLAND *Exchange* and its basic non-replicated operation. Chapter 6 extends this basic model to a system in which control is distributed amongst an arbitrary number of replicas. The implementation model has been influenced by the tools and development environment available to the ISLAND project, and is described in that context. The *Exchange* has been designed and implemented to provide a variety of features with future extensibility in mind. A mechanism for reliably passing control of network telephones to other network services is also presented.

Chapter 6 outlines the issues concerning reliability and availability when implementing an *Exchange* service. The discussion explains the decision to distribute the service over many machines and the issues that needed to be considered in order to implement this type of distributed application. The solutions ISLAND has adopted are presented in contrast to the various models already in existence for resilient distributed-computing.

Chapter 7 describes how faults are determined within the *Exchange*, a method for reconfiguring the system in the event of a failure, and how system state may be replicated and recovered after a server failure. A novel use of a 'Virtual Ring Protocol' for fault monitoring and the distribution of system state is also described in some detail.

Chapter 8 is a description of the interfaces and functions needed to manage the distributed *Exchange* service. It was necessary to build a tool to manage a generalised set of machines. This tool runs as a separate network service and provides an access point by which the distributed PABX can be controlled, debugged and monitored in normal operation.

Chapter 9 is an evaluation of the design and protocols used in the ISLAND project. 'Service delays' and other operational statistics taken from the ISLAND system, are presented and used to assess how the system would scale for use in a real application.

Chapter 10 summarises the points learned from the project and concludes with an assessment of its potential applications.

1.4 Extent of Collaboration

The practical work undertaken by the ISLAND project is extensive and involves collaboration of several members of the University of Cambridge Computer Laboratory. The overall architectural philosophy for ISLAND was designed by Stephen Ades [Ades 87b], who carried out much of the ground work for the construction of server infrastructure. The design of the Voice Provider (chapter 3) was the joint

work of Roger Calnan and Stephen Ades. The *Translator*, also mentioned in chapter 3, was entirely designed and implemented by Roger Calnan. A special note of thanks is given to all members of the laboratory who helped in porting existing code from the CMDS to the ISLAND servers, and to those who helped debug these systems for use in ISLAND applications: Ian Leslie, David Tennenhouse, Roger Calnan, Ian Wilson and Robert Cooper.

All work concerning the design of network telephone hardware, its control software, and the design and implementation of the ISLAND *Exchange* service can be attributed to the author.

Chapter 2

The Evolving Private Automatic Branch Exchange (PABX)

With the advent of modern solid-state digital-switching in the 1960's, the expensive telephone exchange, once only feasible as a shared central resource, could be distributed in the form of the Private Automatic Branch Exchange (PABX)[1]. In the last 20 years this technology has undergone evolutionary changes in its design and application. The catalyst for change has been the development of the microprocessor in the early 1970's. This led to the production of many small personal computers for business applications e.g. the office workstation and, as a result, many of the office tasks traditionally carried out by people could be automated. Despite the distribution of processing power there was still a need to share expensive resources such as printers, mass centralised-storage, and additionally a need for inter-workstation communication providing document transfer and electronic mail. Local Area Networks (LANs) were designed to meet these demands.

Telephony, an equally important tool in the modern office, had initially remained a province of the PABX. Moreover, the PABX designers were bending over backwards to adapt their systems for data transfer as well as voice traffic [Ellis 84], but they could only offer throughput at a rate similar to, and often less than, $64kbs^{-1}$ voice channels. In the early 1980's LANs generally offered data rates between $3Mbs^{-1}$ and $10Mbs^{-1}$ [Wilbur 85], and now LANs are being designed to provide data rates as high as $100Mbs^{-1}$ [Hopper 86] [Ross 86] with potential for still faster data rates in the future. Computer equipment could therefore transfer data at a rate which was in a totally different league to the conventional PABX, even when the network access protocol and bandwidth sharing scheme is taken into consideration. The important difference between the bandwidth utilisation of a LAN and a conventional PABX is that a PABX has a fixed fraction of the switch bandwidth allocated to each client, whereas a LAN divides the network bandwidth between the clients who actually require it. A better evolutionary path would be to use the LAN to carry voice traffic. LAN systems combining PABX and data facilities

[1]The PABX is sometimes more simply called the PBX, but this term can be misinterpreted to mean a manually operated exchange. The term PABX will be used throughout this thesis.

are now beginning to evolve [Rossi 84] [Kay 83] [Greenway 85].

The following sections refer to examples of systems demonstrating advances in PABX design. The classification used is purely an invention to illustrate the methodology of voice/data systems with increasing technological merit as judged by this text. However the economics of the current time defines a different ordering on the basis of their commercial viability.

Section 2.1 reviews a first generation PABX that carries only voice traffic. Section 2.2 gives examples of second generation PABXs that allows voice and data applications to be connected to separate extension lines; third generation systems (section 2.3) use a centralised token-bus to switch voice and data, a technique employed by a only a few commercial systems. A further advance comes with the merging of two technologies to form the hybrid PABX/LAN. Section 2.4, fourth generation PABXs, describes how voice and data are switched by their separate technologies but fuse together at the call-processing phase by making the PABX management processor a station on the LAN. Fifth generation PABXs (section 2.5) have made the complete transition by integrating voice and data within a LAN at the bit-level, and also in the applications that they support.

2.1 First Generation PABXs

These systems behave in the same way as traditional telephone systems but they incorporate additional control facilities which are accessible through codes dialled at conventional voice equipment. Switching of voice is carried out through a centralised private-switch which has facilities that allow exchange lines from the public network to be connected to it. These systems are becoming increasing popular for small and large businesses alike and, because the technology is becoming very cost effective, they are rapidly becoming a standard feature of the modern office.

A typical PABX is the Monarch 120 Call Connect System [GEC 81]. This system is designed around a line-shelf architecture (see figure 2.1). A line-shelf will usually contain 8-line cards, each of which support four extensions. A line-shelf thus supports 32-line circuits (8x4), each with a 64kbs^{-1} voice channel, transferring data as one 8-bit sample every $125\mu\text{s}$. The 64kbs^{-1} voice streams are produced from an analogue signal by the use of a Coder/Decoder (CODEC) which, as the result of telecommunications, is a common integrated component. Additional signalling information is carried by attaching a ninth bit to each sample. An extension data-rate is therefore 72kbs^{-1}. The 32 individual data-pathways are carried on a backplane which is terminated by a shelf multiplexer circuit. It is here that the voice and signalling information are separated again. The voice channels are then multiplexed to form a 32 time-slot Pulse Code Modulation (PCM) group generating a 2.048Mbs^{-1} data bus which feeds the input stage of a *Time Switch.* The signalling bits are multiplexed on to a separate 256kbs^{-1} stream and fed to the input of a unit which then decodes them.

Speech leaving the *Time Switch* also leaves on a 2.048Mbs^{-1} bus and is fed to a multiplexer accompanied by a 256kbs^{-1} control stream. The separate busses are

demultiplexed and combined to form a 72kbs^{-1} stream that is fed to each of the line circuits. The digital voice is then converted back into analogue form by a CODEC.

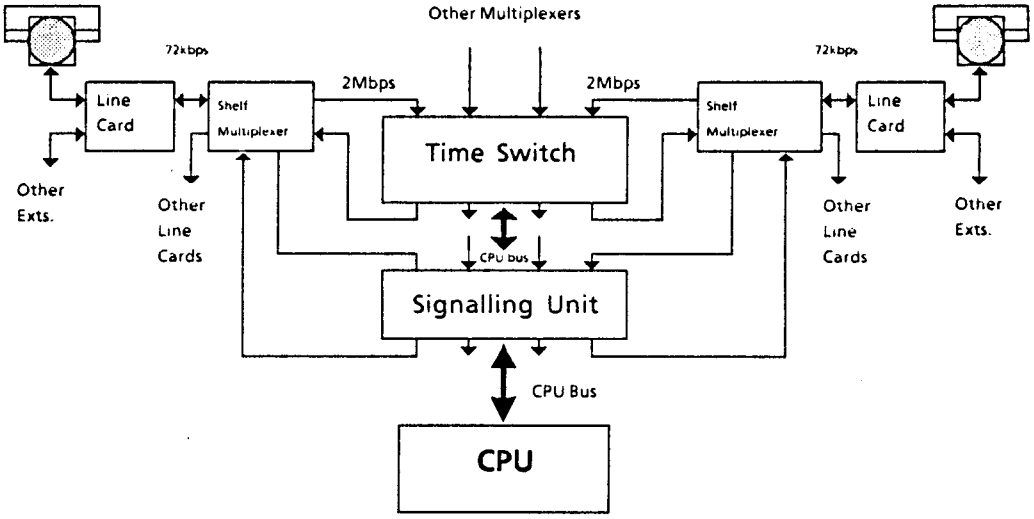

Figure 2.1: Schematic of a first generation line shelf PABX

2.1.1 Time Switches

The *Time Switch* is a voice-stream switch. It transfers samples between voice streams flowing into it and voice streams flowing out of it. A typical *Time Switch* will take eight 2.048Mbs^{-1} voice streams[2] and perform a serial-to-parallel conversion in such a way that an 8-bit 2.048Mbs^{-1} bus is produced with data interleaved and serialised by the line-shelf ordering. Each voice sample is then stored sequentially in a speech RAM (a 256-by-8 Random Access Memory) at an 8-bit location called a *port*. Key-presses coded on the 256kbs^{-1} signalling bus are captured by the system CPU at the start of a phone call and, as a result, data is written to a location in a second RAM, the connection store. This RAM maps a *port* in the speech RAM on to the corresponding time-slot and line-shelf from which it should be played out. By sequentially accessing the connection store using a counter and indirecting the data through the speech store, samples are recovered in the order required by the connection mapping. The resulting samples are passed through a parallel-to-serial converter and re-assembled back into eight 2.048Mbs^{-1} voice streams. The streams are then split up and fed to the appropriate line-shelf. The voice samples are now in the destination time-slots and hence will be played out

[2]The 2.048Mbs^{-1} bus conforms to the CCITT standard for time-slotted data transmissions [ITU 85].

at the destination extensions. The *Time Switch* can thus transfer voice in both directions independent of the number of the connections/ports in use at that time. It is termed a fully-available non-blocking switch.

Tone Generators, Conference Units, and Direct Dial lines will usually occupy one of the shelves and have ports reserved in the switch for their use. This does, however, restrict the number of extensions that the architecture can support. Clearly the connectivity is extensible by combining many *Time Switches* together, although there is a square-law relationship between the number of switches needed and the number of extensions that are to be handled.

2.2 Second Generation PABXs

Commercial pressure and competitiveness combined with client demands has meant that the ability to carry voice and data traffic, other than by using a MODEM, is a necessary requirement to sell a PABX in the current market place. The leading manufacturers, therefore, are providing this kind of facility, examples include Plessey's ISDX and Northern Telecom's SL-1. The design changes are minor since it is easy to carry data in the time-critical slots of a voice stream, providing that the error rate is acceptable. The modifications are associated with the line card, rather than the switch, and involve the addition of interface circuits which provide a standard data transfer protocol such as RS232 or V24. Typical line rates for such interfaces are 9600 baud, and thus a number of these could be multiplexed on to a single time-slot. The important characteristic of this generation of PABX is that data time-slots are permanently allocated and they use up slots that are normally available for voice.

2.3 Third Generation PABXs

Sydis corporation have developed a centralised PABX system using a $128Mbs^{-1}$ token bus architecture [Nicholson 83]. It is composed of many 68000 processors providing multiple services across the bus; these include: file storage, voice connection coordination, voice compression, workstation interface-modules, a standard PABX interface making it compatible with other systems, and a gateway to other Sydis systems across *Symnet*. Workstations are connected to the Sydis system by standard four-wire twisted-pair cabling and are terminated at an interface card. The wires are used to provide four $64kbs^{-1}$ links: two of which are used for data, another provides a voice link, and the remainder is used for signalling. Sydis workstations contain an integral telephone and keyboard in front of a compact display unit. The $128Mbs^{-1}$ bus is used to switch data and voice between the workstations and the storage/compression units. The system can achieve integrated voice and data functions but only between a set of its own homogeneous workstations. The bus offers some attractive advantages. Firstly, bandwidth is shared out on demand amongst bus contenders. Secondly, the system is easy to maintain and thirdly, it

uses cheap technology to switch data since the short backplane allows a parallel bus to be used and clocked at a rate an order of magnitude slower than the LAN equivalents. Moreover, it makes use of existing PABX type wiring.

The problem left unsolved by this project is the inability to connect to the many types of personal workstation already in existence. The outlay of buying sophisticated telephones without integrating any pre-existing workstations may not be practical. However, this kind of centralised approach tends to be favoured by telephone companies. AT&T have their own version of a token-bus PABX [Barrett 85] which uses a number of distribution boxes for linking to telephones and terminals. High-bandwidth fibre-links join these boxes to a centralised token-bus switch. As with all centralised systems so far considered, there are potential reliability problems which result from the system being dependent on one centralised component. In conclusion: the bus systems have good properties for sharing bandwidth but they do not have all the advantages of a totally distributed system.

2.4 Fourth Generation PABXs

These systems have made important contributions to PABX call processing, rather than in the actual voice switching techniques used. In first and second generation systems, call processing is centred around a processor of moderate capability, for example the Monarch 120 System uses an 8080 microprocessor. The processor is primarily concerned with decoding signalling information, keeping track of the state of a call (represented by a finite state machine) and managing the tables of a *Time Switch.* Often such a system is controlled by a program written in machine assembly code. New features require a reassembly of the source code and, furthermore, it is the proprietary information of the manufacturers. More recently, a few research-orientated systems have been built which explore the management and integration possibilities when a PABX switch is attached to a computer system that is running an operating system such as UNIX [Ritchie 74]. Examples are the Berbell and MICE systems being developed at Bell Laboratory's Morristown site.

2.4.1 The Berbell System

Berbell [Redman 87] is a user-programmable telephone-switch which combines a VAX 11/750 with a Modular Switching Peripheral (MSP) built by Redcom Labs. The MSP provides basic interfaces which allow dialled digits and other client-initiated events to be read from a digital interface. Voice connections may be established through the same interface. The advantage of separating out management and switching issues is made apparent by the small number of suitably chosen primitives that are necessary to access the MSP. Berbell contains a number of routines, written in C, which perform the interface function and these have been incorporated into an interpreted language called BERPS (BER Phone Script). These routines are run as the result of dialling in the usual way. The dialled numbers are mapped to a code reference by a simple table. This table can be

changed dynamically, allowing new features to be added without recompiling all of the code. The interpreter provides the client programmer with I/O commands. These include synthetic generation of speech, conditional statements, functional operations and execution control, in much the same way as conventional computer languages. Moreover, calls to the operating system allow access to conventional programming tools. The result is a flexible control language which can provide standard feature packages in the form of a library of scripts, and the ability for client programmers to write their own features tailored to a specific need. The advantages are even more apparent when the telephone dialling operations can be derived, either from the telephone, or from one of the VAX terminal lines, allowing an interface of much greater sophistication.

2.4.2 MICE

MICE [Woodbury 87], another project at Bell Morristown, uses a hybrid approach to control a Redcom switch. A number of client workstations are connected in a distributed environment by an Ethernet, and are linked to the switch by a serial control line. Each of the workstations runs a background task which co-ordinates control over the switch, but inter-workstation communication is carried out through the Ethernet. The primary aim of the research is to develop integrated services incorporating conventional office applications and the telephone system. One of the main research issues is to determine the type of human interfaces that are suitable for such a system.

Redman's approach used a completely centralised scheme for switching voice and exercising control over the system. It should be pointed out that this system represents an architecture that is at the opposite end of the spectrum to the one proposed in this thesis. In both the Bell systems, voice and data are not integrated at the Physical layer of communication and, as a result, it is not very easy to store them on a common medium for the purpose of manipulating them alongside each other.

The 4th generation design approach is not desirable because specialised communication equipment is required for each kind of medium. If real-time video were to be incorporated into the system, further specialised cabling would be necessary between the access points. Furthermore, centralised systems generally have a single-point dependency for the reliable operation of the service. If the VAX used by Berbell fails, so does the telephone system. The MICE project is clearly more resilient to component failures but the switch is a potential weak link in the system.

2.5 Fifth Generation PABXs

Fifth generation PABXs are characterised by their totally distributed approach and the use of high-speed digital-networks. Using high-bandwidth packet-switching networks has two well-known advantages:

- Network bandwidth is divided equally amongst requestors and thus is not wasted on idle clients.
- Silence suppression can make more efficient use of bandwidth.

Implementation difficulties are caused by:

- The increased complexity of the network interfaces having to cope with the higher data rates due to the integration of voice and data, and the serial nature of the transmission encoding.
- Buffering is needed to ensure smooth playback of digitised samples, and thus delay is introduced.
- A suitable access protocol must ensure minimal and relatively constant delays for voice applications.

The problems are solved by good design and increased technological support in the form of VLSI circuits and cheap bulk memory. The third difficulty is related to the choice of network architecture. Once these problems have been addressed, the benefits can be reaped. The work carried out at Xerox PARC (Palo Alto Research Centre) is the most significant in this field, and the most similar to the practical work (ISLAND) used to investigate voice management in this thesis. There are, however, some important differences between Xerox's approach and the approach taken by ISLAND. These differences will be emphasised in chapters 3 and 4.

2.5.1 Xerox's Distributed Telephone System

In 1983 Xerox PARC published a paper [Swinehart 83] describing an experimental telephone system built from distributed processors interconnected by an Ethernet. The project is still making a valuable contribution to the field today [Terry 87]. A key component is the server called an Etherphone[3], which performs digital-voice sampling and packet assembly/disassembly for the transfer of voice across an Ethernet. Control is achieved by using Remote Procedure Call (RPC) through the same Ethernet. Etherphone can operate in the same way as a normal telephone or in conjunction with an office workstation. The latter allows expansion of the user interface, and hence the provision of useful features which are easy to use. The interface also makes it possible for clients to write programs to control Etherphones for specialised applications. Xerox has also built a special purpose Voice File Server which can be used in conjunction with Etherphone to store and retrieve voice from a conventional disk. Experiments have also been carried out with voice incorporated into text documents. Editing of voice presented in this way has been achieved by the use of specialised software tools provided at a workstation [Ades 87a].

[3]Etherphone is a trade mark of the Xerox corporation.

2.5.2 Etherphone

The physical interface of the Etherphone resembles a conventional telephone but, unlike a telephone, it connects to a box containing both an Ethernet server and hardware suitable for most voice I/O applications. It can also be controlled remotely from a variety of different workstations supporting the Cedar programming environment [Swinehart 86] and RPC [Birrell 83]. Etherphone's processing power is limited to simple network-control functions and the transmission and reception of voice. This allows it to be small and economical enough to be made in the required quantities. A standard subscriber line interface to each Etherphone enables calls to be made to the outside world, and provides a backup in the case of system failures.

An Etherphone is based on a dual-processor architecture. Its main processor (an Intel 8088) controls a DES peripheral (Data Encryption Standard) for the encryption of voice, along with an Ethernet controller working at 1.5Mbs^{-1}, and an RS232 controller to provide local interfacing, for example when testing the server. The control program is coded in the C language and occupies 8K bytes of EPROM. The slave processor runs a highly-optimised assembly-language program to perform: real-time control of digital voice-samples, CODEC control, silence detection and the management of voice buffers. The master and slave processors communicate with each other through a 48K block of shared memory. Both processors are overlooked by a *Watchdog* timer providing checks on the integrity of the device and the detection of intermittent faults.

2.5.3 Xerox's Telephone Control Server

The *Control Server* is a non-replicated component in Xerox's distributed system which is responsible for handling Etherphone call-processing. It does not need to carry out any voice switching as this is performed by the addressing structure of the Ethernet. The *Control Server* is responsible for providing the intelligence of the Etherphone, and interpreting key-strokes at an Etherphone console. In response to valid sequences of keys it will set up or clear down virtual connections using the Etherphone RPC interface. Call accounting and statistics are also carried out here. The *Control Server* can access system databases such as directories, repositories of information relating client locations and the relative positions of workstations to Etherphones. By making use of this data, it can synchronise operations between workstations and Etherphones. Regular monitoring of Etherphones enables it to assemble a maintenance log for repair staff. The *Control Server* can automatically load and reboot Etherphones which is necessary when an Ethernet is powered on, but is also the first line of attack for correcting a failure that may have resulted from a transient error. In either case, an Etherphone's *Watchdog* will signal this event to the *Control Server* using the Ethernet broadcast mechanism.

2.5.4 Design Issues in the Xerox System

Due to the operational characteristics of an Ethernet, it is necessary to buffer voice into large packets. The Xerox voice protocol uses a packet containing 160 x 8-bit samples with an additional 36 bytes of packet overhead. For a $64kbs^{-1}$ voice stream the resulting transmission rate is 50 packets/second. If data users were also present on this network and running applications such as file transfer, it would not be possible to guarantee the time in which voice packets could be delivered. As a result the Etherphones, the Control Server, and the Voice File Server are connected to their own $1.5Mbs^{-1}$ Ethernet. A low speed $1.5Mbs^{-1}$ Ethernet is used in this prototype system to take advantage of a VLSI Ethernet controller in the design of an Etherphone. Access to existing services, and in particular workstations, is made possible by a *packet gateway* connected between the Ethernet used for voice and a $3Mbs^{-1}$ Ethernet (a part of the Xerox distributed system).

An interesting feature of their approach is the detection and suppression of transmitted silence. It has long been realised by telephone companies that by suppressing silence in digital-voice transmission, a greater utilisation of shared channel-bandwidth may be achieved. A conservative estimate of the increased utilisation is 1.6 times the channel bandwidth [Brady 68]. This factor is usually referred to a the TASI advantage (Time Assignment Speech Interpolation) [Bullington 59]. Using this value, combined with the 0.2 Erlang occupancy[4] likely to be found for a private office telephone, leads to the following calculation.

```
Calculation:

Voice sampling rate      = 64k bits-per-second
Voice packet size        = 160 bytes + 36 bytes (overhead).
Voice packet rate        = 50 packets-per-second
Using a 1.5M b/s  Ethernet, at worst case, 50% utilisation
             (to avoid collision problems)
  => 750k b/s  available data rate to avoid unacceptable delays.

Number of voice streams possible:
  = Bandwidth / (Packet rate x Bits per packet)
  = 750k / (50 x (196x8)) = 9.56

Taking advantage of statistically distributed pauses in speech
(TASI) gives a further advantage of 1.6 (see text).
  => 9.56 x 1.6 = 15.3
By considering a phone occupancy equal to 0.2 Erlang (see text)
  => int(15.3/0.2) => 76 subscribers or 38 two-way conversations
```

If a standard $10Mbs^{-1}$ Ethernet is used, a similar calculation can be made, which

[4]The Erlang [Bear 76] is a dimensionless unit of 'traffic flow' or 'occupancy'. A telephone with 0.2 Erlang implies that if it is monitored for an hour it will be in use for 12mins. This value is only the mean of an assumed distribution and the period may not necessarily be sequential. A more usual figure used by traffic flow calculations for public telephones is 0.1 Erlang.

indicates at least 28 simultaneous two-way conversations are possible. Using the TASI advantage combined with a 0.2 Erlang telephone occupancy, this figure implies that greater than 225 subscribers may be supported [Swinehart 83].

2.5.5 The Ztel System

Ztel were one of the first commercial ventures to step away from centralised Time Division Multiplexing (TDM) or linear-bus orientated switching-strategies. Their approach adopted the IEEE 802.5 token-ring standard [IEEE 83], integrating voice and data in a loosely-coupled distributed-system. A principal advantage of this standard is that a priority 'voice' (in preference to 'data') token can be established, ensuring the harmonious coexistence of the two media. Their system, called a Private Network Exchange (PNX), uses the programming methodologies found in a distributed system to control and synchronise telephone equipment with terminal activities [Kay 83]. High speed data applications may be connected directly to the ring: phones and terminals can be linked up to 1600m away. Inter-node bandwidth can be increased incrementally by connecting many rings in parallel; the architecture has been designed to make provision for up to forty rings combined in this way. An advantage of this design is that the alternative routes through the network increase the reliability of the system. There is, however, very little published information on the distributed control techniques adopted by the Ztel system.

2.6 Related Work

There are numerous other projects which have carried out work in this area, many of which have been concerned with the representation of voice in documents, and the human factors involved in the use of voice with systems. Some of the more well-known research projects include: 'Diamond' at BBN laboratories, New Jersey (US) [Thomas 85]; 'Minos' at Waterloo University, Ontario (Canada) [Christodoulakis 86]; 'MAGNET' at Columbia University, New York (US) [Lazar 85]; and various experiments at the multimedia laboratory of MIT Boston (US) [Schmandt 85]. Furthermore, there are a number of experimental LANs which are being developed for voice and data integration, for example: 'Fasnet' at Bell Labs, New Jersey (US) [Limb 82]; and 'Orwell' at British Telecom Labs (UK) [Falconer 85].

A major contribution in standardising Integrated Service Digital Networks (ISDN) has been made by the International Telegraph and Telephone Consultative Committee (CCITT) [Davidson 85]. In Europe the CCITT recommendations have been applied to the 2.048Mbs^{-1} networking standard [ITU 85] which uses synchronous Time Division Multiplexing (TDM). This type of network operates on the same principles used in conventional PABXs i.e. combining and transporting several 64kbs^{-1} data streams in time-slots carried by a higher capacity bus. Such slots can be allocated to voice connections and meet the requirements for its

transport delay. The importance of CCITT's contribution is that it has led to a standard for the customer service interface and to a proposed signalling standard [Robin 84]. However, the problems ISDN are trying to solve are oriented towards producing end-to-end voice and data communication in a wide area network. The mechanisms by which applications can integrate voice and data into new services is not the main goal.

The work in this thesis differs considerably from the examples above. The following chapters are concerned with the organisation and management issues that are necessary in order to design a PABX around a Local Area Network, primarily with a view to providing a reliable service.

2.7 Summary

Using the five generation classification for PABX design, it is clear that substantial effort is being put into systems to meet voice/data integration demands. The PABX and LAN technologies are beginning to merge but, as yet, an unresolved commercial battle remains between the centralised time-switch, the centralised linear-bus architectures, and the totally distributed systems. The centralised systems are much better researched than the distributed approaches because of their simplicity, cost effectiveness and longer existence in the commercial world. It is only in recent years that high-speed VLSI support for LAN implementation has made the distributed approach a more favourable proposition. The architecture and control issues of distributed computing systems are similar to those required in a distributed PABX and, as such, can be readily applied.

Chapter 3

The ISLAND Architecture

One of the aims of the ISLAND project was to investigate the design and architectural issues involved in the construction of a distributed PABX. In the following sections the key components of a distributed PABX are discussed, and a general architecture is proposed as a design solution from a research perspective. A great deal of this architecture has already been implemented and from now on this will be referred to as the ISLAND demonstration system.

3.1 The Network

To a large extent the properties of Cambridge Rings were responsible for the initial proposal of the ISLAND project. There are many other networks which can support integrated media in the local context, for example: FDDI [Ross 86] and IEEE 802.5 token rings [IEEE 83], TDM networks [Davidson 85] and Binary Routing Networks [Newman 88]. When ISLAND was proposed, the Cambridge Fast Ring (CFR) project predicted that it would have a $100Mbs^{-1}$ slotted ring available for use in a relatively short period of time. The high available-bandwidth would allow many multimedia applications to run concurrently. However, to avoid delays, the ISLAND project began its work using a $10Mbs^{-1}$ Cambridge Ring. Two $10Mbs^{-1}$ rings were already in considerable use in the laboratory, and were relied upon by a large population of research students. To increase the available facilities would be of benefit to them, and give the ISLAND services a significant test-bed. Initial development began using an additional $10Mbs^{-1}$ Cambridge Ring.

3.1.1 The Original Cambridge Ring

The $10Mbs^{-1}$ Cambridge Ring is an example of a slotted ring. It consists of a number of stations connected in a ring topology. A circulating logical-sequence of slots, each containing a *minipacket*, is established by a monitor station, in conjunction with the other stations in the network. A small gap between the first and last slot is necessary to distinguish the beginning and the end of the sequence. Each

minipacket contains 38-bits with the format shown in figure 3.1a. The full/empty bit is initially marked empty. Stations can transmit a *minipacket* by finding a free slot and then filling in the destination and source address, along with the 16-bits of data to be sent. The circulating slot passes through all the stations which, noticing the slot is full, compare the destination field with their own address. If they match the data is copied out. Furthermore, the receiving station will change the two *minipacket* response bits that follow the data. These bits indicate four types of low-level response: *accepted*, *busy*, *unselected*, and *ignored*. *Busy* indicates that the station contains data which has not been read from its receive buffer, a situation that occurs when the station's host processor is busy with another task. *Unselected* is a means of signalling that the station is currently receiving from another source and, therefore, the sender should delay before transmitting again. An ignored *minipacket* has the response bits unchanged and indicates that the destination is not operational and, most likely, is turned off. A transmitter can thus deduce which course of action is appropriate in each of these eventualities.

An important part of the design is that it is the transmitting station that must empty a slot that is used for transmission and not the recipient. In addition, the same slot cannot be used for two consecutive transmissions from the same station. The transmitter knows how many slots there are in the ring and counts the slots going by after making a transmission. It therefore knows which slot to empty and can pass this slot on to the next station in the ring. If an unusual situation arose in a one-slot ring, whereby all the stations wished to transmit a *minipacket* at the same time, the slot would be passed from station to station on a round-robin schedule.

Minipackets only contain 16-bits of data. A higher-level data link protocol is used to gather *minipackets* into a larger and more useful unit of data transfer, the *Basic Block*. This protocol is referred to as the Basic Block Protocol (BBP) [Walker 78]. The reception of *Basic Blocks* is serialised at a receiver by the use of a selection mechanism. The *unselected* response signal is used to enforce serialisation.

This design gives rise to the following network properties:

- **Bandwidth is shared out equally between equal requestors:** this property results from stations passing on a slot after every transmission. The available bandwidth is also independent of the offered load. This is in direct contrast to CSMA/CD systems which experience a higher probability of packet collision with large loads, and thus network utilisation decreases.

- **Low access delay:** *minipackets* are small packets of data that are passed between requestors, in the worst possible case, at the packet rate divided by the number of stations in the ring. In some Token Bus or CSMA/CD systems the packets are considerably longer, typically up to 1Kbyte, and are necessarily long to reduce the overhead of the packet preamble. It is therefore, in general, not possible to guarantee the transport time for a packet, and hence the requirements of voice cannot be met.

It can be concluded that if data users and voice services coexist on the same ring, each will have access to a slot with little delay. Moreover, the low-bandwidth voice

user's requests will be met in full, leaving the remaining bandwidth divided among the needs of the data users.

3.1.2 Suitability of the Cambridge Ring as a PABX

So far the number of stations in a ring have not been considered as a cause of significant point-to-point delay. However, an increase in the number of stations in the ring increases the point-to-point delay, and there will be a limiting number of stations that can be inserted into the ring while still guaranteeing the requirements of voice. A simple calculation can be made to find the limiting number of stations for standard $64kbs^{-1}$ voice streams.

```
Approximate calculation for a 10M bits-per-second ring:

Voice bit-rate                  = 64k bits-per-second
Ring bit-rate                   = 10M bits-per-second
Bit-latency per station         = 3  bits
Data bits-per-packet            = 16 bits
Bits per packet                 = 38 bits
Let N be the number of stations in the ring:

Maximum voice packet delay = data bits-per-packet/voice bit-rate
                           = 16/64k = 256 microseconds        ...(1)

                      (Bits per ring/ Ring bit-rate) x N
Transmission delay >  ----------------------------------  ...(2)
   (Maximum)                    (Number of slots)

                   >   (3N/10M) x N/ INT(3N/38)

 => N < 66      [ INT() is a function returning the integer value ]
```

This restriction puts a worst-case limit on the number of telephones to be 65. By taking advantage of the probabilistic nature of telephone calls, and by adding management to block calls when they exceed a critical number, more ring stations can be added. If half the voice users were replaced by data users in a 65 station ring, the data users could only obtain the same available bandwidth as the voice users. A typical real-life PABX/LAN would need to support between 100 to 200 extensions and data users. A $10Mbs^{-1}$ Cambridge Ring cannot support a load of this size.

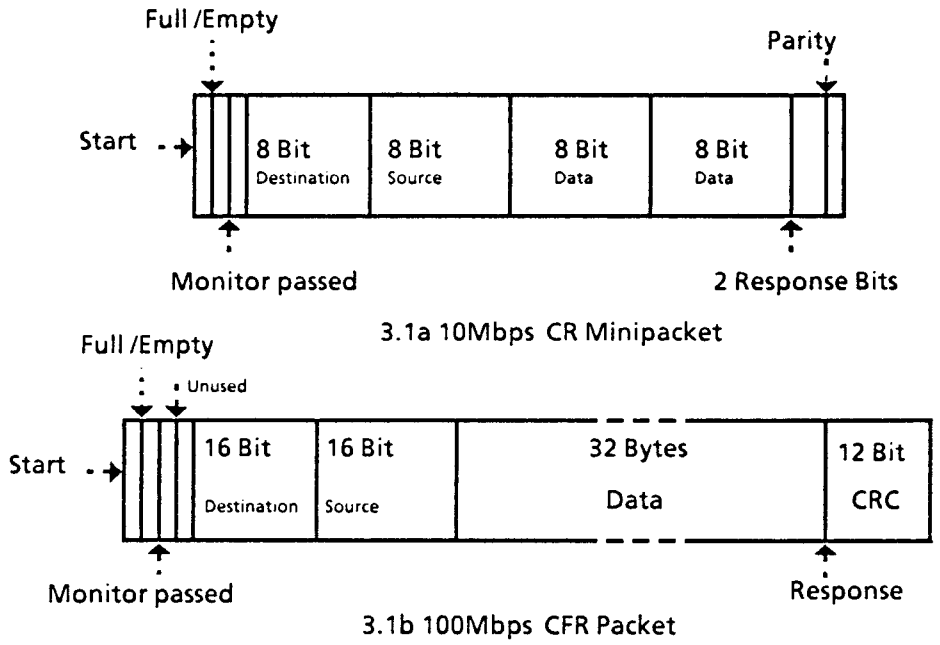

Figure 3.1: Cambridge Ring packet formats

3.2 The Cambridge Fast Ring

It had always been the intention of the ISLAND project to develop its environment on the Cambridge Fast Ring (CFR) [Hopper 86]. However, the CFR project was hindered by problems that delayed the fabrication of a VLSI chip-set until early 1986. As a result the ISLAND demonstration system was constructed on the original $10Mbs^{-1}$ ring, but designed for the characteristics of the CFR with a view to transferring the system as soon as it became available. A description of the CFR packet is shown in figure 3.1b. One of the criteria on which ISLAND based its design is that bandwidth is plentiful in a $100Mbs^{-1}$ Local Area Network, and thus a realistic population of voice and data users are able to coexist. The calculation made for the $10Mbs^{-1}$ ring can be repeated for the CFR.

```
Ring bit rate                = 100M bits-per-second
Bits per station latency     = 30  bits
Data bits-per-packet         = 256 bits
Bits per packet              = 304 bits

Voice packetisation delay    = 256/64K = 4 milliseconds    from (1)

    (30N/100M) x N / INT(30N/304)  < 4 milliseconds       from (2)

=> N < 1311
```

This figure implies that 655 (1310/2) full duplex conversations are possible, using 1310 stations. Of course this figure is an impractical number of stations to have in a single ring. However, it does imply that 200 extensions could easily be supported. In pragmatic terms this would probably be implemented by a ring with 50 or less stations in which several telephones were multiplexed on to one station. In this situation there would still be plenty of bandwidth available for data users (note that this calculation assumes no silence suppression in the voice streams and no blocking of subscriber calls).

A small number of stations in a ring is, in fact, a necessary requirement as there are other factors limiting the number of stations (relating to the problems of phase-locking the clocks at each network node). Every station introduces a certain amount of bit-delay jitter, and this will become an unstabilising factor above a critical number of stations. The original $10Mbs^{-1}$ Cambridge Ring developed in the laboratory began to experience locking problems when it contained between 50 and 60 stations.

The CFR was designed as a kit of parts [Temple 84] which can be assembled into stations, monitors, or packet bridges. Packet bridges allow the network to be extended to other CFRs using a 16-bit global addressing structure. The network could therefore be extended to a topology utilising many inter-connected rings, each ring containing only a small number of stations. The resulting network has different properties for voice and data depending on the topology used. Acceptable delays may be ensured by providing different routes, in the form of specialised bridges, for voice and data. These problems are being considered under a separate research proposal [Porter 86].

3.3 Network Reliability

The ring networks have a high degree of reliability for transferring data. A typically measured figure for the $10Mbs^{-1}$ Cambridge Ring is one bit in 10^{11} bits lost due to noise effects [Temple 84]. This is an ideal medium for conventional data transfer. When transporting voice, such a high degree of network integrity is not so important. Small volume errors do not degrade voice by a significant factor. In fact a packet of a voice stream with a duration as long as 4ms can be lost one percent of the time without significant reduction in the clarity of a client's speech [Gruber 83]. This size of packet is equivalent to a CFR packet that is completely filled with voice samples. This property makes it possible to use the CFR packet as the basic unit for a protocol between two real-time devices transferring voice which do not need, or have time, to acknowledge reception of packets. This is because an occasionally corrupted packet can simply be thrown away. A more acute problem is network availability. The Cambridge Rings are research tools, and as such, have only been built to a tolerance acceptable in a research environment. In the experience of the ISLAND project the mean time between failure (MTBF) is about one month. The mean time to repair (MTTR) can be up to a day. Most computer users expect this kind of disruption every now and again, but

a telephony service is expected to be highly available.

The network failure problem can be solved by using redundant links between stations. There are two well known ways to do this.

- **A Braided Ring:** one in which every nth station is connected to the (n+1)th and (n+2)th, allowing failed stations or links to be bypassed if a failure is detected (see figure 3.2). The scheme can be extended further to cope with two adjacent failures by extending links to (n+3)th stations. If this approach is taken to its logical conclusion there is no difference between a totally connected topology and a fully braided ring. As always, a compromise must be made between cost, reliability and pragmatics [Falconer 83].

- **A Self-Healing Ring:** a popular redundancy model using two parallel counter-rotating rings (see figure 3.2). The two rings connect the same set of stations. This model allows full utilisation of both rings in the fully operational state, and so twice the normal bandwidth is available. If a single link fails, one ring is still intact and availability is maintained. If two links fail in the same segment (occurring in the case when the ring is severed), the two stations adjoining the fault contain hardware which bridge the two rings together. Thus a new ring is created which is twice as large but excludes the failed segment. The mechanism also recovers from a single station failure, or any number of adjacent station failures, by excluding them in the same way as described for a segment. Non-adjacent station failures result in network partitioning. If voice users are present, the recovery mechanisms should allow them to continue to access a useful subset of the network. An example of a 'Self-healing' ring is the FDDI Ring [Ross 86] currently being proposed as a standard for Metropolitian Area Networks (MANs) in IEEE 802.6.

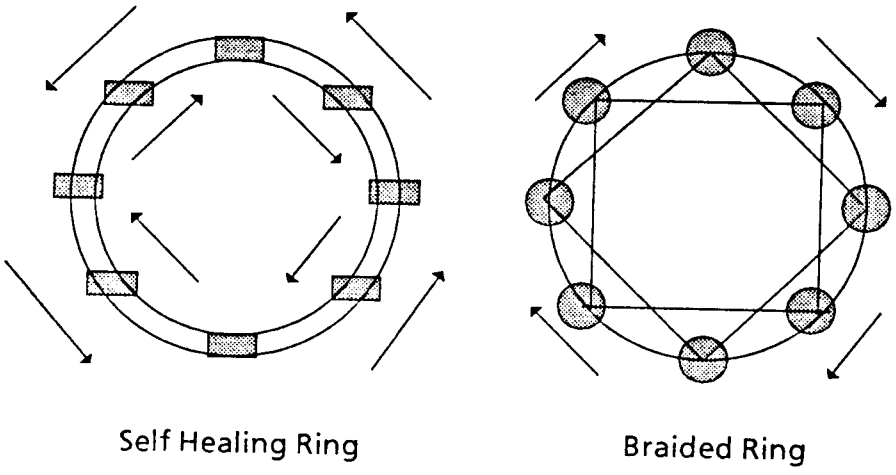

Figure 3.2: Two types of fault tolerant ring network

The Ztel system described in chapter 2 is a further example of redundant interconnection. In this system an arbitrary bandwidth and reliability factor can be

obtained by connecting rings in parallel.

It is important to realise that the gains in reliability (an elusive parameter to estimate before implementation), are offset by the complexity of their solution. A complex recovery strategy will have its own kind of failure modes. Self-healing rings have a significant recovery mechanism because failure of a station executing a recovery will generally be isolated at that station, causing the ring to recover at an adjacent station along the ring structure.

3.4 Division of Network Services

In the Cambridge Model Distributed System (CMDS), network services were partitioned into separate servers. ISLAND has adopted a similar approach to take advantage of modularisation and fault confinement, already demonstrated to be of value in CMDS (see figure 3.3).

3.4.1 Telephones

A detailed description of ISLAND telephones is given in chapter 4. The summary given here outlines the design principles. *Ringphone* is the ISLAND name for its network telephone. *Ringphones* are, in principle, simple system components containing a network interface, a processor, and a peripheral responsible for digitising voice. A network service-interface allows simple operations to be activated remotely, examples of which are:

- Relay key-presses to a particular station
- Send voice to station named **p** and receive voice from station named **q**
- Generate a tone
- Display some text

In an attempt to provide a more suitable human interface than that which is normally provided by the telephone, a single-line text-display has been attached to its front panel. The design is functionally simple in order to reduce the number of failure modes it may have. It is also the only system component which is replicated in large numbers in a real system, possibly 100 to 200 times. Any incremental cost must therefore be multiplied by a comparatively large number and must be justifiable. Because ISLAND is purely a research project, the hardware used for implementing the *Ringphone* is much more elaborate than it needs to be. The *Ringphone*, as it stands, cannot be easily streamlined because it is designed around a network server which is general enough to develop all the ISLAND network services.

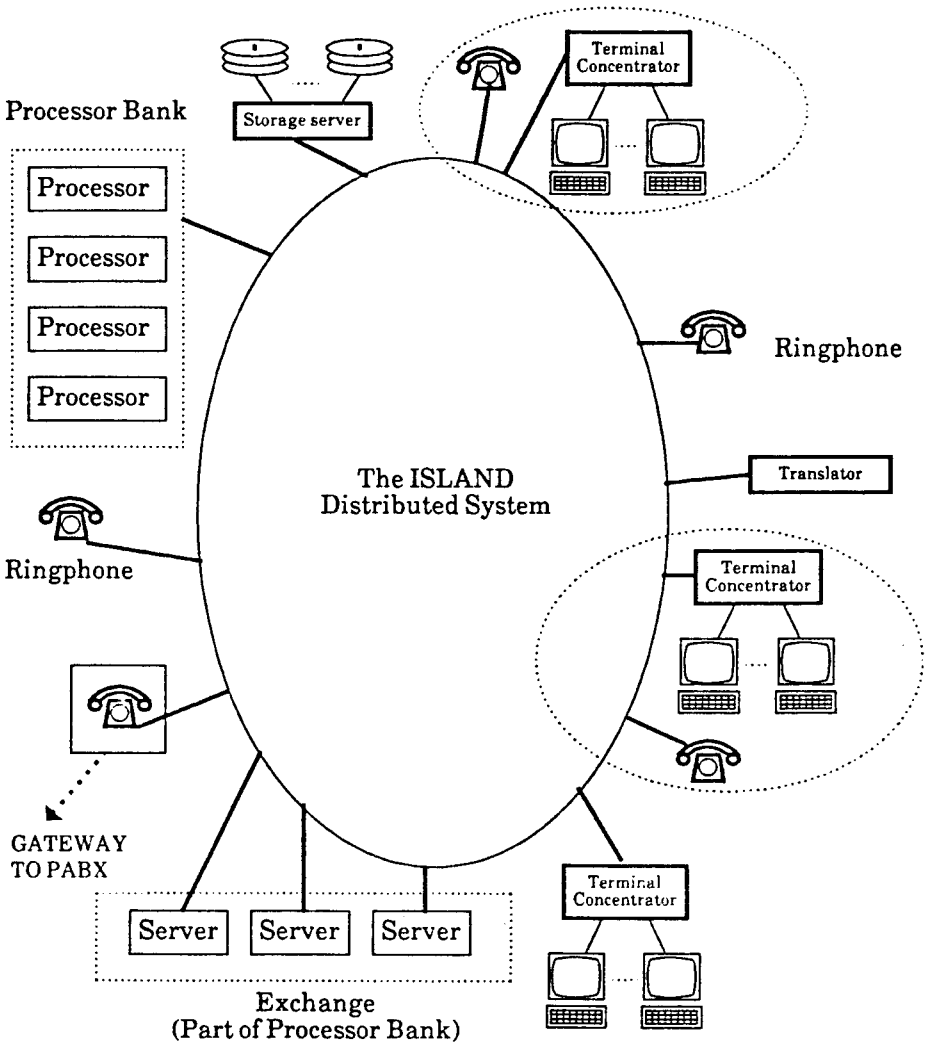

Figure 3.3: The ISLAND architecture

3.4.2 The Exchange Service

This service is the focus of *voice management* in the ISLAND system. The design criteria already stated is that a distributed PABX must provide reliable operation. Various scenarios for a redundantly linked network have been proposed, but a detailed study is not part of this work. The voice interface has been made simple because it is the only economic way to reduce the possibility that the interface will fail. However, *Ringphones* are replicated by a large number and as with conventional telephones, if one fails, there is likely to be a working one nearby. The important issue is that the exchange service should not fail. The remaining system dependency therefore lies in the ISLAND *Exchange Service*. This is an area well suited for distributed programming techniques. Chapters 5 to 9 are devoted to the design of this service and are previewed below.

The only way to construct highly-reliable systems out of unreliable components is to replicate critical components and critical data, or to add information coding,

so as to remove any single-point dependency. Hardware and software can be replicated for the *Exchange Service* in order to provide availability, but service integrity can only be achieved by distributing any dynamic 'state' created during the system's operation. For a specific application it is possible to tailor a state distribution mechanism to be optimal in relation to the level of state recovery required. A basic design for the service is explained in chapter 5. The design is taken further, as explained in chapter 6, to allow it to be distributed and highly available. In chapter 7 a mechanism is described that would allow it to recover critical-state in the event of a server failure. Finally chapter 9 evaluates this design for use in a system of a realistic size.

The *Exchange Service* is also designed to allow access to a number of other modular services, for example, a *message leaving service* or a *conference service.* A generalised mechanism for passing control of a *Ringphone* to other services enables fixed and experimental services to be accessed. A fail-safe solution is adopted to ensure that *Ringphones* will always be able return control to the highly available *Exchange Service*, and then relinquish control from the external service.

3.4.3 The Conference Service

Most modern PABXs have conference facilities. This feature can also be achieved in a distributed architecture. If a network has a broadcast facility, a voice conference can be achieved by arranging that the parties involved send their samples to the broadcast address. Hardware support at the telephones could then be used to combine voice packets, thus establishing the conference. The approach used in ISLAND keeps the telephone operation simple and, to this end, was not seen as the correct place to implement the conference function. In order to allow *Ringphones* to use just one voice protocol, an additional network module, the *Conference Server*, has been specified. Conferencing for a group of n Ringphones, (where n> 2), could be achieved as the result of the *Exchange* service instructing the n *Ringphones* to send their voice streams to a *Conference Server.* This server would perform a summing operation[1], and generate n new streams, each containing the sum of all the voice streams except the one originating from the *Ringphone* it was destined for. The last condition is used to prevent any echo in the system. A look-up table could be used to perform the summing function and would preferably be implemented as part of the loaded service, rather than using a specialised ROM. It is important to make services independent of specialised hardware so that a wide choice of alternative machines can run the application when the host server fails. Moreover, if a service is becoming overloaded, more general-purpose servers can be loaded with the service in order to meet the system's demands.

[1]Due to the logarithmic coding function used in standard A-law encoded $64kbs^{-1}$ voice streams [CCITT 72], the summing of voice samples is not straightforward arithmetic. Because of the speed required in this real-time application, a look-up table is needed to perform this operation.

3.4.4 Voice Message Services

Public telephone systems, and most PABXs, do not have a voice messaging service as an integral part of the system. It is far more common to have separate automatic answering machines to record messages on an analogue tape and then for the owner of the machine to play them back as he, or she, desires. Digital voice in a distributed system can be packaged up into blocks of data and stored in a conventional filing system. Voice messaging can therefore be an integral part of a PABX that is integrated into a distributed system. The ISLAND approach has been to use a server called a *Translator*[2] to act as an intermediary between a phone and a *File Server*. The *Translator* converts between voice streams delivering blocks at a high rate, and large blocks delivered to the filing service at a lower rate. This is necessary to avoid degrading the filing service by dominating its processor with the overhead of handling many small blocks. Furthermore, this approach is in line with the ISLAND philosophy which requires that a *Ringphone* should only use one voice protocol in order to implement PABX services.

A message service can be constructed from a program running in the same machine as the *Translator*, or another network server, which acquires the control of a *Ringphone* by interaction with the *Exchange Service*. Control of both the *Ringphone* and the *Translator* is necessary in order to synchronise voice recording and playback with events at the *Ringphone*. A service of this kind would also offer features typically provided by electronic mail systems but involving voice files rather than text files.

3.4.5 The ISLAND Gateway

A voice gateway has been constructed to allow the ISLAND PABX to be linked to a conventional PABX's extension line. The work was carried out for the following reasons:

- Due to a limited pool of project resources it was not possible to produce more than ten ISLAND *Ringphones*, and thus extending the new ISLAND services to the departmental PABX was a simple way of providing ISLAND facilities for a wide cross-section of clients.
- Easier access would mean greater usage and generate useful feedback about the system and, in particular, the human interfaces.

The design allows an external client the possibility of dialling from any telephone extension provided with DTMF[3] key signalling, and connecting with the ISLAND gateway which would automatically answer the call and provide a secondary dialling tone. Further *Ringphone* extensions could then be dialled by continuing to

[2]The design and construction of the *Translator* [Calnan 87] has been the work of Roger Calnan.

[3]Dual Tone Modulated Frequency (DTMF) is a coding standard which has been universally used for signalling information in touch-tone telephone networks.

press keys on the active line. These key presses are ignored by the conventional exchange but would be interpreted by the gateway as a one-to-one mapping on to *Ringphone* keys. All of the ISLAND services normally available at a *Ringphone* would be remotely accessible in this way.

3.5 The ISLAND Voice Protocol

The voice protocol [Ades 86] was originally designed for the CFR, but due to delays in the fabrication of CFR components, the ISLAND project implemented a similar protocol on the $10Mbs^{-1}$ Cambridge Ring. ISLAND's voice protocol addressed two network issues:

- Network delay
- Network error rate

The principal cause of delay in the ISLAND network is the result of buffering voice samples during packetisation. This is necessary to make optimal use of the network bandwidth when sending voice using a Data Link layer protocol. On the $10Mbs^{-1}$ Cambridge Ring data is universally transported using the *Basic Block* Protocol (BBP). This protocol is implemented as a layer on top of the Physical layer which transfers data in 16-bit units. The data part of the block must be significantly longer than the header for the transfer to be performed with a small overhead. However, when using a CFR, the Unison Data Link (UDL) protocol [Tennenhouse 86] can be used to take advantage of the 256-bit data fields present in a CFR packet. A small block of data may be transported in a single network packet when using this protocol. ISLAND's Voice Protocol is designed around a 2ms unit (16 x 8-bit samples) of voice. A 2ms unit of voice will comfortably fit into a CFR packet transferred using the UDL protocol. However, for the time being, the demonstration system transfers voice as *Basic Blocks.*

The protocol operates by periodically transmitting 2ms blocks along with a sequence number and a flag indicating whether the packet's average sample level is below a fixed silence threshold. Voice blocks are sent without acknowledgement and hence without retransmission for unacknowledged blocks. The sequence numbers allow the received voice to be placed in the correct position of a playback buffer. If a block is lost, the subsequent packets will still be placed in the correct position in the receive buffer, and thus played back at the correct time. The value of the playback pointer referencing the playback buffer is always incremented at the sample rate. The reception pointer is incremented by a value equal to the product of the blocking size and the pairwise difference of the last two sequence numbers to be received. These two pointers are initially synchronised with each other and, on reception of the first voice block after a voice stream is established, the pointers move as described. The reception buffer is made logically circular to restrict buffering space to a sensibly sized region. If the transmitter station clock and the receiver station clocks are not matched, the playback and block

reception pointers will begin to drift, relative to each other, in the period of time that the link is operational. To cope with this problem, a regular check on their separation is made, and if it exceeds a critical value the protocol brings them back into alignment once a consistent error has been determined. A consistent error is the important trigger which must not be confused with a delayed packet due to congestion, or any other reason which causes transient delay. Chapter 4 describes the hardware designed to compensate for the clock synchronisation problems that can be used when holding a two-way conversation.

The transmit section of the protocol must also use a circular buffer to cope with the case when there are network delays while launching a voice block on to the network. A transient loss of the ring integrity may cause this problem. In this case, by buffering voice blocks it may still be possible to deliver voice to the destination within the delay restrictions.

Voice blocks are flagged to be either above or below a silence threshold. The Xerox Etherphone avoids transmitting silence blocks to preserve bandwidth for other clients on the network. For ISLAND it was considered that the silence blocks should not be removed if the local bandwidth is plentiful. This is the case for the CFR network, since it has already been shown that it may support over 600 two-way conversations without silence suppression. Moreover, the reconstruction of voice from a silence-suppressed source produces results which are variable in quality. Pathological cases can occur in which low-level background noise is heard in the speech periods, but miraculously disappears in the silent periods.

However, when digitised voice is required to be stored on a physical medium the capacity problem becomes a limiting factor. Although the future promises optical discs and larger winchesters with 1 gigabit capacities, current disc technology must be treated with an economic respect for space. The *Translator* mentioned in an earlier section is responsible for removing silent blocks before storage, and reconstructing these periods on playback. Inserting white noise during the silent periods, at an attenuation equal to an average of samples taken from the background noise, is an additional technique which can be used to produce a more natural concatenation of stored voice-blocks.

3.6 Development Servers

For the initial ISLAND development a great many servers were needed. The 8MHz 68000s built into the *processor bank* of the CMDS could have been used for ISLAND purposes. At that time many of the processors in the *processor bank* were already in use during the working day. Combining this factor with the need to attach specialised hardware to some of the ring stations, for example voice digitising equipment, prompted the ISLAND group to develop its own modular 68000 server.

3.6.1 Server Hardware

In Cambridge, general purpose network processors had, in the past, been constructed from purpose-built hardware which, in itself, merits little research value. To allow, in principle, the speedy development of basic infrastructure, 'off the shelf' boards were used as a kit of parts to build a system in a 19in rack. SEEL Cambridge Ring $10Mbs^{-1}$ station/repeater cards were mapped into the local address space of a 68000 processor card manufactured by Microsys. The processor card was chosen because of its use of the VME bus standard [VMEbus 85]. The VME bus may accommodate VME peripherals which can either be purchased commercially or designed in the laboratory. The basic ISLAND server can become a general processor simply by adding VME memory or, by adding suitable VME peripherals it can become a printer, a file server, or a specialised server such as a *Ringphone*. The uniformity of server design was of great benefit allowing interchangeable parts to be moved between servers while constructing the system. Some of the early ISLAND servers had faults that were found more easily by swapping parts. The choice was undoubtably a good one for the future, and the project should continue to benefit from this approach.

3.7 Development Systems

There were two main choices for porting an operating system on to an ISLAND VME processor: Tripos [Richards 79] or Mayflower [Bacon 87]. Both had desirable attributes and both were modified to run on the new processors. An operating system and its language support have a great deal of influence on the design of a service, and for that reason a brief summary of the issues which influenced the use of these two systems are given here.

3.7.1 Tripos

Tripos is a portable, multitasking, single-user operating system, and is based on message passing principles for inter-task communication. It is implemented in the BCPL language and provides optimal support for system applications written in the same language.

Compiled BCPL forms an efficient translation of a problem into executable code. However, it is a typeless language, that is to say, the *machine word* is the only type. All program structuring and data validation is left to the programmer. This can be exactly the requirement needed when designing small real-time services such as the *Translator*, but it is generally unsatisfactory for a large-scale system development.

3.7.2 Mayflower

Mayflower is a purpose-built, single-user operating system, providing concurrency and synchronisation orientated around the *Monitor* construct [Hoare 74]. The kernel is written in the CLU language [Liskov 81] and purely intended to support applications written in *Concurrent CLU*. The Cambridge version of CLU is an extended version of the original providing concurrency control in the form of the *fork* and *semaphore* operations. Other language level synchronisation constructs such as *Monitors* and *Critical Regions* [Cooper 85b] have also been added. For the development of distributed programs, Remote Procedure Call (RPC) is by far the most useful addition [Hamilton 84]. Two classes of RPC are offered to programmers, *Exactly Once* and *At Most Once*, thus allowing a choice of protocol overhead suitable for the problem in hand. CLU is an object-based language: abstraction of data and information hiding are features that allow the definition of new abstract types and the enforcement of interfaces. For the development of large systems, modules are compiled separately. Their procedural interfaces are allocated Unique Identifiers (UIDs) which are stored in a library file and only change when the interface is modified. The UIDs of modules must be self-consistent at link time and thus type-checking extends to link time. Furthermore, the UIDs of inter-server RPCs must match at runtime, thus guaranteeing that the interface specifications are enforced at all points in the development of a distributed system.

A disadvantage of Mayflower is that program development is not carried out on the target machine. Instead a 68000 compatible compiler and linker run under UNIX on a VAX, and object modules are transferred across to the Tripos filing system where the Mayflower kernel can retrieve and then run the code. Another disadvantage is that CLU garbage collects to recover heap storage. In Mayflower a synchronous garbage collector is used, and thus an application is stopped while storage is being recovered. The delay incurred by the application is dependent on many things. Good design can reduce this problem to have minimal effect on the real-time service provided by an application.

The advantages of language level concurrency control and distributed communication far outweigh the disadvantages. CLU running under Mayflower was therefore chosen for the design and implementation of the *Exchange Service*.

3.8 Summary

The $100Mbs^{-1}$ Cambridge Fast Ring (CFR) has bandwidth sharing properties which allow it to support a realistic population of voice and data users. If only voice users access the network, more than 600 two-way conversations would be possible. In principle, the design of the CFR network could be extended to a 'Braided' or 'Self-healing' structure, and thus provide highly-available connection-paths between stations.

The ISLAND design for a distributed PABX architecture uses the same modular philosophy that has already shown itself to be successful in the computing envi-

ronment of the CMDS. Separate servers in the form of telephones, gateways, exchanges, conference servers, and translators are the key components of the system. These may all utilise the computing facilities already available on the network.

The voice protocol has been designed to use 2ms voice blocks – a unit which can be thrown away on occasion without being particularly noticeable to the human ear. The protocol utilises streams of these blocks which are sent without acknowledgement and hence without retransmission in the event of an error. However, sequence numbers are used to allow the correct management of pointers referencing the reception buffer. Silence flags are also sent with blocks, but these are only used if the voice is to be stored on a filing system with limited capacity.

ISLAND servers are general purpose and based on 68000's. These servers may be used to construct any of the required PABX services. Two operating systems, Tripos and Mayflower, have been ported to run on these servers. The ability of Mayflower to run implementations written in an extended version of the CLU language has been particularly useful in the research and development of a distributed *Exchange Service.*

Chapter 4

Network Telephone Design

To distinguish between conventional telephones and those used in the ISLAND project, the network telephone has been given the name *Ringphone*. In the design of the ISLAND demonstration system, it was of prime importance to ensure that the *Ringphone* acquired the correct division of functionality in relationship to other system components. Emphasis was placed on three issues:

- **Simplicity:** to gain reliability
- The Human Interface
- The Network Interface

4.1 Simplicity

This criterion is one which characterises the ISLAND approach to designing a network telephone server. Simplicity is a step towards achieving *reliability*.

At Xerox PARC, a site where similar research is being carried out, their Etherphone has also employed a simple approach, although some of the design issues have introduced a level of complexity which ISLAND has tried to remove. Etherphone uses two processors: one for driving the Ethernet, and the other for controlling voice digitisation. It also employs three protocols: an RPC control interface, a voice protocol between phones, another to a dedicated voice file server. To achieve a conference call, additional mechanisms working in combination with the phone-phone voice protocol are required to sum all the conferenced voice-streams at each Etherphone.

The *Ringphone* has a more simple design concept. An interface to the Cambridge Ring is provided by the same processor which drives the voice hardware. Functionally, only two protocols are used:

- The Single Shot Protocol (SSP) [Gibbons 80] for control
- The Voice Provider [Ades 86] for all voice connections

4.2 Human Interface

A well thought out human interface was considered to be essential. Of the many commercial PABX's in service today (ISDX, BTex, SL-1) only a small fraction of the facilities are used by clients on a daily basis, mainly because the interface is far too cryptic. Many special numeric codes have to be remembered and typed at various stages of a feature call. Most people cannot remember the codes and often need to access the facility in too much of a hurry to look them up. If mistakes are made, there is no feedback to indicate where the error has occurred.

ISLAND considered that the problem could be tackled by adding a 40-character display. In retrospect a larger display would have been even more appropriate. The display can provide elementary help information and lead a client through a series of options towards a target operation. Moreover, it can be used to provide warnings about mis-keyed options and additional information about incoming calls (see figure 4.1).

Four indicator LEDs and eight extra switches, in addition to the 12 keys found on most push-button phones, were also also added. These were intended for indicating the state of a call, for example 'New calls waiting' and 'Calls held'. The extra switches were supplied as a means of invoking help information and local editing-functions.

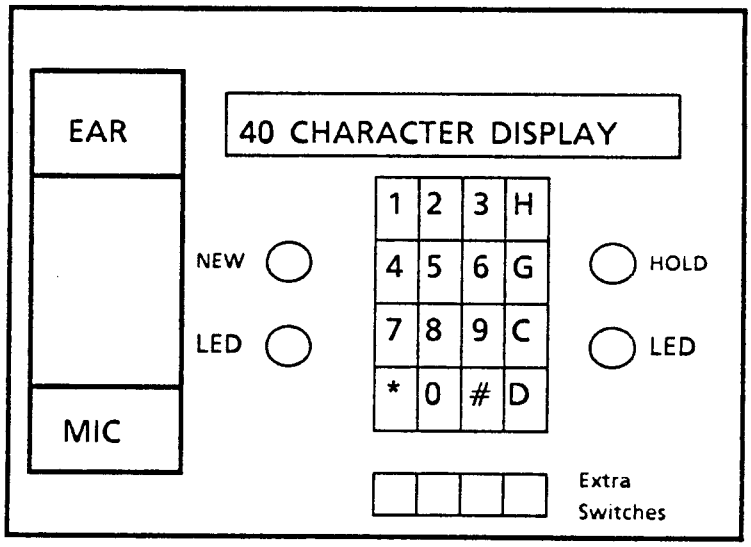

Figure 4.1: A *Ringphone's* physical appearance

4.3 Standard Network Interfaces

Within the CMDS the SSP protocol is commonly used for the purposes of inter-server control. SSPs are request/response message pairs. In the event of a protocol

failure, re-tries must be carried out as a higher-level function. They provide the same functionality as an **At Most Once** RPC system, without language level support. SSPs can be generated easily by clients working on the CMDS and were a natural choice for *Ringphone* control. The BBP protocol uses addresses and *ports* (subaddresses) to deliver data in the form of a *Basic Block* to a station. The arrival of an SSP at a *port* requires an *SSP handler* to process the request. Several different SSP interfaces can therefore be present at different *ports* within the same station. The control interface has been partitioned into two logical *channels* to distinguish state investigation from a control function:

- The Investigation Channel
- The Control Channel

4.3.1 Investigation Channel

This is an interface which allows any network server to execute a group of investigation functions returning local information about a *Ringphone's* state. An example is a function to determine where voice is being transmitted or from where it is being received. The channel is not authenticated in any way, and for this reason does not permit any modification of a *Ringphone's* internal state.

4.3.2 Control Channel

This is an authenticated SSP interface for manipulating a *Ringphone's* state. Examples include setting up voice channels, generating tones, and displaying text. The first argument of all requests made on this channel is a 64-bit UID[1] (an authenticated *session* key). The UID is compared with a privately stored UID, which must match before the function is executed, otherwise a non-zero return-code is sent in the reply. A particular function call allows the UID to be changed, but in such a way that the old UID and the new UID are both accepted until a second access of the channel has been made using the new UID. If the controlling body obtains a valid response, it will know the UID has been adopted. If the controller has to pass this new capability around to other agents it can do so before committing the *Ringphone* to take on a new private UID. If a new UID is not confirmed, the next invocation of the change function will establish the original UID and the last installed version to be valid. Thus if the controlling machine fails while establishing the initial update, a second machine could take over control, even though the private UID has not yet been distributed.

[1] A UID is a Unique Identifier and implemented by using a random binary number selected from a sparsely used number space. At Cambridge 64-bit numbers are used for this purpose.

4.3.3 Authentication

The purpose of authentication on the control channel is to ensure that the use of the telephone resource is restricted to authorised clients. There is of course a 'start of day' problem when a *Ringphone* is reset, powered up, or re-booted. In this case, the private UID must default to a known value, and for a period of time the resource is vulnerable. As soon as the authorised controller discovers that the reset has occurred it can change the UID to a secret one. If an impostor takes over control in this vulnerable period, the genuine controller will obtain a return code indicating that its own UID is invalid. If the genuine controller finds the same problem with the default UID, a message can then be sent to a human operator who can act on the situation.

4.4 Event Channel

Asynchronous events, such as key-presses or the raising of a handset, need to be signalled to the controlling entity. This information could be obtained by providing an investigation channel function permitting key-presses to be polled for. However this would be an unnecessary waste of CPU cycles. A more efficient approach is to supply the *Ringphone* with an SSP interface to signal events to the controller. This technique has been implemented in the form of an outgoing SSP channel. Events that need to be signalled are:

- Handset off-hook
- Key event
- '*' key event (see chapter 5)
- Voice channel established
- Handset on-hook

Note that for simplicity a key event simply indicates that the key buffer needs reading. It is the responsibility of the controlling server to respond in the correct way to an event. In the case of a key event the key buffer should be read through the *Control Channel.*

The destination of signalled events is set by the use of an *event vector* and this may be assigned to by calling a set_event_vector procedure through the *Control Channel.* This is an important part of a mechanism for passing control of a *Ringphone* between control servers.

The overall control mechanism is analogous to that used by a processor and a peripheral device. Events produced by a *Ringphone* are similar to a peripheral device generating interrupts indicating that it should be serviced by its processor. In our model the controlling server corresponds to the processor and the *Ringphone* to the peripheral. In the LAN environment communication is more difficult; the

servers are loosely coupled and as a result event signals need to be re-tryable. However, the model of a dumb peripheral and a controlling entity provides a simple design framework.

4.5 Finding a Controller

In ISLAND it was chosen to distribute the *Exchange Service* to gain reliability. Each individual exchange component has been termed a *Director*[2]. When a *Ringphone* has an event to signal, the name of a *Director* must be found so that the event can be passed to it and then processed. The location of components of a distributed service is a problem which is not addressed by the *Naming Service* used in the Cambridge Model Distributed System. As a result ISLAND had to find its own solution. Several design options were open:

- Broadcast
- A Name Service supporting sets of names
- A table of names maintained locally

If a broadcast mechanism were available on the slow ring, it would be possible to broadcast a request for a service using its textual name, and listen for the replies returning from the machines supporting it. The *Directors* could then reply with their name and a current-loading factor. The *Ringphone* initiating the call could then select the least busy controller. Alternatively, a centralised name service could contain service names as well as machine names. A service name look-up would return a set of machine names providing the service. The scheme suffers from a single-point dependency at the naming service. A preferable solution would combine both techniques, using a centralised name service but allowing a client to broadcast in the event of its failure, or in the case of an invalid entry. Such a technique uses the name service as a *hint* at the correct address.

Broadcasts do not scale well: therefore if a broadcast mechanism were implemented it should only extend to one network. However, a site may contain many networks, and important distributed services should also be spread out so that some of their components exist on all of the inter-connected LANs. For this reason and the lack of reliable naming support in CMDS, an alternative half-way solution has been adopted.

The *Ringphone* solution has been to maintain a set of names in a table which is initialised by each *Director*. On raising the handset, a random choice is made from the table, and the resulting name is looked up to determine the address to which the event should be sent (see figure 4.2). The *Name Server* is still a single-point dependency in the demonstration system. A solution to this problem is to

[2]In previous ISLAND literature the term General Operations Director has been used and abbreviated to GOD. Throughout this thesis *Director* is the preferred name.

use a table of addresses rather than names. However, in the CMDS environment the communication path to a *Director* may be through a bridge in which case, without a name look-up, the path through the bridge would eventually time-out. If the design is moved to the CFR, 16-bit global addresses could be stored in the *Ringphone* event table instead of using names. This solution removes the dependency on the name server and thus solves the problem.

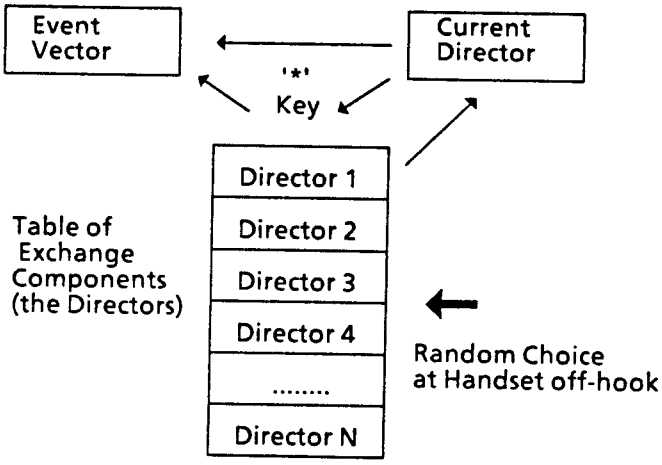

Figure 4.2: A mechanism for finding a controller

A new controller name is only re-selected by a *Ringphone* when the handset is raised, and thus, once call processing has begun with a particular *Director*, all other events are also sent to it. A **handset on-hook** event ends call processing, and further **handset off-hook** events make another random selection. The period between an off-hook and an on-hook event will from now on be referred to as a *session*. If the selected *Director* fails to respond to an off-hook event within a reasonable time-out another name is chosen. It is up to the remaining *Directors* to remove the bad name from the table in order to avoid delays in obtaining future controllers.

The above mechanism is simple, but assumes that a random selection will distribute the load evenly across all *Director* components. For large numbers of extensions in proportion to *Directors* this may be true, but if the ratio is smaller it may not work out so conveniently. A Multicast protocol[3] making use of the table of names would have a similar result to a broadcast. The selection could then have been made on the basis of least loading. Clearly, it would be advantageous to all system components to choose the least busy controller. Multicast is, however, a time consuming operation on the Cambridge Ring, and if the requests are to be overlapped in order to reduce the time penalty, the protocol handling is complex. If the number of *Ringphones* is small, the load on any one *Director* will also be small, even with an imbalanced load distribution. Multicasting was

[3]Using the Cambridge Ring network, a Multicast would have to be implemented by explicitly sending n messages to n servers and waiting for n replies.

therefore rejected.

4.6 Physical Architecture

Ringphones were to be constructed around the standard ISLAND VME server. VME Voice equipment was not available commercially, and as a result a custom VME *Ringphone* Card was designed. Figure 4.3 shows the essential components required by the *Ringphone* design.

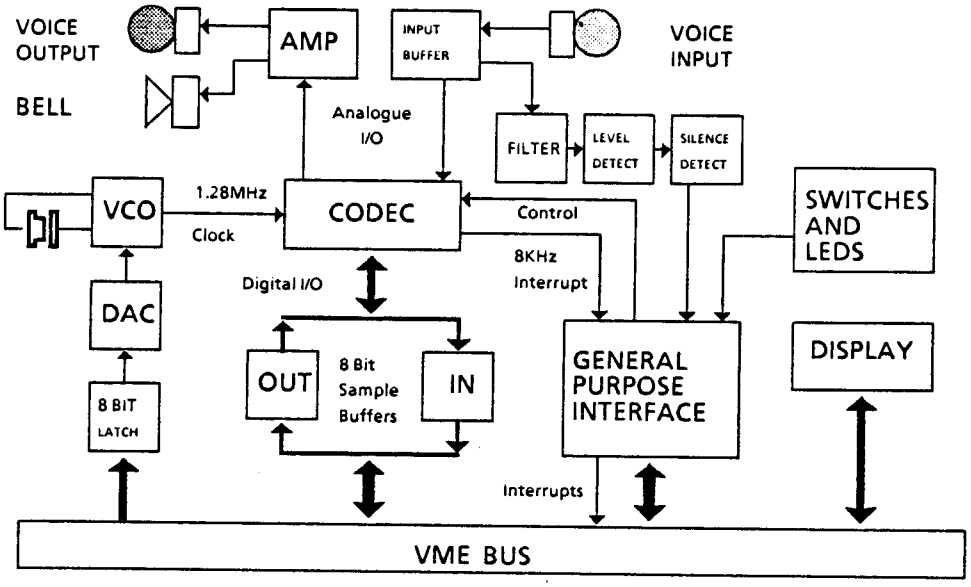

Figure 4.3: The essential components of the ISLAND *Ringphone*

The VME bus provides mechanisms for data transfer, interrupt handling and arbitration for multiple processors wishing to access the bus. The *Ringphone* card simply decodes a region of VME memory enabling four functional blocks to be accessed:

- The voice I/O buffers
- A software-controlled oscillator
- A general purpose I/O peripheral
- Display hardware

4.6.1 The Voice Interface

A voice CODEC is responsible for coding and decoding standard CCITT 64kbs^{-1} voice-streams (8-bit samples every 125μs). This device can operate at a variety of different clock speeds. A 1.28MHz clock has been chosen, it allows a convenient

frequency division of 160 to provide an 8kHz interrupt signal for servicing the voice buffers. The CODEC is buffered from the VME bus by two 8-bit latches, one of which is readable by the processor, and the other which is writable. Control hardware ensures that the buffers contain the last sample taken and the next to be played out. The buffers are needed so that the service time of the CODEC interrupt is non-critical, and may, in fact, occur any time within a 101.6μs window of the 125μs inter-sample time period.

The 1.28MHz clock is generated by a crystal oscillator derived from a hybrid voltage controlled oscillator (VCO). A standard TTL component forms the basis of the VCO but with a crystal placed across its capacitive timing pins it oscillates preferentially at the crystal frequency. However, if a control voltage is applied, the crystal frequency can be trimmed on either side of its resonance value by up to 0.1%. As explained in chapter 3, the voice protocol can run into difficulties when two *Ringphones* send samples to each other at different rates. By connecting the control pin of the VCO to a Digital to Analogue Converter (DAC), and supplying the DAC with a value derived from an 8-bit latch mapped into the VME memory space, it is possible to change the sample frequency in a small range divided into 256 steps. The exact adjustment depends on many parameters, which include the DAC reference-voltage, and individual characteristics of the crystal and the oscillator. This variability made it impossible to work out an absolute adjustment for the VCO when exchanging samples between two *Ringphones*, but because the crystal oscillators have an operating frequency stability of better that 0.01%, a simple solution is possible. Software is now implemented to measure when there is a significant difference between the rate of samples received and those played back. In this event the latch is adjusted by only 1 part in 256, either up or down in value, in order to make the sample clocks slowly converge on a common frequency. The process is that of a heavily damped negative-feedback system. The voice buffer pointers are then put in their corrected positions and the process continues. If many calls are made, all the voice-sampling clocks in the system will begin to converge on a common value, thus reducing the minor loss of fidelity due to this problem.

4.6.2 Silence Detection

Silence detection is a feature required by the ISLAND voice protocol. Each block of voice is flagged either silent or noisy. Such information could be derived by software carrying out an averaging function over the voice samples in a block and comparing it with a threshold level. Because of the rate at which voice is sampled, such processing would lead to a considerable increase in overhead per block. A simple solution was adopted which uses hardware components. The analogue signal fed into the CODEC is also routed to a filter removing frequencies that do not normally occur in speech. The result is passed to an analogue comparator and compared against a threshold reference set manually with a potential divider. The comparator output is high or low for each half cycle of the signal that is greater than the threshold. Its output is not a useful representation of silence. The signal

is processed further by feeding it to a re-triggerable monostable with a manually adjustable period. The period is set to be long enough to extend between two peaks of the lowest discernible tone (10Hz), but not so long that it spans between normal inter-word pauses. A period of 0.2 seconds was found to be satisfactory for this value. The silence threshold is far more experimental and has to be manually adjusted so that a loud whisper can be discerned. However, in a noisy room a higher threshold level is more appropriate. The boolean output of the monostable is fed into the general purpose I/O peripheral and is thus readable by processor during the period when a block is being prepared for transmission. Silence can now be determined with an overhead of only one memory-access.

Further work[4] on silence-suppressed stored-voice, found that the efficient reconstruction of silent blocks with white noise could only be successful if an average value of low-level noise in the silent blocks were known. Again this could be carried out by software, but with great overhead. A better hardware solution would have been to keep a running total of the samples taken by the CODEC in a latch with its value readable in VME memory. If a block of voice is ready for transmission the total could be read with little overhead, and then cleared before the start of another block. Instead of a boolean flag this value would represent an average amplitude for the block, and could be used to adjust the attenuation of a white noise 'filling' routine when playing back silence compressed in this way.

4.6.3 Interrupt Layering

Due to the real-time properties of voice, the relative importance of concurrent *Ringphone* tasks had to be chosen carefully and then allocated suitable interrupt priorities. The most time critical operation is that of transferring samples to and from the CODEC. Because only one sample is buffered per interrupt, CODEC interrupts must be given the highest priority.

The reception of a block from the network is notified by an interrupt from the station at a lower interrupt priority and the corresponding received-block up-call is serviced at this level.

All network transmissions are carried out at a priority lower than the previous two. This makes them independent of any foreground process which may idle too long causing transmissions to fall behind schedule. It is important to place receive priority above that used for transmission, in order to ensure that packets placed on the network are mopped up before more are allowed to be transmitted. It should be noted that receive up-calls handle both voice and SSP (data) receptions and, as far as the interface is aware, there is no difference between the information it is handling.

Finally, the lowest priority tasks are carried out in a foreground poll-loop. The poll-loop checks for any events that need signalling, it reads the state of the handset and, amongst other mundane tasks, it is responsible for setting up the voice channels.

[4] All research on stored voice in the ISLAND project has been carried out by Roger Calnan.

4.7 Timing Considerations

The previous section describes a priority ordering for interrupt handling. It is important to remember that the processor's time is spread across all of its tasks, and that heavy processing at the highest level reduces the number of cycles available at a lower priority level, and likewise for other levels below it. If the high priority routines are not made efficient, it can lead to a situation in which there are not enough processor cycles available to handle data in lower priority tasks. The network load cannot be reduced with flow control when real-time data is present because it must be handled within a narrow time constraint. When designing and implementing the voice provider, it was difficult to estimate how much processor time would be used up in executing the protocol. As a result some timing problems occurred. The voice channel consisted of three protocol layers built on top of the $10Mbs^{-1}$ Cambridge Ring (the Physical layer).

- The CODEC handling routine
- The Voice Provider: (Voice Protocol)
- The *Basic Block* protocol (Data-Link layer)

The combined overhead of all three layers left the processor with very few cycles to spare. It is unfortunate that modularisation, which is so necessary in designing predictable programs, usually results in a degree of inefficiency. The lesson learnt from the first *Ringphone* implementation was that 16 samples per *Basic Block*[5] overwhelmed the capability of the processor to transmit and receive voice simultaneously. The CODEC service time, combined with the overhead of receiving and transmitting a *Basic Block*, was too great. A demonstration of the problem could be achieved by configuring a *Ringphone* to transmit to itself and listening to the disjoint results. An interesting observation was that the adjustable software sample-clock began to change the sample rate on the basis that it was too fast for its partner *Ringphone*. By allowing the oscillator an artificially greater frequency range, the sample rate was thus slowly reduced until the processor could keep up with the data rates. The final settling frequency of the sample clock was a good indication of how much more efficient the implementation had to be made. The main inefficiency resulted from the overhead of handling a *Basic Block* in relation to the number of *minipackets* it contained.

To solve this problem the number of samples transmitted in each block was increased. An early implementation using an incomplete voice protocol was found to be workable with 32-samples per block. A full implementation required 64-samples per block to guarantee a margin large enough to allow monitoring routines to be added without pushing the processor too close to its limitations. The ISLAND demonstration system therefore uses 8ms of voice in a single *Basic Block*. For the purposes of normal use this has produced an acceptable overall delay of 24ms. This

[5] 16 samples are equivalent to 16 x 125μs = 2ms of voice

is due to one block being buffered at the transmitter while it is being assembled, and two others being buffered at the receiver to cope with jitter in the transport delay. Transport jitter delays may result from several transmitters competing for one receiver.

Despite these problems the ISLAND *Ringphones* performed well and have been sufficient to demonstrate the problems in building a packet-voice telephone on a Cambridge Ring (CR). However with this experience, if the *Ringphone* were re-implemented, the buffering of voice samples would have been carried out in hardware, thus reducing the overhead of servicing the CODEC at 8KHz. Further advantages would be gained if a Cambridge Fast Ring were used. This is because a 2ms voice block can be transported by a single CFR slot and thus a host would only incur a reception overhead per block rather than for every 16 bits of voice delivered in a CR *minipacket*.

4.8 A Future Fast Ring Telephone

The Cambridge Fast Ring station is implemented from two components:

- An ECL chip providing a serial interface with the ring
- A CMOS chip providing a parallel interface and the characteristic station function

The CMOS chip utilises two FIFOs 32-bytes deep for buffering reception and transmission packets. These buffers can be accessed through the host interface of the chip, along with various other registers for its control. For the purposes of easy development, the station chip can be mapped into the address space of a small microprocessor and can be configured to interrupt the processor on packet arrival.

Developments on the Cambridge Fast Ring have only been possible recently. One example is a simple prototype server developed for the CFR and called the *Tiny Server*. It utilises a Motorola 8-bit 6803 processor to perform the function of a host. The *Tiny Server* can easily be extended to accommodate more peripherals on to its bus and thus the voice I/O design described earlier could be mapped into its address space. A CFR telephone could be constructed around such a simple server if it was given suitable hardware support. Two 16-byte FIFOs could be used in place of the single packet buffers and an interrupt generated only when the sampling buffer is full. The responding service routine could then sit in a tight loop filling and emptying the two voice FIFOs. A separate interrupt service routine for the station chip could also be in operation to handle incoming voice blocks and execute the transfer between the receive FIFO of the station and the playback FIFO attached to the CODEC. Such a scheme would greatly reduce a voice block's assembly time.

Figure 4.4 shows a possible hardware configuration for this design, and includes an optional dedicated hardware component to transfer data between the voice FIFOs

and the station FIFOs. If this hardware support were used an interrupt would still be issued when the voice FIFO was full, but the CPU would simply write to the hardware controller to initiate the transfer which could take place at full component speed (an order of magnitude faster than possible with the processor). By freeing up so much of the 6803's time, more voice I/O circuits could be placed in the *Tiny Server's* address map, sampling in parallel, but accessed for the 16-sample transfer in quick succession. The advantage of attaching, say four, pieces of voice equipment to a single CFR interface is primarily cost: network attachments, especially to fibre links, will always be expensive. The CFR has the advantage that several stations (CMOS chips) may be connected to a single ECL ring-interface [Temple 84]. However, it will still be more cost effective to have a telephone concentrator constructed around one processor and one station chip, rather than four of each. Unfortunately, attaching more devices to a single station causes a reduction in availability; when using a telephone concentrator, if a station fails, four extensions fail rather than one.

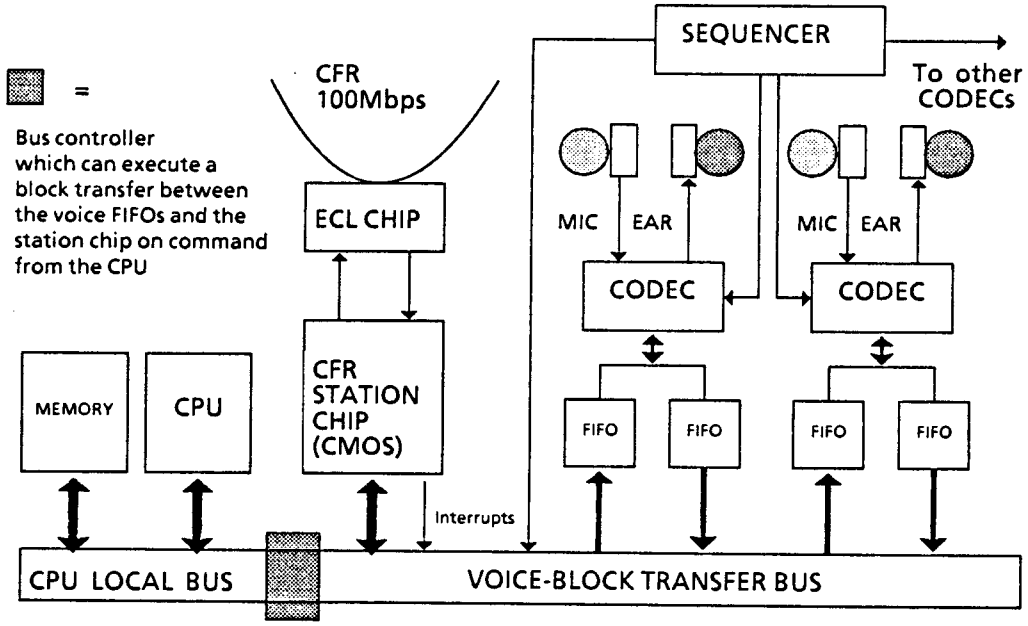

Figure 4.4: A Cambridge Fast Ring (CFR) telephone

The CMDS model uses the concept of a *Terminal Concentrator* to connect many clients to the network rather than directly to a host. A station connecting several pieces of voice equipment uses a similar philosophy and could be considered as a *Telephone Concentrator*. A *Telephone Concentrator* is also basically dumb in its operation. It provides a source and a sink for voice, controlled by either the *Exchange* managing virtual connections between *Ringphones*, or from other applications such as the storage and playback of voice from a disk.

ISLAND approached a telephone design by taking the opposite extreme of the closely coupled PABX architecture used in conventional systems (for example ISDX, BTex, and Monarch), and has reached a compromise. The *Telephone Concentrator* is analogous to a line card supporting 4 to 8 phones, and the CFR serves

the same purpose as a *Time Switch*. Figure 4.5 shows the various arrangements possible for different degrees of decentralisation, including a further intermediate design using the line shelf principle to multiplex a number of line cards onto a 2.048Mbs^{-1} bus, but is then attached to a special purpose station interfacing it with the ring. This hybrid configuration could be used in conjunction with *Telephone Concentrators* and individual *Ringphones*, if it were necessary, to suit several different needs at a particular installation.

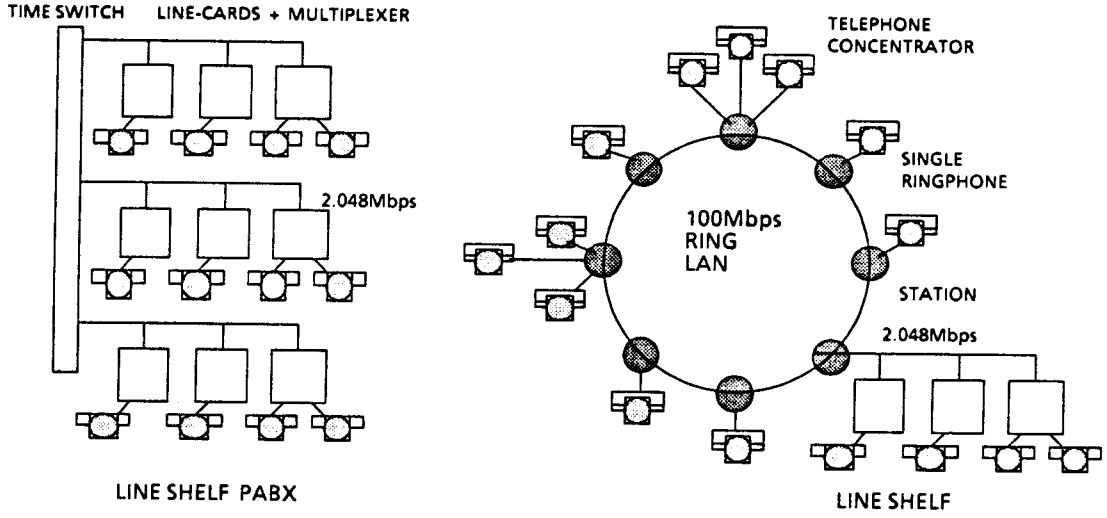

Figure 4.5: Alternative approaches for decentralising a PABX

4.9 A Gateway to ISLAND

Chapter 3 discussed the need for an ISLAND gateway to the existing laboratory PABX. An interface was designed and built to connect a *Ringphone* and a single PABX extension line (see figure 4.6).

Isolation was the first consideration. The second criterion was simplicity, with the intention of using as much of the existing *Ringphone* and conventional extension hardware as possible. Electrical isolation between the audio circuits used standard 1:1 isolation transformers. An unused parallel port on the *Ringphone* processor provided control and sense lines for the purposes of controlling the extension hardware. The mechanical operations were put into effect by buffering the signals with isolation relays and using their contacts to bridge the switches.

Incoming calls were the most difficult events to deal with. A bell detector was implemented by driving an optoisolator with the bell current and converting the isolated signal into a logic level which, in its active state, causes a processor interrupt. The service routine responds by logically lifting the handset and thus answering the call while also generating a *Ringphone* handset off-hook event. The ISLAND *Exchange* responds with a dialling tone which is now heard by the caller. It was realised that the DTMF tones, which would be superimposed on

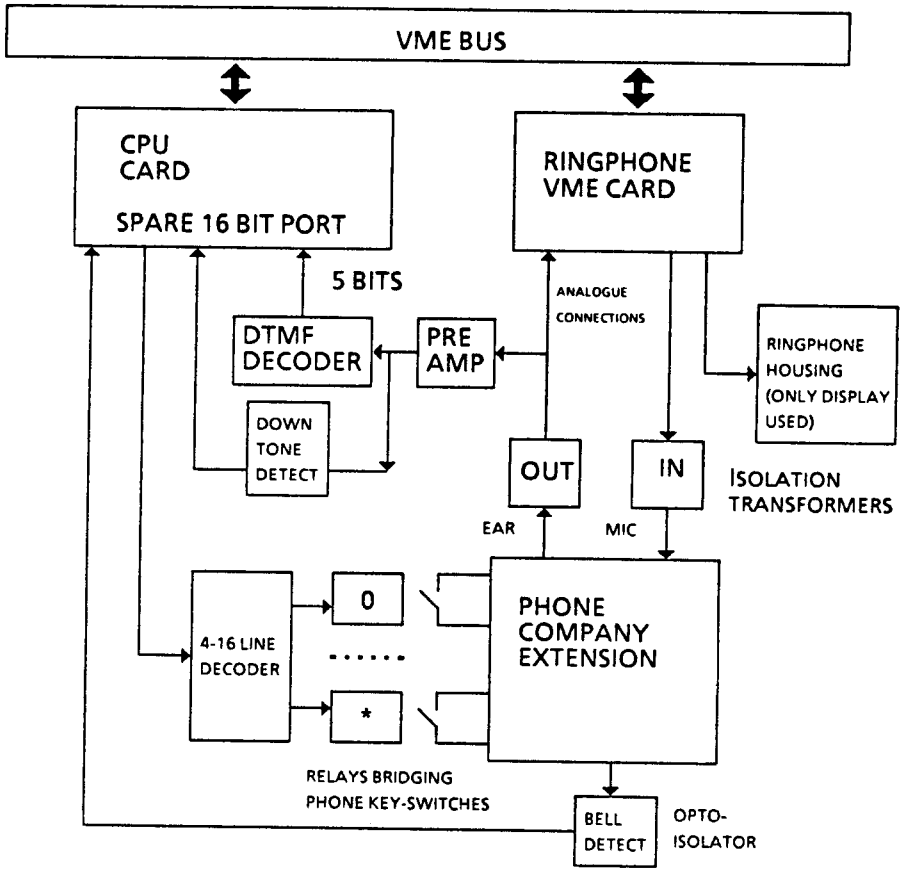

Figure 4.6: A block diagram of the ISLAND gateway

the voice signal when the connection was complete, could be used to initiate *Ringphone* key presses. A DTMF decoder was built from standard telecommunication devices and attached to the isolated audio signal. The resulting digitally-decoded output was further arranged to cause interrupts, and the decoded key placed in the *Ringphone* key buffer, thus generating a key event signal as if it had been physically pressed on the key pad. By mapping the extension keys to the most functional *Ringphone* keys it is possible to access all of its features remotely. The standard configuration of keys shown in figure 4.1, and used by ISLAND, has been adopted by most telecommunication companies. Once a call has been answered automatically the only way it can be cleared down is by detecting that the remote party has replaced the handset. The laboratory PABX in fact responds with a tone of characteristic frequency at 1645Hz. A decoder was built to detect this tone and generate yet another source of interrupts, causing the replacement of the handset and a *Ringphone* handset on-hook event, thus closing down the call in both systems.

The gateway could be further enhanced by providing the remote party with information that would normally appear on the *Ringphone* display. Commercial devices, of which DecTalk^{TM} (a DEC product) is an example, are now available and make an excellent attempt at speaking ASCII text. The output signal of such

a device could be combined with the voice signals in the Gateway station. The Ringphone could simply route its text to the 'Talkers' input with the result that the remote caller would hear help information that is normally only available at a *Ringphone.*

4.10 Summary

The ISLAND *Ringphone* is functionally very simple. However, some complexity has been added to improve the human interface. This involves the addition of a single-line text-display, some lights, and some extra switches.

Ringphone utilises two SSP-based network-interfaces for the purpose of investigating and controlling its own state. These are the *Investigation Channel* and *Control Channel* respectively. In addition, events occurring as the result of client actions are signalled by an outgoing *Event Channel.* The destination to which events are sent is governed by the *event vector.* The *event vector* is randomly set to point at one of its controlling entities by randomly choosing a name from an internal table after a **handset off-hook** event. However, the *event vector* can be further modified by use of the *Control Channel* service interface.

The hardware employs a voice CODEC to code and decode digital voice samples. These samples are collected by a handling routine and used in the voice protocol described in chapter 3. The overhead found in executing the protocol, in addition to servicing the CODEC and the network interface, was high enough to push a Motorola 8MHz 68000 to its limits when using 16-samples per voice block. A larger block size of 64-samples per *Basic Block* is now used to provide a less time-critical implementation. If future network telephones are designed for the CFR, hardware support could be given to the current design to reduced the overhead of handling the CODEC. This improvement, combined with the fact that 2ms of voice will fit into a single CFR packet, would also allow a much smaller processor to be used for the telephone's control.

Chapter 5

A Model for the Exchange Service

The ISLAND *Exchange* was designed to fulfil the following requirements:

- An advanced PABX service
- Extensible on-line addition of features
- Mechanisms for transferring control of *Ringphones* to other services
- An ensured highly available service

Clearly, there are numerous implementations which could satisfy these requirements. The intention was to find a model which would lend itself to the requirements of a growing multi-feature system, and be suitable for extending it to a distributed implementation (chapter 6). The ISLAND demonstration system was implemented around this model. The main body of this chapter describes the design and the issues involved.

5.1 Advanced Requirements for a PABX

The ISLAND *Ringphone*, by the use of a text display, enables a variety of new features to be incorporated into the service which would normally be too complex for the standard telephone interface. The following features were considered as advanced facilities and designed as an integral part of the distributed PABX.

Multi-lining: Conventional telephones have the ability to connect to only one extension at a time. It is only the switchboard facilities that have the ability to manipulate many lines at once. Some PABX products allow a 'hold and call' facility. This allows the current call to be held while another is made, with the possibility of switching back to the original. All connections within ISLAND are virtual by the very nature of the system and thus there is no reason to limit the number of connections associated with a *Ringphone*. Multi-lining, a facility designed to take advantage of virtual connections, is the ability to set up arbitrary

numbers of logical connections at any telephone, and select between them using a random access scheme. Incoming calls to a busy telephone are indicated by an LED on the front panel[1] and a client is given the option of accepting the call. The accepted call simply becomes another virtual connection and can be selected in an identical way to the others.

Profiles: Conventional telephone systems are relatively impersonal. Numbers are dialled to contact telephone equipment associated with those numbers, and traditionally, also associated with a person or group of people. ISLAND wanted its clients to be able to personalise a telephone. Incoming calls were not to be signalled with the impersonal bell, but a tone characteristic of the person the call was destined for and, furthermore, provide textual information about a caller's name or number.

In the ISLAND system *Ringphones* and clients are distinguished as separate entities and have numbers divorced from each other. Temporary associations allow clients to move between extensions, informing the exchange of the location at which they would currently like to receive calls. A client has the choice of calling a location or a person. Any entry in the directory can be associated with a profile that defines parameters which include:

- **Bell** tone: the tone used to summon the called party
- **Short** codes: a short hand version of a number
- **Hate** and **Love** lists: calls to be rejected or those only to be accepted
- **Pick-up** groups: lists of trusted numbers that are allowed to pick-up calls
- **Banner**: default banner display-text
- **Hotdial**: makes the telephone into a hot-dial telephone e.g an alarm phone

A client or a *Ringphone* can have a profile. At the 'start of day' for the *Exchange*, all *Ringphones* would have their own default profile. Whilst clients worked with the system, *Ringphones* would take on the required personalisation. The most recently invoked profile at a particular *Ringphone* would be the one which was active. Underlying profiles would now merely mark the location to deliver a call relating to that profile.

5.1.1 The Directory

Directories traditionally contain an ordered association of names to numbers and locations. ISLAND has extended the function of the directory because it is the logical place to store references to the extra data needed for personalisation. The record entries for the directory were finalised as:

[1]There are 4 LEDs, only two of which have been used, one to indicate that new calls have arrived and the other to indicate that some calls are held.

- **Number:** the number actually dialled
- **Status:** 'S' in Service, 'W' Withdrawn (temporary repair)
- **PIN:** Personal Identification Number (secret)
- **Station Name:** the textual name
- **Computer User-id:** the owner or default owner
- **Owner name:** the name in real life
- **Class:** (Phone, Person, Gateway, Service)
- **Location:** location of the default telephone associated with this entry
- **Profile:** the *file name* of a client's profile

The directory is stored as a restricted-access system-file, but a client is given tools to browse public information in the file from a *Ringphone*. It may be read sequentially or hunted through using a binary chop mechanism to home in on an entry.

These enhancements combined with conventional PABX features, provide an improved human interface at the telephone without needing any additional help from a personal workstation. Facilities for conferencing and automatic voice messaging were not designed as an integral part of the exchange because they were considered as external services[2]. As such they should be implemented in separate network services which interact with the exchange to establish control of telephone resources.

5.2 Implementing a Director

The *Director*, a single component of the distributed exchange, was developed in the Mayflower/Concurrent CLU environment, for reasons considered in chapter 3. CLU is an object-based language allowing systems to be defined in terms of abstracted data. The choice of abstraction boundaries is the most daunting task in specifying a design. Early mistakes can make the system awkward to expand and force further bad choices of abstraction as it grows. A trial system was made to determine the problems faced by the design. The resulting implementation was adopted with a software model satisfying the aims already set out. The following sections describe key components of the model used.

[2]Conferencing and message-leaving services had not been integrated into the ISLAND demonstration system at the time of this work. However, voice storage has been achieved as a separate demonstration.

5.2.1 The Event Handler

Events at a *Ringphone* are signalled by SSPs which are sent to a controller (see figure 5.1), the name of which is randomly selected from an internal table of all known *Exchange* servers (the *Directors*). A new controller is selected at the beginning of every *session*[3]. All *Directors* will initialise the table in each *Ringphone* after their own initialisation procedure and will then listen for incoming SSP events. When an SSP arrives, the format is checked, the events are copied, and a reply is sent back. If the reply is not returned within a fixed period of time, the *Ringphone* repeats the request. The destination name of the reply is determined by performing a 'reverse look-up' of the incoming request at the *Name Server*. The source address itself is not enough to determine the name because the request may have travelled through one or more bridges on its journey[4]. Once the textual name of the *Ringphone* that produced the SSP has been determined by the chosen *Director*, the information can be passed to an internal object which is responsible for processing the event.

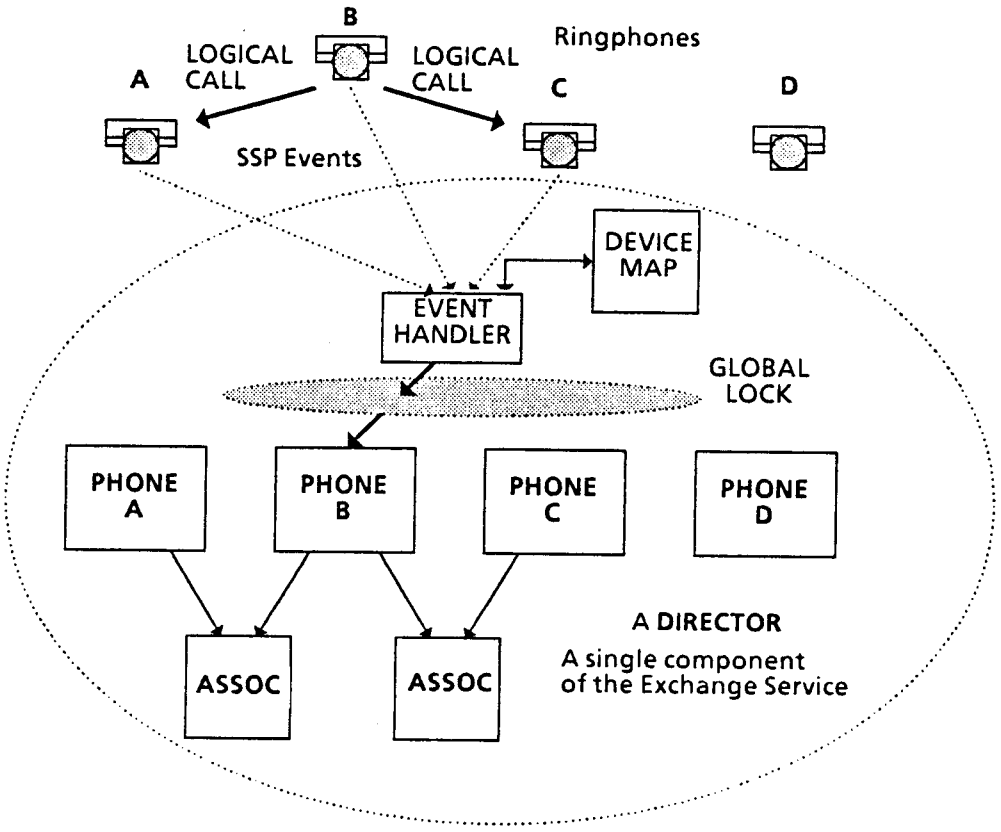

Figure 5.1: The main data structures used by a *Director*

[3]A session is defined to be initiated by a handset off-hook event and terminated by a handset on-hook event.

[4]The problem arises from the fact that inter-connected Cambridge Rings do not use a global addressing scheme.

5.2.2 The Device Map

At the initialisation of the *Exchange*, a directory (see section 5.6) is used to find out which *Ringphones* are present in the system. Each *Director* then creates an identical set of objects to provide a replicated model of each *Ringphone's* state. These objects are represented by instances of the abstract type **phone** (see figure 5.1). To allow the *Event Handler* and other components of each *Director* to obtain a handle on these objects, a *Device Map* is created associating the station names of *Ringphones* with the objects themselves. Thus a 'find name' operation can be performed on the *Device Map* by the *Event Handler* and a **phone** object returned. The events are passed on to **phone** objects through a procedural interface which queues them inside the object for processing. It should be pointed out that the *Event Handler* can set a concurrency level to handle a number of SSPs in quick succession. Thus the first handling process does not have to return before another is fired up. The concurrency level was set to 10 to ensure that this was not a bottleneck in handling the load presented by many *Ringphones*.

5.2.3 Phone Modelling

Phone objects are instantiated with a **create** operation. A process is attached by forking an activation process that waits on a semaphore for the notification of events placed in its own event queue. Thus each **phone** object has its own thread of control for processing events.

Phone is defined as a *monitor* for which a global lock (at one *Director*) is used to serialise the processing of all events that are destined for any instance of a **phone** object contained in that *Director*. The reason for this choice will be explained in section 5.8 along with a description of an alternative approach for controlling the concurrency of **phone** operations.

Once an event has been queued at a **phone** object and the semaphore on which the activation process has waited is signalled to, the *Event Handler* process is free to deal with more event SSPs. The activation process is now free to run and process the event in its own time.

5.2.4 The Call Finite State Machine (CFSM)

A telephone call can be modelled as a finite state machine, the state changes of which are effected by events from a telephone console. The ISLAND model implements a CFSM using a group of procedures which represent the inter-state transitions (see figure 5.2). These procedures are defined in the specification of the type **phone**, an instance of which also contains the state of a related *Ringphone*. The states themselves are waiting periods which are associated with a *Choice* set containing *Event/Transition* pairs. After execution of a state transition procedure, the next valid *Choice* records are constructed and placed in the *Choice* set. Events are represented by integers in a FIFO event queue. When an event arrives, the

activation procedure matches it against those in the *Choice* set. If a match is found then the corresponding transition procedure is called, the execution of which causes a state change at the *Ringphone*. These changes will then prompt a client for further events, thus executing the Call Finite State Machine.

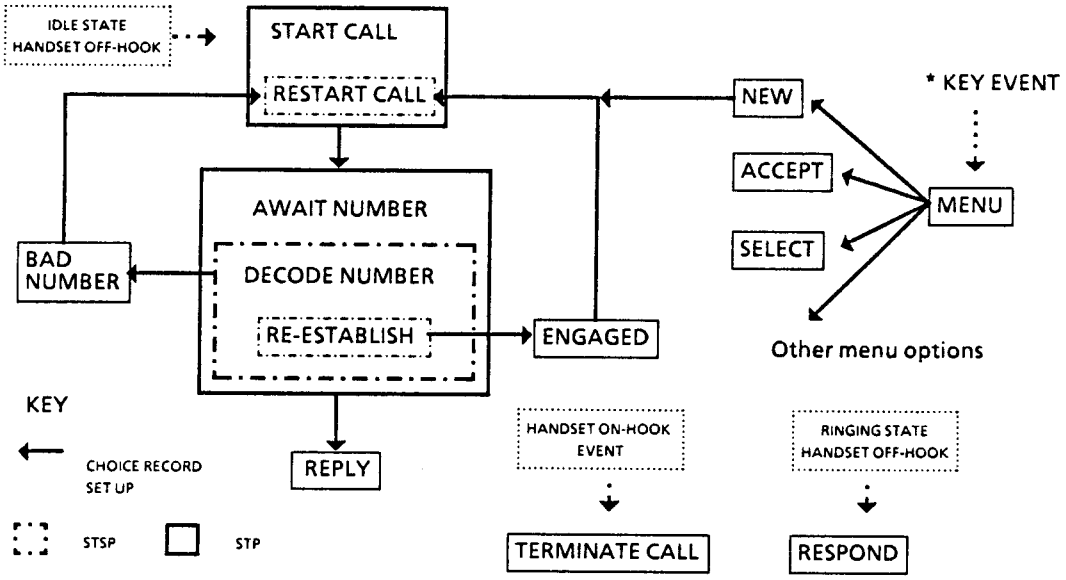

Figure 5.2: The ISLAND Call Finite State Machine (CFSM)

5.2.5 State Transition Procedures (STP)

A state transition is necessarily made up of a number of sequential actions. Some actions will affect state in a *Director*, others will affect the telephone. All operations carried out on a *Ringphone* are remote operations, and need to be packaged up in a modular unit that handles error conditions and performs retries. The remote operations are carried out by SSPs. Return codes have to be dealt with and converted to relevant CLU signals. Returned results must also be converted into a CLU type. To hide the complexity of these low level tasks a type **phi** (PHone Interface) is defined.

Ideally, the state transition procedures would be atomic operations. The various actions which compose them must all succeed for this to be true. The remote actions carried out in **phi** objects will not necessarily complete, and despite a

sensible retry period may signal an SSP failure. Each procedure has an exception handler which will catch the failure and provide either a tidying up operation or simply return. Thus the transition will always commit or fail resulting in the *Exchange* adopting a valid state. Failures will always return a **phone** object to the state before the last event, and it is left up to a client to repeat an action. Some **phi** operations are not idempotent. For instance, reading a key buffer will also clear it. Lost results will cause the exchange to read it again on a retry. All cases of lost results have been arranged to produce controlled effects. For example, a digit-less number read as the result of a previous lost message would produce an unobtainable response and thus prompt the client to try again. There are only a few cases for which this is known to occur and exception handlers have been written to correctly deal with the situation.

5.2.6 State Transition Subprocedures (STSP)

Some state transition procedures can in fact be broken down into a number of subprocedures. These were identified when it became clear that a number of state transitions shared the same sequence of actions. The shared actions must be sequential and typically only the last will establish a new *Choice* set. Figure 5.2 shows an example of an STSP called by an STP. The **start_call** procedure is an example of a sequence of actions terminating in the STSP of **restart_call**, even when the state transition is the procedure first executed after a **handset off-hook** event. Its purpose is to allow the restart operations to be used by later transitions, for example the **engaged_state** and **bad_number** transition procedures.

Some STPs were found to need multiple exit points. That is to say, there are sometimes several conditions in which a state transition procedure could return before executing in full. This may be as a result of a value read from the *Ringphone* key-buffer. For each exit the *Choice* set must be correctly reconstructed.

There is another kind of state transition that needs to be dealt with. Some events are unconditional. That is to say the action taken in response to an event is not related to the current state. Two *Ringphone* events are of this category: **handset on-hook** and **attention**. **Handset on-hook** is unconditional because of its unequivocal signal to terminate the activity of a *Ringphone's session* and establish the idle state. The **attention** signal originates from the '*' key being pressed, and indicates the activation of special features. In fact it causes the *Ringphone* to enter a state at the top of the feature menu described in 5.4.

These unconditional events may be considered as parallel to the non-maskable interrupt found in the hardware model for a processor/peripheral pair, while other events, such as key-presses, are maskable by the *Choice* set. The ISLAND model for control is similar, using an intelligent controller and a peripheral of simple design to interface with its environment.

5.3 Associations and Associators

Mechanisms for leading a single *Ringphone* through a CFSM have been described. The procedure by which it can be associated with another *Ringphone* or service requires additional consideration. Traditional telephones use real circuits for interconnection. The modern PABX and the ISLAND connections are virtual circuits transporting packetised voice samples. The connection descriptions are nothing more than associations. The ISLAND approach has been to model these associations with an abstraction of type **assoc** (an abbreviation for associator). **Assoc** objects provide a level of indirection between associated components, which may be *Ringphones* or service objects (see 5.9.1). An associator also contains state information that is attached to a connection rather than to the parties involved e.g. accounting and connection statistics. Furthermore, it is a repository for the essential state information relating to a telephone call and can easily be transferred between machines to distribute state about the connectivity of the system. In general an association object will only last the duration of a call and for that reason they have been termed 'transient objects' containing 'transient state'.

A four-function procedural interface was found to be sufficient to implement a protocol allowing **phone** objects to switch between many associations:

- **establish** signals(engaged, deleted)
- **connect** signals(engaged, wait, deleted)
- **hold** signals(failed, not_established, deleted)
- **delete** signals(failed)

These routines represent the four types of state transition which an association can make, and is referred to in this text as the ECHD interface. A description of legal state transitions is given in figure 5.3. When dialling a number, an associator is constructed by a **phone** object with a **create** operation which takes arguments containing the numbers of both parties and the station name that originated the call (since the calling number may be associated with a person at the *Ringphone* rather than with that station itself). Once created, the associator is added to a local association store and then the **establish** procedure is called, which in turn, in the case of a **phone** object, calls the **establish** procedure at the target object. Appropriate actions will be taken according to its state, for example if it is busy, an engaged signal is given. If it is free, the bell tone will be activated by the corresponding transition procedure, and information will be displayed about the caller which is derived from the association object.

In all service features, ISLAND has tried to make the rights of a caller and a called-party equal. An incoming call has traditionally taken an unjustified precedence over other activities in its locality. The display of caller information combined with the incoming call filters ('Love' and 'Hate' lists), provide a suitable means for a client to be protected against the intrusiveness of a phone call.

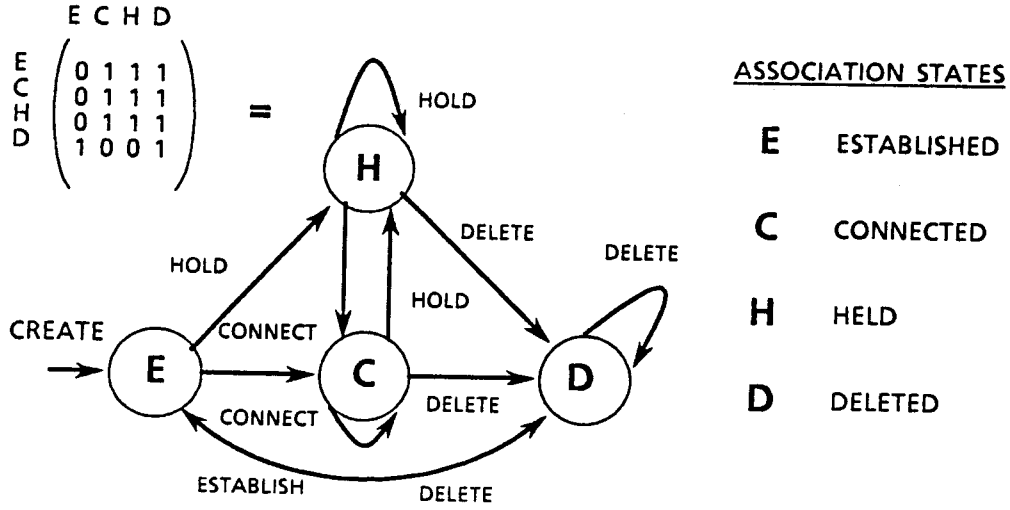

Figure 5.3: State transitions allowed by the ECHD interface

Once the called *Ringphone* has its handset lifted, it responds to **establish** by calling the **connect** procedure on the current association object, and also prepares its related *Ringphone* to receive voice from the calling party. Originally, on receiving the **establish** call it will also have assumed that the calling party's **phone** object has prepared its own *Ringphone* to receive voice and will instruct the called *Ringphone* to start transmitting voice (see figure 5.4). On receiving **connect** the calling party's object will also instruct its *Ringphone* to transmit voice at the destination and hence the association is now active and complete. A **delete** operation, caused by a **handset on-hook** event, may be invoked by either of the two parties and close down the call: transmissions first, followed by the voice reception handlers. So far **hold** has been omitted from this sequence of events, because it is necessary to have a prior knowledge of how special features are activated (**hold** is described in section 5.4.1).

5.4 The Feature Menu

The '*' key has been described earlier as a means of causing an unconditional event. The CFSM will respond by providing a client with a feature menu. The menu will provide a prompt on the display for a numeric entry which leads a client through a tree of options. The 'leaves' of the tree contain pointers to the procedures that implement the features. The result of activating these routines may further prompt for parameters which are used to carry them out. Figure 5.5 is a textual description of the tree structure. With a standard numeric key-pad a maximum of ten options are available at every menu level. The '0' key has a special function meaning 'ascend one level in the tree'. '*' will always return a client to the top of the menu —an invaluable operation when a client is lost in the intricacies of a little-used and unfamiliar feature.

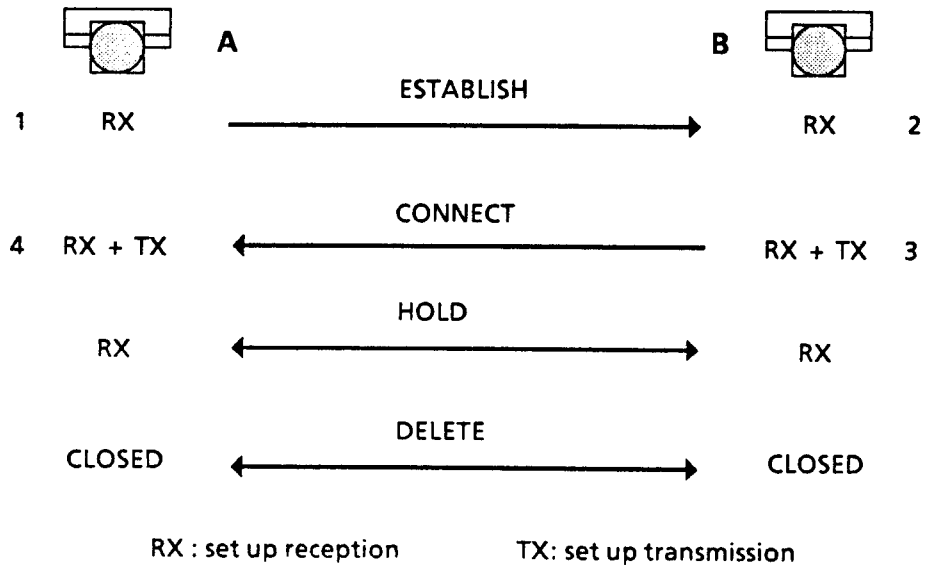

Figure 5.4: Using the ECHD interface to set up a voice connection

A 40-character display is not sufficient to show all the necessary information at any particular menu level. The client can access the complete help line by using the 'HELP' key which will cyclically guide a client through many 40-character multiples of text. It is hoped that an initially unfamiliar client will make full use of the help facility to determine the sequences of key-presses necessary to reach a feature, but the common sequences will become well known. A client can be reassured that the help mechanism can always be reverted to, even when he or she becomes forgetful.

5.4.1 Multi-lining: a Switch-Board in Every Telephone

This mechanism has been achieved using four menu features:

- **new**
- **accept**
- **select**
- **list**

If a call has been established between two parties A and B, and B wishes to establish a second line, then the **new** feature can be invoked. The result will be a call on the current associator to **hold**, forcing both parties into the held state, a situation in which transmitted voice is stopped but voice reception is not cancelled. B will then re-enter the CFSM and a number will be prompted for creating a new associator and thus establishing the new call. The **select** feature allows the current association to be held and an old one resumed. Of course, the

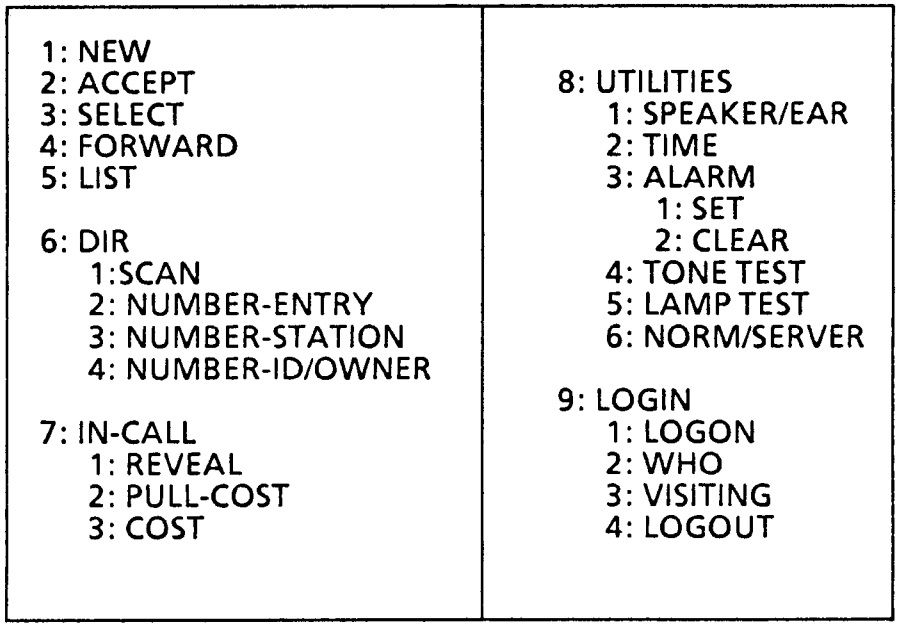

Figure 5.5: The ISLAND feature menu

remote party A may have already held the call or even deleted the association, in which case appropriate messages will be sent to B. Each **phone** object maintains a linear list of its associations and allocates integers to them. A client therefore has a handle on a call, and can use the integers as arguments to the **select** function. Deleted associators are marked to be in the delete state, but not removed from a *Ringphone's* association list until it has also put its own handset on-hook. This scheme allows the ordering of associators to be maintained. An associator can also be re-used: on selecting a deleted associator there is an option to ring the remote party again. Associators are useful tokens and should not be thrown away until a *session* has been ended.

When a *Ringphone* is engaged it is informed about incoming calls by an LED on its front panel. The **accept** function will cause the current association to be held and the incoming call selected and added as the next entry in the list of associations. Incoming calls are in fact stored in a FIFO list to cater for many calls arriving at once. The new call LED is only extinguished when the last call has been accepted. An additional mechanism causes the new calls to ring the bell if the handset is replaced with outstanding associations. When building a generalised scheme several so called 'special features' automatically become available; for instance in this model an engaged tone no longer means 'try again later', but 'wait until you are connected'.

5.4.2 Call Forwarding

A common feature supplied by PABXs is the ability to redirect a call. ISLAND's model is slightly different from most. In figure 5.6, *Ringphone* A calls B and sets up an association. After a discussion with B, A realises that he wanted to talk to C but doesn't know the number. B can **forward** the call by manufacturing an associator which is queued in the new call FIFO of A and C. If C is free, the bell will ring and A can select the call using **accept**. The advantage is that A now has two associations and can select between them. If B hangs up and A selects that association, a request will be made asking A if the call is to be re-established. This scheme gives a client the control that most people would like when phoning into a large company and having their calls uncontrollably forwarded about the building. The ISLAND system ensures that a caller can always return to any of the intermediate associations, after all, in the example C may still not have been the target for A's call, and B may be the only person who could help.

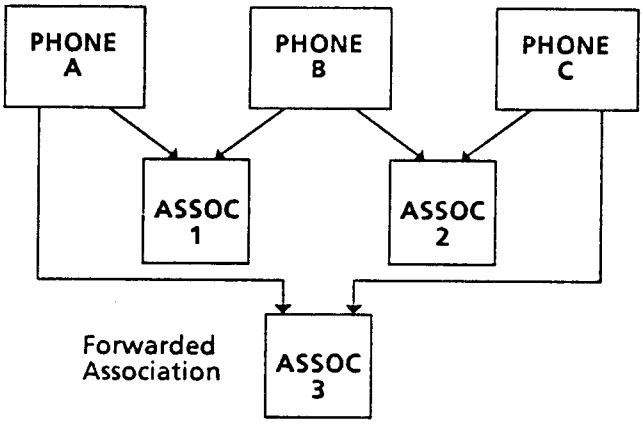

Figure 5.6: Call forwarding: a special approach

5.4.3 Logging-on to a Ringphone

A *Ringphone* can be associated with a phone-number identifier by using a combination of a publicly known number and a secret number passed to authorised people allowed to masquerade as that number. In the Cambridge Computer Laboratory, public numbers have been derived from the initials that normally make up their computer user-ids (user identifier) by implementing a simple letter to number encoding scheme similar to that found on older telephones in England. When using the **logon** function both numbers are prompted for in order to check the authenticity of the client before making the corresponding profile the active one associated with that station. Calls for that number will now be directed to the associated *Ringphone* which will also be personalised with the features described in the profile. Other features extending the **logon** function are:

- **who:** indicating the profile number active at a telephone
- **visiting:** a temporary logon without bringing up any features
- **logout:** returns a *Ringphone* to its default profile

5.4.4 Utilities

It was thought that some utility functions would be useful to satisfy a customer that the *Ringphone* equipment was working. Lamp and tone tests are provided for this purpose. Additional features allow conversations to be played from an amplifier, provide the cost of a call, and read the time. A *Ringphone* can also be made into an auto-answering telephone and may be useful for providing voice services e.g. 'dial a disc'.

Continuing on the theme of equal rights for both parties, a **pull_cost** feature enables both clients to be able to claim an arbitrary percentage of a call's cost. The payable percentage can of course only be increased while decreasing the cost to the connected party. The traditional reverse charge call is achieved by pulling 100% of the cost but for a costly call it may be better to split the cost equally at 50%.

5.4.5 Alarm Calls

Another common PABX feature is the **alarm** call. ISLAND has provided two alarm control functions. **Set** and **clear** permit an alarm call using a 24-hour clock to be set or cleared. Once setting an alarm to ring at a specific number, an associator is created and a process is forked which waits on a semaphore with a time-out set to coincide with the alarm time. On timing out, the process simply calls **establish** at the initiating **phone** object and the *Ringphone* is activated in the usual way with the caller displayed as 'ALARM'.

5.5 Guardians

Ringphones send events to a particular *Director* by randomly selecting a name from an internal table. When a *Director* is initialised it establishes its own entry in the table of every *Ringphone* known about by the directory (see section 5.6). An entry in this table is established by a *Director* for every **phone** object that is created. However, if a Ringphone were to fail in some way and its program were reloaded, the new table would be empty and event signals would not be signalled anywhere. To allow for *Ringphone* failures of this kind, a safety mechanism was designed. When a **phone** object is created, an additional process given the name of 'Guardian' is forked which sends an **enter_name** command to the corresponding *Ringphone* every 30 seconds. This function is implemented idempotently by checking to see if the name already exists in the table before entering it again. The

5.8 Synchronisation of Phone Processes

The mechanism described for associating **phone** objects had to be carefully synchronised to ensure that neither inconsistency nor deadlock could result from concurrent activities. A shared globally-accessible *monitor* lock (at one *Director*) had to be acquired by the event handling procedures for each instance of a **phone** object, but not by the ECHD interface. Atomicity is thus preserved at a level which serialises the state transitions of each **phone** object. If this had not been done, two active telephones might decide to call each other resulting in an inconsistent state. If, however, each object had its own *monitor* lock, events could be processed concurrently by many **phone** objects, and thus the ECHD interface would also need to acquire the same *monitor* lock. However, if two telephones simultaneously called each other, deadlock might result. The global lock has the advantage that state transitions are serialised at a level which can be seen to work effectively; however, the scheme is wasteful of processor time. Each state transition carries out a number of actions, some of which execute remotely. In the transit time for these remote actions the processor is idle, locking out all **phone** objects on the global *monitor* lock. Most CFSM transitions are implemented with on average three remote actions, and as a result the state change is relatively quick. This property has made it a workable implementation for the current experimental scheme. It would, however, be more important to optimise the use of processor time if many more client extensions were added to the system, making the probability of processing simultaneous telephone-events much higher.

An alternative approach is to use separate *monitor* locks on each **phone** object and extend the lock to monitor the association interface (ECHD), but to include a semaphore **wait** operation at the entry point of all association-object interface-procedures. The semaphore would need to be globally accessible to all association objects (at one *Director*) and could be initialised to allow the first queued process to continue execution without waiting, but to queue all that followed it. All exit points from the associator interface would **notify** the global semaphore. In this way all processes calling across association boundaries are serialised on the global semaphore. Associator interface procedures represent a considerably smaller percentage of the time spent in executing state transition procedures. When using this model, interleaved state transitions involving many parallel remote actions are possible without inconsistency or deadlock problems.

5.9 Control Management and Service Objects

One of the most desirable properties of a PABX/LAN system is that client *Ringphones* could be controlled from a workstation. Three mechanisms were designed. The first is a simple SSP interface to the *Ringphone*. It causes *Ringphone* events to be generated artificially. A client can dial and activate menu options without physically touching a telephone before it is necessary to pick up the handset to engage in conversation.

```
Activation Name:                    phone-con-<station name>
Function code:                      4000
String Character Arguments:
"+"      ->  Handset off-hook       "-"  ->  Handset on-hook
"<0-9>" ->  Normal digits           "*"  ->  Attention
"#"      ->  Enter key
```

This mechanism has only limited use in conjunction with a workstation, because it only offers features that are normally available in the *Ringphone* feature menu. However, it does offer a client access to common operations in a form which requires minimal effort to implement. For instance, a five-line shell-script can be written for Tripos which will activate any of the menu features at a particular *Ringphone*.

A second mechanism is needed to allow the workstation to be informed about incoming calls without the *Ringphone* sounding its bell. A possible scheme (although unimplemented) would be to put a field in the profile file which would contain the name of an interested party. The workstation would simply have to log that profile on to the *Ringphone* in order to tell the *Exchange* where to send the additional information. However, for a completely generalised control scheme the mechanism described in section 5.9.1 needs to be put into play.

5.9.1 The Service Selection Mechanism

Service numbers can be entered in the directory file along with those representing telephones, clients, and gateways, and given the **class** attribute of 'Service'. When a directory object is initialised, **Service** objects are created at the same time as the **phone** objects. They provide the same association interface as **phone** objects and an additional remote ECHD interface. Thus a remotely implemented service can use the same association protocols as the *Ringphone*. The only difference is that the *Ringphone* should not be configured to transmit voice to the associated service. The association is simply there to tell the service machine that it is entitled to take control of the *Ringphone* and may redirect the events it produces by rewriting its *event vector*. Control reliability is ensured through the '*' key which will always redirect the *event vector* to send events to the *Exchange*. Service machines are assumed to take absolute commands from the *Exchange* and to contain a network ECHD interface. A **hold** function call on the service from the *Exchange* will signal that it is taking back control, and the service should release the controlled *Ringphone* until it obtains a **connect** call.

All service implementations must take precautions to avoid interference with the *Exchange*. If there is an occasion in which the *Exchange* and a service are in disagreement (for example due to a protocol error) about who is controlling a *Ringhone*, the service must always back-down. Some precautions can be taken by the *Exchange* to protect the *Ringphone* against a disobedient server. One possibility would be to modify the *Control Channel* UID. This area would be suitable for further investigation.

5.10 Gateway Objects

A *Gateway* is not represented by a unique type but instead by the **phone** type, which is configured in a special mode of operation. This design decision was made because of the similarity of *Gateway* and *Ringphone* operation. Numbers dialled on ISLAND telephones which are intended for connection outside the network are prefixed by the digit '9'. The directory recognises a number to station search of this kind to be special and responds by returning the gateway-station name and a *subname*. *Subnames* are stored in an associator and normally contain null strings when associating two *Ringphones*. The *Gateway* acts as an auto-answering service, responding to calls when it is not engaged[5]. If it finds a *subname* which is not a null string, it will dial the digits contained in the string on the connecting telephone network. Incoming calls from the connecting network are also handled by auto-answering the call and then interpreting the digits as if they had been dialled on a *Ringphone* at that station. Thus associations between outside telephones and *Ringphones* can be set up in either direction through a *Gateway*.

5.11 Summary

A model has been described which is suitable for implementing numerous features, many of which are easier to use than those provided by commercial PABXs. The feature menu allows for incremental additions to the *Exchange* facilities, although it is intented that most new services should be provided by additional network servers called up by dialling numbers. These numbers may be made available by creating new entries in the directory. The mechanism for achieving an association with an external service permits control of telephone resources to be passed to that service. Reliable access to the *Exchange* is maintained by providing the '*' key on a *Ringphone* console, the operation of which will always return control to the *Exchange* service. A client may then select the required telephone function by using the feature menu.

The essence of ISLAND's model for the *Exchange* is that telephones, services, and associations are modelled by objects which are assembled into a data structure representing and controlling the real situation. An important aspect of associating two objects is that only four primitives (the ECHD interface) are needed to change the status of an association. This property is made full use of when extending control of associations between *Directors* and is fully described in chapter 6.

[5]The Gateway only connects to a single external extension-line, and can only handle one outside call at a time.

Chapter 6

Distributing Control

A principal aim of the ISLAND demonstration system is that it should offer a quality of service which surpasses the abilities of most commercial systems. Chapters 4 and 5 have already demonstrated the flexibility and variety of features made possible by integrating a PABX into a computing environment. This chapter describes the mechanisms by which the service can be distributed over many machines in order to ensure highly available operation.

Typical private PABXs supporting 200 extensions are designed to have a mean time between complete system-failures of 17 years [BT 83]. This standard is supported by field trials of existing systems over a long period of time. It is very difficult to predict this kind of information from first principles because of the number of system variables involved. If the distributed PABX/LAN is to compete with these requirements it must overcome several problems. A major factor is that it is exposed to more environmental conditions than a centralised system. The network, however, can have connective redundancy (in the form of braided rings and self-healing rings) in order to solve this problem. If the CMDS processor-bank philosophy is used, the processors can be given the same kind of environmental protection as a centralised exchange.

A far more difficult problem is to find ways of correctly handling faulty server-hardware or faulty software, while continuing to provide a service. ISLAND has designed a system which can be scaled, through distributed control, to provide arbitrary fault tolerance in many of the common failure situations.

6.1 Fault Tolerance

Reliability of a system can be measured as the the Mean Time Between Failures (MTBF) [Shooman 79], but this is insufficient to describe the effect that the failure has on its environment.

A second parameter, the Mean Time To Repair (MTTR), defines the period of time for which the machine is likely to be unavailable. The availability can therefore be defined as a percentage.

```
(MTBF/(MTBF+MTTR)) x 100
```

If a system's MTBF and its MTTR are very small, the overall effect will be high availability, even though the system components are clearly unreliable. Provided that the loss of state between failures has no significant effect, the poor reliability of components will not be noticed and the system will be considered to be reliable. Such systems are termed fault tolerant or *resilient*. An availability figure must be quoted along with the MTTR otherwise it is not possible to judge the effect that it has on an application. If a *resilient* system interacts with a human operator in real-time, the MTTR must be chosen so that it is not perceivable. In other words a MTTR below human reaction time must be chosen together with an availability so that the loss in performance of the system is not obvious: perhaps as much as a 30% loss of performance would not be noticeable.

Fault tolerance is achieved by adding redundant parts to a system in such a way that the system no longer depends on single components for correct operation. In general, it is possible to increase the redundancy of a system to produce an arbitrarily small probability of failure (this proposition is based on the assumption that the probability of failure of any redundant component is independent of the others). Unfortunately this is difficult to guarantee, and situations are often possible in which a fault will propagate across all redundant parts. Therefore it is essential to ensure that faults are confined to the area in which they have occurred.

A PABX must have high availability of service and a low MTTR. The reliability of conventional processing components and their operating systems is notoriously bad because of the complexity and interdependency of components contained therein. Processor replication was therefore a necessary design approach. The CMDS processor-bank provided an important resource for the design and implementation of a replicated system.

6.1.1 Manifestations of Failure

Network servers may experience faults which cause the following types of failure:

- Fail-stop
- Erroneous but continued operation
- Transient or intermittent failure

A fault is realised as an inconsistent machine state. Faults which cause the processor to fail and halt its operation are the least undesirable because the fault state

can be isolated, examined and, hopefully the cause determined in full. Erroneous operation is a rare event because of the destructive effects that most faults have on a system. Transient failures may be related to either of the other two cases, and are the most difficult to detect because the cause of the problem may not be obvious.

In a distributed system, failures may occur in the host hardware or the network interface. Servers may contain faulty software resulting from many causes. There may be timing errors in an operating system, for example when synchronising concurrent activities. The compiler with which a system was built may have generated incorrect code and of course the application program may contain logical faults. A new application program is more likely to fail than one which has been in use for a long time. However some failures may result from the occurrence of a probabilistic event combined with a rare system state. Long term testing may be the only way to find this kind of a problem.

6.1.2 Byzantine Failures

A generalised study of the difficulty of reaching agreement in the presence of faults in a distributed system is given in 'The Byzantine Generals Problem' [Lamport 80]. An analogy is made between generals on a battlefield trying to reach co-operative agreement on their battle strategy despite some traitors among them, and that of network servers trying to perform a unified network task even though some servers may be operating erroneously and sending misleading messages. The problem is to determine the ratio of m traitors to n generals that can be tolerated before an incorrect action is taken. The result is $n \geq 3m + 1$. An interesting aspect of the problem is to reduce the communication costs to a minimum whilst still tolerating the maximum number of traitors given by this condition. There have been many solutions [Dolev 81] [Chor 85], some with connectivity constraints between the generals. The problem, however, is slightly removed from reality because it assumes that a traitor may be malicious in its activity. ISLAND has assumed that failures will result in inactivity at a network server. Intermittent failures of this kind are also considered to be possible. This is a reasonable assumption because the coupling between network servers through the LAN is very loose, and a failed server would have to send self-consistent but malicious messages before it could cause this kind of effect. The probability of this happening as the result of random failure is very small. The Byzantine Generals problem is far more suited to a network in which some of the nodes are subject to malicious human intervention. The components of a processor-bank can be protected from this eventuality by keeping them in a secure environment, and the messages themselves can be protected by using encryption to guarantee their authenticity.

6.1.3 Enhancements for Fault Tolerance

There are no solutions which guarantee that an implementation will be totally resilient to faults, but some steps can be taken to remove single component dependencies or to prevent the propagation of errors.

N-version programming. This technique uses a specification language or a formal description to define a problem. N different implementations are then made to work in parallel and compare their intermediate and final results by taking a majority vote on the correct solution [Chen 78]. The method aims to derive N perfect implementations, but if any of them have hidden logical errors or ambiguous states it is less likely that they will be made in the same places by different programmers. The solution is further enhanced by applying the same technique to the compiler or by using different languages. A related example of this kind of work is in the flight computer of NASA's Space Shuttle. In normal operation it uses a four-processor parallel-computation system which votes on the validity of calculated data before offering its results. There is a fifth, completely separate, flight computer [Gifford 84] which operates in a listening mode, gathering up-to-date flight information. At any time it can be manually switched in by the crew to function as the primary computer. However, the fifth computer runs software written independently of the other system in order to remove the possibility of a generic programming error.

N-version programming requires a large investment of time and resources to remove generic errors. Common pragmatic restrictions are economics and a system designer's obligation to ensure correct operation: these two factors will ultimately dictate the degree of fault-tolerant design in a system.

Proving Programs Correct. There has been a great deal of work in recent years to provide formal specifications of programs and methods of verifying that a program's implementation meets its specification. The scope of this subject has great potential for the future, but currently it is confined to relatively simple systems. Functional programming languages show great promise for this kind of work but provide little support when building systems which require concurrent techniques, synchronisation primitives, and support for remote operations.

6.2 Models for Fault Tolerance

6.2.1 Hot-standby Systems

This is an approach to fault-tolerance in which a system is replicated but where only one of the components actively carries out its operation. In the event of failure a comparatively 'fail-safe' mechanism will switch in one of the standbys making it the new active component. Crude systems will simply switch in alternative hardware when faced with failure, even though previous system state may be lost. A more general system in which the integrity of state is important, will

regularly deposit state or *Checkpoints* to the dormant components enabling them to take over control at a state close to that of the active instance when it fails. If a system adopts a policy whereby it assumes a previously defined state, it is commonly termed a *roll-back* mechanism. If the backup system can calculate the state it should currently be in using a previous *checkpoint* and the elapsed time, it is termed a *roll-forward* mechanism. Examples of systems designed using the hot-standby philosophy include:

Tandem Systems.

Tandem/16 non-stop systems [Bartlett 81] was the first commercial organisation to venture into fault-tolerant computer-systems, and has been a leader in the field since 1977. Their systems are built from up to 16 processors on a bus, operating independently and linked by dual data-paths. All peripherals are dual-ported, permitting two independent access points and are powered in isolation. **Tandem**/16 executes a multi-tasking operating system which supports the reliable implementation of applications run as *process pairs*. Each active process, the primary, makes checkpoints to a secondary process on another processor providing state backup in the event of failure. Every processor is monitored by all the others. Each one must broadcast an 'aliveness' message over the inter-processor bus every 2 seconds. If any processor fails to do this, the remainder consider it to have failed. When a single-point processor failure occurs removing half of some *process pairs*, the remaining single processes become the primary and continue operation. If the faulty processor is fixed, the presence of aliveness messages causes the primaries to generate new secondaries and the redundancy is restored. The mechanism only guarantees recovery from a single point of failure. If the processors are reliable enough, this is normally long enough for an operator to effect the repairs to the on-line system. The system architecture will sometimes do better than to tolerate a single point of failure, since a number of faults may independently disrupt different process pairs. These systems are successful because the MTBF is much greater that the MTTR and thus the system hardly ever experiences more than one fault and does not appear to stop.

Rebus

Rebus [Ayache 82] is a fault-tolerant distributed system designed for real-time control applications. It has been used in the MODUMAT 800 industrial general-purpose control system sold by Sereg-Schlumberger. **Rebus** uses a distributed architecture without relying on any centralised memory or any central processing component. Servers are connected by a linear bus that uses a dual data-path to increase its reliability. A number of servers are allocated for an application, one of which becomes the primary. The primary checkpoints its data to the other servers. If it fails, a virtual ring protocol [Le Lann 77] is used to sequence the order in which secondaries become the primary. This protocol ensures an orderly agreement on the server that takes over control of the application. **Rebus** has been successfully used in industrial situations in which the reliability requirements have been met.

ISIS

ISIS [Birman 85] is an experimental system for the development of resilient objects in a distributed environment. The project set out to design a system which could implement a distributed application with an arbitrary degree of fault tolerance. The **ISIS** system tolerates **k** failures out of **k+1** processors[1], and allows the automatic restart of hardware and software components after repair. **ISIS** employs *roll-forward* techniques which enables progress to be made in the event of failures and the system as a whole behaves as if no failures occur.

Objects and processes are replicated and distributed by the system **k** times in the form of *process groups*. Each *process group* contains a *Coordinator* and **k**-replicated component parts called *Cohorts*. Checkpoints are continuously sent by the active *Coordinator* to the *Cohorts* using various broadcast protocols which guarantee the order in which messages are delivered. If the *Coordinator* fails, then the *Cohorts* elect a new *Coordinator* amongst themselves, and the application continues. To enable a conventional distributed object-oriented program to be made **k**-resilient, ISIS provides a preprocessor to carry out the automatic conversion of objects into this resilient form.

Atomic Stable Storage

Argus [Liskov 82] is an example of a distributed development environment that employs atomic stable-storage. This system achieves fault tolerance by backing up state at a remote server containing a non-volatile storage medium, such as a disk. This system also employs Atomic Transactions to ensure consistency of system state in the event of failures. An Atomic Transaction is a sequence of actions that must complete in full or *roll-back* to establish a state as if the transaction had never begun. The execution of atomic actions in a concurrent environment must be atomic with respect to a single thread of control and equivalent to some serial execution of these actions.

Objects in **Argus** may be defined to be volatile or non-volatile. Volatile objects will be lost if their host fails. When non-volatile objects are created or modified their state is backed-up by sending messages to a stable disk. It is necessary to use a two-phase commit protocol [Gray 78] to ensure that the state of the disk is atomically updated. If a machine fails and is later reinstated, it will execute recovery procedures to recover its stable state and reconstruct the non-volatile objects. The volatile objects can then be reinitialised with a state representation that is acceptable for restarting the system.

6.2.2 Parallel Replicated Systems

An alternative to the 'Hot-standby' approach to ensure fault tolerance is the 'Parallel replicated' system. These systems utilise replicated processors to carry out the same computation in parallel, the results are then compared in a similar way

[1] This property has been given the term 'k-resilience' in literature on this subject.

to N-version systems. This technique can be applied equally well to single-version replicated-systems. A majority function is a typical means of combining parallel results and thus deciding on a course of action. There are a number of systems which have adopted this technique. In general it is more applicable to systems that need to make fast calculations with a high degree of integrity and cannot afford to tolerate a recovery delay.

August systems.

August [Wendsley 85] is an American company which produces industrial control computers. Their products have been used for flight-control in aviation, in components of the space programme and also in public telephone systems. The **August**/300 systems are pitched at applications where reliability is far more important than just guaranteeing availability. In some systems the occurrence of even small failures while a system reconfigures itself can lead to catastrophic results. The main principle of operation is the use of 'Triple Modular Redundancy' (TMR). Input data is collected by interface modules and independently passed on to three separate processing units where parallel calculations are performed. The results are collected, synchronised, and then fed to a *voter* which performs a logical majority function on the data. The system can therefore tolerate a single processing failure. The majority function must also deal with failures amongst its own components. A redundant design has been used which allows for a single internal failure. Some of the designs adopted by **August** systems originated from the design of the SIFT flight computer [Melliar-Smith 82] which also used correctness proofs to validate the software architecture of its system.

Stratus

Stratus [Wilson 85] is a highly resilient system which uses a combination of replicated hardware and transaction based software. Stratus hardware is characterised by processor boards that contain dual logic, dual data-pathways and a comparator component. On a processor board two simultaneous calculations are carried out and fed to the comparator, signalling a failure when they differ. In a system configured for a high degree of fault tolerance, these processor boards are mounted (and operate) in pairs. If one board operating to provide the primary output fails, the other will be switched in as a backup. **Stratus** is therefore able to confine faults effectively.

Circus

Circus is a project undertaken at the University of California, Berkeley, to research into a mechanism for constructing reliable distributed programs [Cooper 85a]. The approach used in Circus is to replicate the objects in a distributed program into sets called *troupes*, the replication factor of which may be changed dynamically. The state of each *troupe* member is kept in step with the others by parallel execution of the distributed task. Although inter-*troupe* calls are specified as ordinary procedure calls between objects, the run-time system converts them into a multicast/broadcast protocol termed 'Replicated Procedure Call'. Failures will not stop the program executing, provided that they are not malicious failures and

6.3 ISLAND's Approach to Fault Tolerance

Parallel replicated systems achieve fault tolerance without having to employ a recovery phase and thus failures do not incur a time penalty. However, such systems are costly on resources and do not gain any performance benefits for increased replication, in the short term. Furthermore, because computation results must be synchronised and combined in the parallel replicated approach, they are more easily implemented by closely-coupled hardware than by distributed systems. ISLAND's needs are more oriented towards a highly available service rather than to maintaining data to a high degree of integrity.

Stable storage was an option for backing-up state associated with the ISLAND *Exchange*. However, recovery must be carried out in a real-time environment, and must be carried out fast enough to avoid annoyance to the subscriber community. In the CMDS, the filing service is often heavily loaded by client demands, and recovery of information may be slow. A more significant problem is that there is no guarantee that it will be available. The filing system, although of high integrity, would have to provide a service as available as the *Exchange* or else it would become a dependency for the service. Its use as a means of backing-up state was therefore rejected.

A Hot-standby system can utilise replicated hardware to increase fault tolerance and performance at the same time and as a result, this approach is the most applicable to distributing the ISLAND *Exchange*. ISLAND's building blocks are not tightly coupled processors, but distributed servers which can be dynamically allocated to a task. This environment provides a flexible foundation for a resilient design because a service can be created with dynamic fault-tolerance and on-line replacement of faulty components.

In ISLAND an extensible system is needed to provide a high quality of service for a growing population of extensions and subscribers, whilst at the same time maintaining fault-tolerance. Performance may be increased by distributing the load on a system equally over a number of processing elements. Fault tolerance may be gained by using each processor to back-up its active state to the others. Thus a server would be both active in managing the system, and acting as a repository for backing-up state from the other system components. Such a scheme also has the advantage that all the code is being exercised all the time and therefore removes the possibility that a dormant failure in a server will only show itself when it becomes the active management component.

6.3.1 Distributing Control

If a load is to be distributed over a number of processors, this gives rise to the problem of dividing the computation task up in a way that minimises inter-processor communication. The ISLAND *Exchange* is a very specific application and can be dealt with in a way that is tailored to its needs.

Some observations can be made about this kind of distributed exchange service. Firstly, if the load on the service provided by a population of *Ringphones* is spread over a number of servers, the only time an inter-server control-message needs to be sent is when a *Ringphone* attempts to contact another *Ringphone* which is found to be 'engaged' and is being controlled by a different *Exchange* server. All calls made to *Ringphones* which are not 'engaged' will be handled by the same server and clearly, this will sometimes be true of a *Ringphone* which is engaged.

Another observation is that the only **phone**-associator-**phone** interactions necessary are those using the ECHD interface explained in chapter 5. In order to extend this interface across server boundaries, the component procedures need to be made available as remote operations. CLU remote procedure call makes this extension possible with a minimal amount of extra coding.

A mechanism to make use of these properties is described here. *Ringphones* can be considered as the shared resource of the *Exchange* components (the *Directors*). In order to enable synchronisation between the *Directors*, a concurrency-control locking operation was added to the *Ringphone* network interface. The lock is not physically enforced on the premise that the *Directors* will never act maliciously. This boolean lock is controlled by the atomic procedure 'Test, Set and Read' (TSR). The request takes a boolean value and a name string. It returns a boolean value and the name of the last machine to set the lock. Trying to set a lock that is already set will return a **true** value and the name of the *Director* currently in control. If it is free, the lock will be acquired establishing the requester's name as the controller. It is an extended form of the more conventional 'Test and set' operation but provides a handle on the server currently in control. Figure 6.1 shows how the lock can be used to synchronise control fought for by two servers over one *Ringphone*. *Ringphones* B and C are involved in an association set up by *Director* 2. A trys to ring B, and B's corresponding **phone** object determines that its physical *Ringphone* is engaged by executing an **establish** operation on B's object. One of the first actions in the **establish** (part of the ECHD interface) tests the internal lock of *Ringphone* B using the TSR operation, and determines that it is engaged and controlled by *Director* 2. It is not enough to only report this event back to A, because an association needs to be made for A and B at *Director* 1. The remote version of **establish** is therefore called at *Director* 2. The associator is the only data that needs to be conveyed in the RPC[2], because the data structures representing all the *Ringphones* are replicated in all the *Directors*.

[2]CLU RPC allows the dynamic binding of RPCs to network addresses. A network address can be calculated from the textual name of the destination server and the subaddress (port) on which it is to be received.

A similar mechanism is used to extend **connect**, **hold** and **delete** to be remote operations. The overall effect is that **phone** objects make calls to other **phone** objects through associators, but the control mechanism is hidden from the client when control is extended to a remote server.

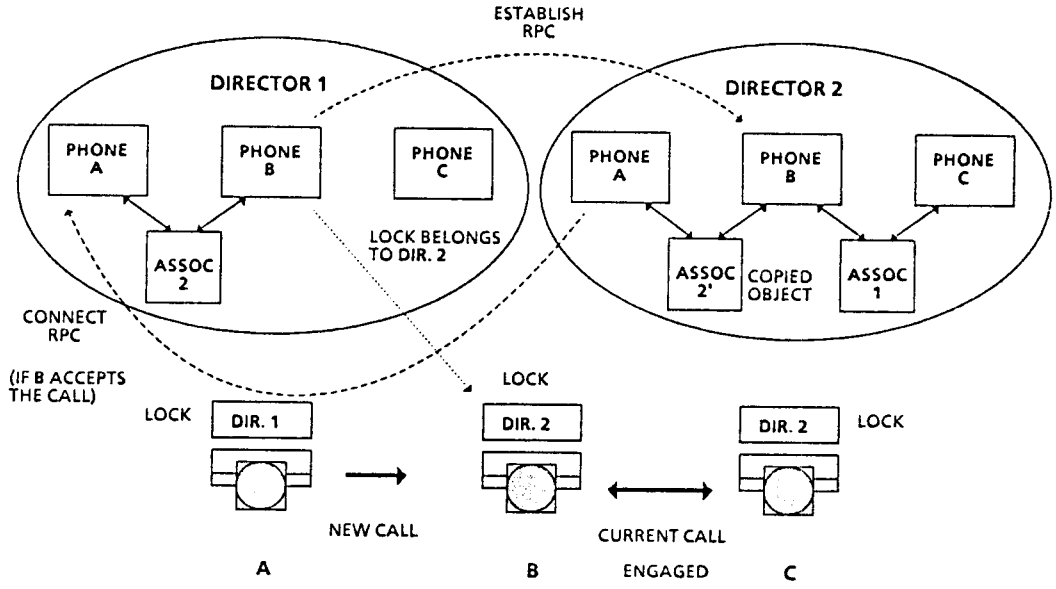

Figure 6.1: Distributing control between the *Directors*

6.3.2 Failure Modes when Distributing Control

Failed remote ECHD operations will always secure the association to a safe state and prompt the client for retry action. There is, however, a synchronisation pitfall which can occur in this association model. The single server model enjoyed the benefit of a global lock to ensure that only one *Ringphone* was making a CFSM transition at any time, and hence the ECHD procedures did not need to be *monitored*. The global lock cannot be shared by all instances of the *Director* unless a complex remote locking scheme is designed at a much greater cost to normal operation. The situation results in the possibility that two remote operations may be executed on mutual phone objects across a server boundary, causing inconsistent data. The solution is to force the remote ECHD interface to acquire the monitor lock at the server in which it is executing. This solution resurrects one of the problems considered when evaluating the use of a separate lock on each instance of a **phone** object. There is now a danger of *deadlock* between two servers making simultaneous calls on each other through the same associator. However, because the operations are remote, each will be timed-out by its own client and the condition caught by an exception handler that prompts the client for a repeated action. The effect is sufficiently random that the event is unlikely to occur often. In fact, this event seems to be so rare that it has never been observed whilst the ISLAND

experimental system has been running.

6.4 Summary

This chapter has described the various options available for designing a distributed fault-tolerant service. In the ISLAND application, faults are assumed to be in the fail-stop category and the designs have not considered the possibility of a maliciously operating server. N-version programming is a powerful way of removing implementation faults but is too costly on resources to be practical in most cases. The 'Hot-standby' approach is more applicable than the 'Parallel replicated' approach for designing a distributed exchange service, because it is primarily concerned with supplying a highly available service. By the equal sharing (as the result of random *Director* selection) of the load presented to the service across a number of servers, performance may also be increased. It is possible to distribute control in this way because the interaction that must occur between servers is very small and it can be achieved by extending the ECHD interface (described in chapter 5), to the server network-interfaces.

Chapter 7 considers how the design described so far can be taken a stage further to allow: the distribution of critical state information, the detection of server failures, and the recovery of state after a failure.

Chapter 7

Fault Detection, Checkpointing and Recovery

7.1 Overview

Chapters 5 and 6 have described a framework whereby a replicated fault-tolerant exchange service can be performed by a variation on the 'Hot stand-by' design approach. The primary benefit of this choice is that it is possible to enhance both fault tolerance and performance by increasing the replication factor. Furthermore, in the event that some replicated components fail, service performance will only be reduced in proportion to the loss.

To achieve fault-tolerance in a system that contains backed-up state the following issues must be considered:

- Fault Detection
- Fault Confinement
- Reconfiguration
- State Recovery
- Re-start

These operations must be carried out within a 'critical' period of time for a recovery action to be acceptable. The key word is critical. In the context of an exchange, an order of magnitude times that of typical human reaction time (0.3s [Smith 85]), i.e. about 3 seconds, is a reasonable upper limit on this delay. However, if the fault-tolerant system were controlling the flight path of a rocket, for example in NASA's space shuttle guidance system, the critical time would be about 20 milliseconds [Gifford 84]. A recovery taking significantly longer than this would not be considered fault-tolerant for obvious reasons. 'State recovery' implies a

previously shared knowledge of the overall system state prior to failure. This can be carried out in the normal operation of a distributed system by exchanging state information contained in inter-server messages. These state descriptions are usually called *checkpoints* and for the purpose of efficiency, they usually contain a compressed version of the state they represent. At a recovery stage this information can be used by the operational servers to rebuild the data structures which were present in the failed servers. In a system which can dynamically allocate servers to an application, new servers may be made available automatically to replace the failed ones. Alternatively, a failed server may simply be removed from service and it is up to a human operator to allocate more servers to the application.

The following sections describe the pragmatic issues and solutions which were designed to detect faults and to recover from them.

7.2 Fault Detection

A fault in a system may be defined as an event which causes erroneous information (an error) to be present in that system. A fault may cause a system to adopt a state that is inconsistent with its own specification, and when this state is read by the processes executing within the system will result in a system failure [Randell 78].

Faults may cause data to be lost, corrupted into a nonsensical state, or transformed into valid but misleading information. In a distributed environment lost messages may be detected by time-out strategies. Corruption can be determined to an arbitrarily small probability by using redundancy codes e.g. Hamming Codes [Peterson 72]. A corrupted but nevertheless self consistent message is only recognizable in the context of other information. However, as explained in chapter 6, these kind of errors are considered to be extremely unlikely and as a result no defence has been offered against them.

ISLAND has only concerned itself with faults of the first and second type. The Basic Block Protocol (BBP), used as the basis for all communication on the Cambridge Ring, contains a checksum for the purpose of detecting transient errors in inter-server messages. A block with a bad checksum is simply thrown away. In a higher-level request/reply protocol a bad checksum must result in the same action as if the the request or its reply had been lost in transit. A corrective solution is for the client to time the reply out and then retransmit the request block again.

In the context of the ISLAND *Exchange* the detection of a fault in a remote machine can be established by its lack of response to network messages.

Most faults manifest themselves as a complete server failure; however, in many operating systems there is one notable exception. If a process fails due to some logical error, it is often only that thread of control which is halted, and its failure may not be detected by a remote message probe. There are also many cases when a process may have acquired a monitor lock during its execution, and when that process fails the monitored objects will continue to be locked. The locks may

block other processes from running and partially inhibit, if not stop, the normal operation of the system.

ISLAND has assumed that logical failure will cause processes which experience failure to stop executing. For the *Exchange* to detect failures in its component servers a probe mechanism was required. The following issues were relevant to its design.

A very simple way to probe a remote server is to make use of an existing remote procedure call mechanism. The failure of a procedure offered by one server and remotely called by a party periodically monitoring it, can be an effective way of determining the failure of that server. In the Mayflower/CLU environment when an **At most once** remote procedure call is activated at a destination machine it will generate a new process to handle the call. This will terminate in due course when the procedure returns. If a critical error occurs in the server hardware or in the execution of server code, it will not be possible to reply and the call will time out, indicating failure.

One method of detecting a failure amongst many different threads of control in a server is to use the probe to call a procedure that tests all of the major monitor locks within the probed server. If any of these are held for more than a short period of time, the remote probe will time-out. Within the ISLAND *Exchange* the shared monitor-lock, used to serialise all events destined for a **phone** object, is a necessary target for such a test. A lock can be tested very easily by calling a null procedure which needs to acquire the lock before it can execute. The type **phone** has a procedure defined for this purpose.

An additional mechanism is necessary to allow all the replicated components to be informed about the fault. A significant issue is that no server should have the sole responsibility for detecting failure, as this would result in a single-point dependency, and thus defeat the availability requirement. Instead, the responsibility should be equally distributed. One possible scenario would be for each server to probe every other server. This satisfies the requirement, but leads to a cost of order n^2 messages for n nodes. If many servers were used and the fault detection period was required to be small, the servers would experience an excessive load. In practice it would not be necessary to have more than about five servers to ensure the required availability of service. Nevertheless a less costly and more scalable approach was considered.

If all the servers are arranged into a logical-ring ordered by their station name, each server can be given the responsibility of monitoring the failure of the next server in the ring. When the replicated service is established each server can also be informed of the names of the other servers in the ring. The servers making up the logical-ring, and any attributes each of them may have, will be referred to as a *view* of the set. If a failure is detected, the remainder of the logical-ring can be checked for its integrity, and the information passed on to the other members. If the server distributing this information fails, the probe that is monitoring it will detect the failure, and thus the information will be propagated. In normal

operation this scheme only generates the same number of probes-per-validation as there are servers. It is only when a failure has occurred that the cost increases. From now on this ring arrangement will be referred to as a *Virtual Ring* (Note: the use of the *Virtual Ring* described in this text is different from the uses of a virtual ring in the mutual exclusion scheme of G. Le Lann [Le Lann 77] and also in the virtual resource ring used as a resource manager at the Univ. of Sussex UK [Hull 84]).

This kind of fault monitor was adopted in the ISLAND *Exchange*. Its implementation was based around a process called a **kicker** and a remote procedure called a **sink**. The **kicker** periodically tests the remote server by calling its **sink** procedure (see figure 7.1). All servers must have a **kicker** and a **sink** and each server must asynchronously call the next server in the ring in order to ensure that the operations are independent. The asynchronous operation makes the technique perfectly scalable, because the fault detection time is independent of the number of servers in the Virtual Ring.

A call on the **sink** procedure will signal an exception in the event of unrecoverable conditions such as the lack of a response from the destination server or failure of the network. Because the Cambridge Ring is an active network many types of communication failure are distinguishable. The hardware provides low-level data about the way a destination server has rejected a minipacket. The lack of a valid train of ring slots is also distinguishable at the network interface. An active network has a distinct advantage over passive networks (e.g. Ethernet) for diagnosing faults of this type. However active components are more prone to failure. In the case of a network break all inter-node communication is brought to a standstill and therefore retrying is the only sensible thing to do. A network server[1] should assume that the network will eventually be repaired, and that its request will eventually be satisfied.

[1]The servers referred to here are the type used in the CMDS processor bank. Each server contains memory and has a network interface as its only peripheral.

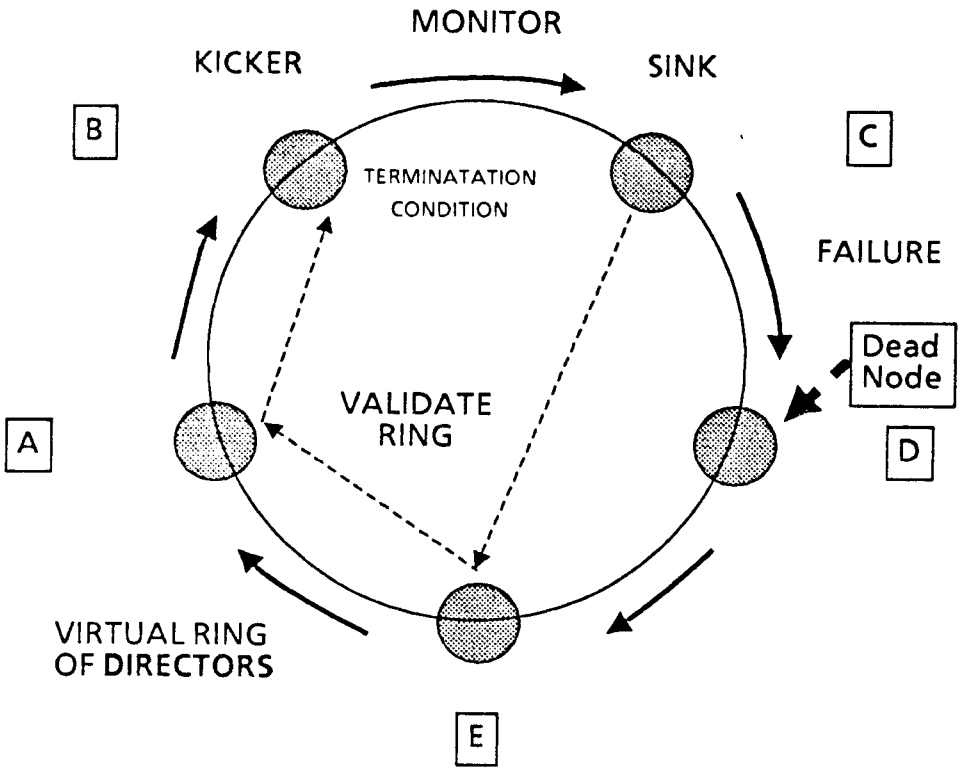

Figure 7.1: A Virtual Ring of *Directors*

7.3 Fault Confinement

Once a fault has been detected in a system which must be highly available, it is important that the fault should not be propagated to other components which are still operating correctly. In a distributed system, the loosely coupled architecture has a natural bounding effect on the propagation of faults. The failure of hardware in a server generally confines itself to that network node. The only way a fault can propagate is by way of inter-server communication. Modularisation is the key to fault confinement. Object-orientated programming extends this philosophy to the development of system software.

There is, however, a further consideration that must be made. A recovery mechanism introduces the possibility that a failed server may inadvertently spread faults to the distributed components while recovering a failed servers state. If a system were being designed to make arbitrary distributed-programs fault tolerant, it would be a difficult design issue. In the ISLAND system the state held by the *Exchange* is of a very specific nature, and the consequences of its recovery can easily be checked for the propagation of these errors, therefore no further considerations have been made. In a system which cannot afford to take this kind of risk in the event of a failure, a safer way to make a recovery is to introduce a new

server[2] before any of the remaining servers recover state. In this system, if the fault propagates, there will always be a pool of working servers. If the system is designed well the remaining servers could continue the service although probably with a lower performance due to the recurring recovery problem.

7.4 Reconfiguration

A fault detected amongst the replicated components of the *Exchange* must be announced to all other components, enabling a valid *view* of the set to be established at each server. There were three main techniques by which this information could have been distributed.

- Broadcast
- Multicast
- A Virtual Ring protocol

If a network supports a Broadcast protocol, this is the fastest and least costly method to distribute data. However, a broadcast mechanism is difficult to scale when several networks are linked together. Multicast is a more costly[3] but scalable method with an equivalent effect. One disadvantage is that a single server has all of the responsibility for distributing the information to all other servers. Furthermore, additional problems occur when one or more servers fail during the state distribution. The result will be several servers trying to distribute a *view* that each considers to be the correct representation of the set. There are no guarantees about the order in which messages will be processed by the servers; it will require a complex scheme to ensure that some servers do not adopt a *view* that is out of date. A Virtual Ring protocol was adopted as a more effective means for distributing the *view* of a set. The principles behind this protocol are described below.

7.4.1 A Virtual Ring Distribution Protocol

The essence of the Virtual Ring protocol is a recursive RPC between members of a logical-ring that is terminated by a call which is about to be made to the originating server (see figure 7.1). A possible specification of this recursive procedure is given below.

---

[2] A 'new server' in this context may be one which has simply been reloaded with a fresh copy of the application program. Logical failures in a program are unlikely to cause any physical damage.

[3] The Multicast is expensive because it involves sending and receiving all the distribution messages when implementing it on a Cambridge Ring.

```
VALIDATE_RING  = remoteproc( start:          string,
                             new_view:       array[string] )
                 returns(array[string])
                 signals(starter_failed)
```

In this example *view* is represented as an array of strings, each string is the station name of an *Exchange* server making up a Virtual Ring. The network address of a server can be obtained by applying a simple function to its name string. The procedure validate_ring has the following skeleton implementation.

```
(1) view := new_view
(2) Determine the new value of next_name and next_address
(3) IF start = next_name THEN return(view) END
(4) view := CALL validate_ring(start_name, view) AT next_address
   EXCEPT WHEN rpc_failure:
   (4.1) IF the network has failed THEN repeat (4) END
   (4.2) Remove next_name from new_view
   (4.3) Recalculate next_name and next_address
   (4.4) IF start = next_name THEN return(view) END
   (4.5) repeat (4)
   END
(5) RETURN(view)
```

This procedure is conceptually simple, it has the same communication cost as the Multicast, and spreads the responsibility for state distribution over many servers. It also recovers easily from errors occurring in intermediate servers while the recursive RPC is still being executed. Automatic re-routing of a call around failed servers and their removal from a *view* is carried out in steps 4.1 to 4.5. The next section (7.4.2) has been included to explain the significance of RPC failures in the Cambridge Concurrent CLU implementation.

The function employed to calculate *next_name* is implemented by searching for the host server's own name in the circularly ordered list and selecting the next entry. The recursive remote procedure will terminate if it finds itself about to contact the server in which the RPC originated, in which case it returns the validated *view*. If one of these servers fail at this point, the call will be further propagated by the preceding servers.

It is also worth considering that two or more failures determined simultaneously by several kicker routines will cause many validate_ring procedures to be in progress at the same time. However this eventuality should have no effect on the validity of the final *view* held by each server. In this case the last return from an execution of the validation procedure will cause all the servers to reach the same consistent *view* irrespective of previous validations.

7.4.2 The Cambridge CLU RPC Mechanism

Remote Procedure Call and concurrency features were additions made to the original CLU specification at the University of Cambridge Computer Laboratory [Hamilton 84]. Two types of RPC were designed for use in distributed applications.

- **Maybe** RPCs [At most once]
- **Zealously** RPCs [Exactly once]

Maybe RPCs are lightweight calls employing a one-off transmission and a time-out for reception. **Maybe** RPCs were considered for use with the Virtual Ring protocol, but the calculation of time-out values is clumsy. Consider **n** servers involved in the Virtual Ring protocol and two arbitrary servers in the ring **r** and **r-1**. If we consider a fault is indicated by a time-out **t** when **r-1** calls **r**, then the time-out used by **r-2** making the call on **r-1** must be **2t** since **r-1** may fail, and **r-2** may have to call **r** directly. The time-out used for the first server in an **n** server ring must therefore be 2^n**t**. The shortcoming of this technique is that on the majority of occasions an RPC failure will cause a client server to wait a longer amount of time than necessary before the failure is acted upon. This is because on the majority of occasions only one intermediate server will have failed. The server preceding the failed one must wait the additional time as it cannot be certain that other time-outs are not taking place further around the ring.

Zealously RPCs, however, adopt a protocol that ensures the request is delivered to the destination with a high degree of certainty, and that despite an automatic retry mechanism the server executes the request only once. While the call is executing, a mini-probe RPC is sent every 2 seconds to check the aliveness of the server. If 50 successive mini-probes fail before the results are returned, the RPC is aborted. The results are also sent back as many as 50 times until an acknowledgement is received. The retry and time-out parameters for **Zealously** RPCs were not adjustable in the original implementation. A complete failure of the target server would therefore result in a 100 (50 x 2) second time-out, a time too great to be used for ISLAND purposes. However, the mini-probe used by the **Zealously** RPC mechanism makes it unnecessary to calculate time-outs if used in the Virtual Ring protocol. Each intermediate stage of a recursive RPC will have its own mini-probes acknowledged when the call takes longer than 2 seconds to execute due to other failures. The mini-probe, a simple RPC, provides evidence that the call is still in progress and should not be timed-out. In a more common failure mode there will be only one failure, the one detected by the **kicker**, and the resulting reconfiguration will be a straightforward recursive procedure without any mini-probe calls.

In order to take advantage of this **Zealously** RPC mechanism, ISLAND performed a number of small modifications to its implementation. The number of retrys and time-out per retry information had originally been bound into the **network_address** object as manifest constants. The ISLAND version added another

kind of create operation for objects of the type **network_address** so that the retry and time-out period could be reduced. The new operation allows the RPC mechanism to use arbitrary time-out and retry parameters. A time-out can now be set which is suitable for the ISLAND application while still taking advantage of the mini-probe mechanism. Thus nested RPCs which take a long time to execute will not cause an inappropriate time-out signal.

7.5 Checkpointing

If a distributed service of the type described in ISLAND is to back-up state amongst its own components, two issues need to be considered. Firstly, the degree of recovery required from a server failure must be assessed in order to decide the content of state checkpoints. Secondly, a suitable distribution mechanism for checkpoints must be found.

7.5.1 Classification of System State

Four types of state have been identified to be characteristic of data in the *Exchange*. The following classifications have been used.

Transient state is the information necessary to hop between menu options in preparation to selecting a particular feature. Short term state is anything relating to the state of an association. Profile data has been classified as medium term state because it contains information which lasts longer than a *session*. Finally, long term state has been defined as data held in the directory; this is changed relatively infrequently in comparison to profile data.

ISLAND wanted to create a service in which long, medium, and short term state were recoverable, but transient state was vulnerable. This distinction was chosen to provide a trade off between usability and performance. If recovery was to be totally transparent, the time spent checkpointing data for a large population of clients would be detrimental to the service's performance. Furthermore, in this event recovery time would also be considerably increased. For a PABX application, it seemed reasonable to tailor the back-up and recovery mechanism to ensure that all current associations and profile updates were maintained. The position a client had reached in the feature menu may be returned to a different, but well defined state.

Changes in directory data are made on a longer term basis. Directory updates are never made at a *Ringphone*, instead they are made by updating a single central file and then distributing this information to the replicated components. If the *File Server* is not operational for a period of time, it is only a minor inconvenience that one or two entries may be out of date in the version contained in the *Exchange*. Long term state is therefore not an issue that has to be considered here.

It can be concluded that only a relatively small amount of state information needs

to be distributed between the components of the *Exchange*. The following sections describe the way this state may be represented and the techniques by which it can be passed between machines.

7.5.2 Distribution Options

The methods available for state distribution are the same as those considered for providing a consistent *view* of the operational members of a replicated set. The choice amounts to either a Multicast or a Virtual Ring distribution scheme. As described before, the Virtual Ring has the advantage that it distributes the workload evenly among the component servers and automatically re-routes data in the event of a server failure. A Virtual Ring protocol was therefore chosen for the distribution of state checkpoints. Chapter 9 describes the effect this choice has on the service delays for the system.

7.5.3 Associator State Checkpoints

At any point in time, only one server will contain a **phone** object which is actively representing a particular *Ringphone*. All the other **phone** objects representing that *Ringphone* will be in the 'idle' CFSM state. The active objects, however, will be distributed evenly amongst all the servers as a result of a randomly spread load. A convenient way to make a state checkpoint involves taking state from an active object and attaching it to all the idle replicated versions.

Checkpoints for active associations may be made by copying an association object and then distributing it to the other servers. A procedure almost identical to the **validate_ring** procedure (described in 7.4.1), can also be used for checkpoint distribution. Associators can then be deposited in the replicated **phone** and **service** objects in which they were held in the original server. This process amounts to placing the associators in the equivalent internal lists in which their host object originally held them. If the *event vectors* at the relevant *Ringphones* were to be pointed at a server containing an idle object it would now continue to handle events as if nothing had really changed.

An identifier can be used to allow associations to be overwritten when the corresponding association has changed its internal state. This process has the additional effect that updates are totally idempotent. If a checkpoint is repeated as the result of a server failure during state distribution, the effect is harmless.

Checkpoints should be created whenever there is a change in the state of an association. The distribution process can be made to have a minimal effect on the state transition time of a **phone** object by forking a process to carry it out. In the event of failure, that is to say the *view* of the logical ring observed in the checkpoint is not that of the current *view*, a **validate_ring** procedure can be initiated to correct the current *view*. The checkpoint can then be re-tried after a consistent state has been recovered, and the property of idempotence ensures this will not cause an

invalid state.

At the end of a *session* the **handset on-hook** event is normally used to cancel all associations which are in the current association list of an active object. This event can also be distributed to remove checkpoints at all the replication sites.

7.5.4 Profile State Distribution

Profile objects are replicated in all the exchange components. A profile update is derived from a client file. It contains state which lasts longer than a single *Ringphone session.* A profile can also be modified at a *Ringphone.* For instance, when logging-on to a *Ringphone* a number to station association is established and this data is stored in the corresponding profile object. A profile update must be distributed to the other servers if they are to be made aware of this new state. Therefore a profile object must be copied to all members of the replicated set if it has been modified. The distribution of state may be achieved in a similar way to that described for associators.

7.6 State Recovery

In the event of a failed *Exchange* server, the remaining operational servers must recover its state. The recovery procedure must also cope with the following issues.

- Failure of a server while already recovering state in another
- Failure of a server that is recovering state
- Additions of new servers to boost resilience

In addition to adopting a new *view* created by an activation of the **validate_ring** procedure it is necessary to execute a state recovery procedure. The following skeleton description of a procedure is suitable for recovering state in the ISLAND demonstration system. The concept of a *view* in this context has been extended to include the 'Age' parameter.

```
RECOVER_STATE =
  PROC( new_view, old_view: array[record[name: string, age: int]])

(1) Calculate difference in the new and old views:
     Difference := Difference_Of(old_view, new_view)
(2) Calculate oldest member:  Oldest := Oldest_Of(new_view)
(3) FOR all names D in Members_Of( Difference ) DO
  (3.2) IF Own_Name ~= Oldest
        THEN Change all Ringphones with the Controller
             named D to Oldest and RETURN
        ELSE Remove D form the table of event vectors
             at every Ringphone.
  (3.3) FOR all phone objects P with a Controller = Oldest DO
     (3.3.1) IF Ringphone P's state disagrees with its checkpoints
               THEN validate the checkpoint state
     (3.3.2) Put the phone P into  Call State (CFSM) of MENU_TOP
     (3.3.3) Change P to have a Controller = Oldest
     (3.3.4) Change Ringphone P to have a Controller = Oldest
(4) LET old_view := new_view
END recover_state.
```

This procedure is designed to be executed at every server as the recursive **validate_ring** routine returns by way of the servers in the Virtual Ring. The essence of the algorithm is that one server will take over the state of the failed server and that the others will recognise this server as the new controller of the recovered state. It operates as follows:

The servers making up the distributed application are considered to be part of a set. Each member of the set has an 'Age' which is represented by a simple integer

to define the order in which servers joined the set. Servers may only join the set one at a time.

The oldest remaining server is calculated. The server executing the routine then checks to see if it is in fact the oldest server. If it isn't then it must tag all **phone** objects that were controlled by the failed server to now be controlled by the oldest server. In this case it must now return.

However, if it is the oldest, the failed server's name is removed from the table of *event vectors* contained in every *Ringphone*. This operation can be performed through the SSP interface, and will prevent any new *sessions* trying to contact the failed server. Every **phone** object in the oldest server that was controlled by the failed server must now confirm that its checkpoints agree with the voice association established at the real *Ringphone*. It must correct any that are not in agreement with it. These **phone** objects must now be placed in a standard call state, because 'transient' state cannot be recovered. The top level of the feature menu was thought to be a suitable state for this purpose. Finally, the tag indicating the controller of the recovered **phone** objects must be set equal to the oldest server's name, and the corresponding *event vector* of the related *Ringphones* must also be changed to this name. At this point any events which these devices had accumulated during the recovery process as the result of clients impatiently pressing keys, can now be handled by the oldest server. The oldest server has a correct model of the association state and can continue processing these events in the normal way.

It is essential to use the oldest server for the recovery action to ensure that the maximum amount of checkpointed state is available. A newly-added server will contain a relatively small amount of state information. However, association state is thrown away at the end of a session, nominally after 180 seconds (see chapter 9). Therefore a new server will acquire a similar cache of state to the other members of the set in a short period of time. Alternatively, association state could be passed in its totality when a new server joins the set. Because it is unlikely that servers will fail at a rate which would make this worthwhile, the existing servers should be spared the overhead of this transfer.

The **recover_state** procedure must be encapsulated in a monitored region to ensure the serialisation of recovery tasks when other servers are executing a recovery. The algorithm will cope with simultaneous failures of servers because the recovery operation is performed on all the servers indicated by the difference between the old *view* and the new *view*. If a server fails during a recovery in which it is not the oldest member, a recovery procedure would be serialised after the current one to handle the new failure. If the oldest server, say **a**, were to fail while it was performing a recovery, the failure would be detected by the **kicker** process responsible for monitoring it, and a recovery by the next oldest server **b** would begin. However, the other servers will have changed the name of the controller of all objects that were already being recovered by **a** to have the tag of the server **a**. The server **b** that must recover the two outstanding failures will now recover both sets of objects by simply recovering those marked **a**. The requirements for

handling failure during recovery have therefore been satisfied.

7.6.1 Replacing Failed Servers

So far a mechanism for adding new servers to a replicated application has not been considered. There were two approaches to the problem. In the event of failures new servers could be automatically allocated to replace them, or else they could be replaced through human intervention; the fault tolerance mechanisms of the remaining set of servers will keep the application running. The failure modes that could occur when automatically adding a new server were found to be complex and could be the subject of further investigation. Chapter 8 describes a management service which can be used by an operator to allocate new servers to a distributed application, and manipulate their existing *views* to include it amongst them. The service can also be used to upgrade the fault tolerance of a correctly operating distributed application. An important part of its design is that it has not created a single-point dependency for the continued operation of the application.

7.7 Miscellaneous Reasons for Recovery

State recovery may also be triggered by the memory management-processes of a run-time system. One example is *Garbage Collection.* CLU is a language in which heap memory is dynamically allocated for the creation of objects. Mayflower recovers un-referenced memory from the heap by the use of a marking garbage collector which is run synchronously with the application. The result is an occasional 'dead' period of time for an application while the garbage collector is performing its operation. If a very large heap is allocated at link time, garbage collections will be infrequent, but when they occur they will last for a long time. However, if a heap which is just large enough to store a handful of new data structures is used, garbage collections will be frequent but only occupy a short time. The latter example must be used in the case of a server that must be highly available. The mechanism for fault detection requires response times that are within a critical time-out period; long garbage collections will cause a **validate_ring** procedure. One approach is to ensure that all servers in the set have a random but large heap size, and therefore will definitely be removed from the *view* while they are garbage collecting. Different heap sizes should imply that individual servers are affected at different times. The heaps should be chosen so that there is a sufficiently large amount of time to recover totally from one failure before another begins. If the heap is very large it may actually be faster to use the resilience mechanisms to recover state than waiting for the garbage collector to do its job. Of course all these problems could be solved if the development system were to support an asynchronous garbage collector.

It is perhaps a good philosophy to cause artificial failures on a regular basis. Recovery code is normally executed considerably less frequently than the code in

everyday use of a system. Implementation errors are far more likely to be spotted during a trial period if the recovery code is regularly tested. Many models have been made to try and predict the diminishing rate of failures due to the correction of these errors. An example is the Musa model [Musa 75] which assumes that logical faults in software occur as independent random processes, and thus have a Poisson distribution with respect to time. In this model the faults are always successfully repaired and every fault has an equal effect on the reliability of the program. The model is shown to predict that the rate of fault correction falls off with a negative exponential function. Such models are generally too idealistic and have limited application for making useful predictions.

7.8 Summary

In this chapter a mechanism has been proposed for a self-monitoring distributed service. The method has been described in the context of ISLAND's distributed *Exchange* service. The development of a convenient method for distributing data, the Virtual Ring protocol, has provided a way in which all servers making up a distributed set can agree on the membership. This method can also be extended to the checkpointing of state information between servers.

State in the *Exchange* which needs to be protected has been classified as either association state, or state held in a profile object. These objects are relatively small and can be passed between servers efficiently. Checkpoint copies of associations can be attached to the idle replicas of the objects that originally created them. When a server fails, recovery of previously active objects state by another server amounts to little more than informing its idle versions of those objects that they are now the active version. It is also necessary to tell the relevant *Ringphones* to send their events to the server now responsible for them.

A detailed comparison of the Multicast and Virtual Ring protocol is found in chapter 9 along with an evaluation of the effect they have on the service delays of the *Exchange*.

Chapter 8

Management Coordination

This chapter discusses the problems associated with managing a distributed program, coordinating its component parts, and providing a suitable interface for an operator to carry out routine checks and statistics on the distributed task. Early experience of this kind of problem is found in John Shoch's work at Xerox PARC [Shoch 82].

8.1 Creating a Distributed Program

So far it has been assumed that a program can be compiled, linked and loaded into many servers as a matter of course. Furthermore, each server has been assumed to have a working knowledge of the members composing the replicated set. This type of distributed program requires a tool or a specialised service to provide a mechanism for instantiating the component parts. This was realised early on in development and a service was created with the following properties:

- The ability to load a program into an arbitrary number of machines forming a *machine set*
- The ability to manage the resilience of a set dynamically.
- Provision of a central point to log component failures
- Provision of a mechanism for managing several *sets*
- Provision of an interface for managing the distributed task —in this case PABX control
- *Machine sets* should not be reliant on the creating service's operation

This service functioned as a central control point and was referred to as the *Master.* Its implementation was based on an interface with the *Resource Manager* (RM)

[Craft 85]. RM is normally accessed through the *Session Manager* that allows a byte stream to access the RM function at a remote terminal. RM also offers a network SSP interface which takes a string argument and interprets it as if the characters had been typed over the byte stream. RM presides over an abstraction which can be considered as a 'pool' of free machines (the *processor bank*). On request for a system, a machine is chosen which is best suited for running it and then the system is loaded into that machine by an *Ancilla*. The service making the resource request also passes a token of authentication to RM, allowing the allocated machine to set up its own byte stream to the terminal. Systems will usually present some sort of command line interface across this connection. The *Master* is initially established in a *processor bank* machine using the same mechanism(see figure 8.1). The *Master* provides the operation **create** with the following semantics:

```
CREATE  <k-resilience: int>  <load_file: string>
```

The *Master* then makes a request for a machine loaded with the Mayflower operating system **k+1** times, passing its own authentication token so that each machine can establish a connection with the same terminal as the *Master*[1]. Once **k+1** machines have been allocated, the *Master* uses the Mayflower RPC mechanism (accessible at each machine) to load the **load_file** given as the second argument.

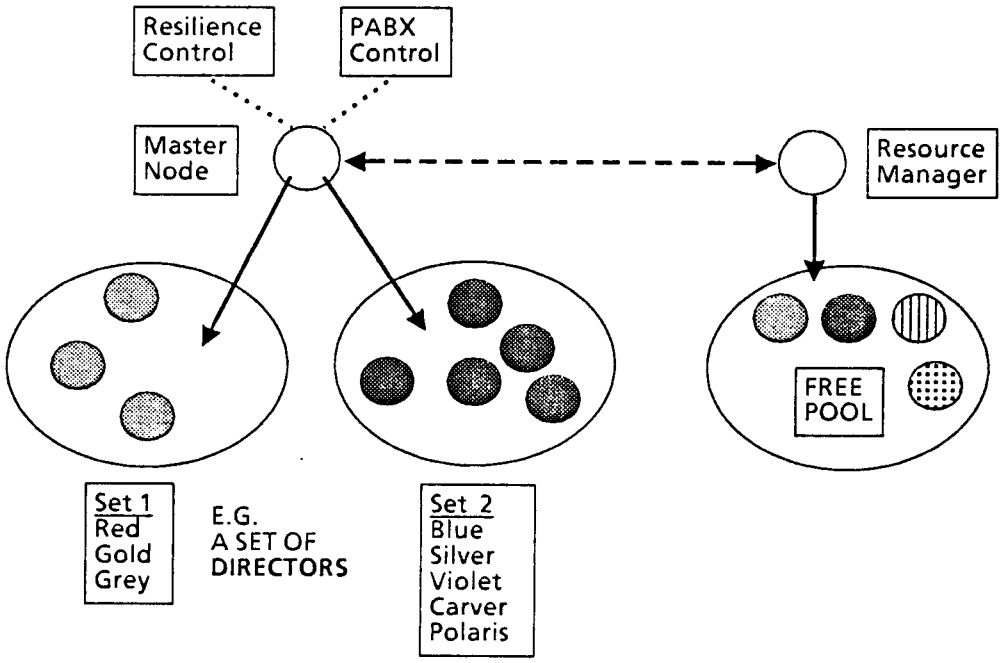

Figure 8.1: The abstractions used by the *Master* node

On completion, the names of the set are entered into a data abstraction called a *machine set*, of which there may be an arbitrary number. A remote initialisation

[1]In practice this option was only necessary while the system was being developed and debugged. A further parameter could also be set up to prevent reverse connection.

procedure called by the *Master* at each machine, gives each set member knowledge of its fellow members, the **load_file**, and the name of the machine currently running the *Master*. At this point, the program in each instance of the *machine set* will begin execution. However, the mechanism by which the checkpointing and recovery mechanism operates and the details of the application program defined in the **load_file**, are not part of the *Master's* operation.

8.2 Machine Sets

The *Master* became an important tool for distributed program development. Further enhancement allowed multiple sets of machines to be created in association with integer handles, enabling an operator at the *Master* to selectively control any particular set. This kind of functionality allowed a service *Exchange* to be executing in one set, while developing the next incremental version in another.

A number of operations were found to be a necessary part of the *Master* Command Line Interface (CLI). These operations are summarised below:

- CREATE <k-resilience: int> <load-file: string>
  Creates a machine set with an integer handle
- CHANGE <new-resilience: int>
  Increases or decreases the k-resilience of the current set
- MEMBERS
  Lists the names of machines in the current set
- KILL_MACHINE <name>, KILL_SET
  Renders the specified machine(s) inoperative (kill is a useful command for testing the fail-stop resilience mechanisms of an application)
- STATE <name>
  Returns a list of machines, a **load_file**, and the *Master* name which a specified machine believes to represent its current state (such an operation can be used to check that a machine has a consistent *view* of its own set)
- VALIDATE
  Corrects the view of the current set, so that each component has a view that agrees with the *Master's* model

Further operations are required on multiple sets and are also listed below:

- LIST
  Lists all *machine sets*, their handles, and their contents
- KILL_ALL
  Performs the same function as **kill**, but works on all machines in all sets

- MERGE <set1: int> <set2: int>
  Joins two sets together giving it the 'handle' of the first, **merge** will automatically perform a validation operation. This tool is available to the operator should it become apparent that a *machine set* has become partitioned due to a network break, or network congestion.

- LOCATE
  The master node is not replicated and is not relied upon for its operation, it is merely a device for communicating with a distributed program. If the *Master* crashes, its state will be lost. When an operator reinstates the *Master*, a **locate** operation can be invoked to send investigation RPC's to all machines known to exist in the *processor bank*. If a reply is obtained, the data contained in the returned result enables a model of the *machine sets* still in existence to be reconstructed.

- LOG < operation(+, -, @, >, <): string> , <level: int>
  A system operator should be informed of erroneous system states as a matter of course. Log messages are incorporated in key positions in the *Exchange* software, and interface to the *Master* through an RPC interface. Messages are tagged with a priority level from 1 to 10. The levels to be reported are controlled by a filter. A convention of larger numbers for more important events is used. The filter is set up remotely in the **log** command and prevents unnecessary RPCs being sent. The filter is represented as a set of values. **Log** levels can be added or removed by using a '+' or '-' operation respectively. Logging can be set to an absolute level with the @ operation, or a range of levels using '<' or '>'. For example, '< 7', means log all messages in the range from 1 to 6. Logger messages are reported at the *Master's* terminal with a machine name and log level prefix. Messages are further suppressed if they do not originate from the currently selected *machine set.* This scheme has proved an effective way of determining the condition of remote machines without being swamped by an unmanageable quantity of debugging information.

8.3 Command Line Interface

The operations described have been implemented as a Command Line Interface (CLI) at the *Master's* terminal session. The executive program, a part of the Mayflower libraries [Craft 85], can switch in a number of different CLIs in addition to a global set of commands available from all of them. This arrangement allows operations which have a logically similar function to be grouped together and prevents a client operator having to sift through numerous options when deciding upon an action. Three CLIs were identified as useful sets of commands. **MASTER** as already described, **SM** as an alternative interface to the *Resource Manager*, and **PABX** as an operational interface to manage and coordinate the PABX operations.

8.3.1 The SM CLI

The SM CLI provides a set of enhanced RM commands. The need for a better interface was the primary motivation for implementing SM. A secondary benefit is that it saves the need to create another byte stream to the terminal, and the need to switch between Mayflower and it using the *Terminal Concentrator* commands.

SM commands include:

```
LIST        <status>   <attribute>   <owner>
WITHDRAW    <machine name>
DEPOSIT     <machine name>
FREE        <machine name>
```

The list command gives a condensed view of the state of processors known about by RM. Attributes include the size of the machine in kilobytes and the processor type (68000, LSI4, 68020). The *status* field can be set to one of the following: *free*, *allocated* or *withdrawn* (a state used for testing). The *owner* field specifies the name of the machine that the resource is allocated to. A '*' parameter is interpreted as a wild-card match: an unspecified field will default to '*'.

The remaining three operations change the status of a machine. The **Free** operation should be used with care otherwise it may result in a client losing a session.

8.3.2 The PABX CLI

The *Master* CLI is a generalised tool for the creation and manipulation of a replicated distributed program. For the purpose of interacting with the *Exchange* it is necessary to have a specialised CLI providing an interface with specific PABX functions. There are two benefits from putting the PABX interface into the *Master* machine, rather than into each of the components of a set.

Firstly, all operations updating state in the PABX, for instance when instantiating a new directory, must be repeated at each component of the set. These operations can be easily automated by code in the *Master*, which has the complete knowledge of the set.

Secondly, user-interface code is verbose, it must cope with all erroneous cases and prompt for missing information. The result is usually a code module of some considerable size which would have to be replicated throughout a *machine set.* The *Master* is a place where this code can reside uniquely and communicate with the *machine set* through a relatively naive RPC interface.

8.3.3 PABX Control Operations

The following operations have been supplied in the CLI interface.

- READ_DIR <file: string>
  A command causing each *Director* to read the directory file specified and to instantiate a new directory object

- REINCARNATE <phone: string>
  Recreates a phone process and its guardian process in the event that an error has caused either to fail

- ACCOUNT <file: string>
  Turns the accounting mode on and logs data in the specified file

- STATISTICS
  Returns the total number of associations that are currently active and the number that have ever been active

- ASSOCIATIONS <phone: string>
  Lists all associations that the named *Ringphone* has attached to it, along with the state of that association

- DEBUG <flag: bool>
  A debug state can be set up in which all log information is sent to the local byte stream, if one is present.

8.4 Alternative Distributed Management

The method described is a simple approach to the problem and was implemented successfully. It is worth mentioning an alternative strategy which was under consideration when the *Master* was designed.

If the mechanism for requesting a machine from the 'free pool' is instantiated in every loadable program, it then becomes possible to write a recursive program to generate a set with an arbitrary resilience, by a single **create** call. A **create** operation would take the same form as before, but a single machine request would be converted into a recursive call, each stage adding a machine to the set while reducing the passed parameter **k** by one (the termination condition being **k** equal to zero). Such a scheme is more complicated to code and, in particular, the error conditions are more difficult to handle than centralised allocation. Moreover, there still needs to be an equivalent of a *Master* to provide an operator interface. The recursive scheme is therefore far less attractive than the centralised one.

8.5 Implementation: Problems and Solutions

A notable implementation problem is the time it takes to allocate and load a free machine with an operating system, for example, 15-20 seconds must be allowed for the Mayflower Kernel. If it is found that the k-resilience of a machine set has

fallen below some critical value and needs to be restored, there will be some delay before a machine can be brought into place. If the k-resilience is initially made far higher than need be, this is not a problem, but servers may be scarce and needed by other system components, or clients, at peak times. A potential solution is to fill the 'free pool' with a number of dormant standbys, adding this property to the attribute list and thus these machines will be allocated in preference to a non-preloaded server. There is, however, a danger that the dormant server will experience some undetected transient memory-fault, making it inoperative when it attempts to execute a program. However, this would be detectable at the initialisation phase, and only in the event of this rarer error would the full loading delay be experienced. This scheme requires modification of the RM service and was not carried out.

A second problem was relatively unexpected and needed special software support. Some free pool servers could be loaded with an operating system but would fail to load the application program. If the machines were released and a new RM request made, they would still be the most suitable machines free, and would be reallocated demonstrating the same problem. An abstraction termed the '*Hate* pool' was added to the *Master's* operation and used to deposit machines which failed in this way. In practice, the pool was initialised with servers which were known to fail in this way and, by making a comparison at allocation time, were disposed of during the create operation. Operations were added to the *Master* CLI to add, remove and list members of the *Hate* pool, allowing an operator to find out which machines should be considered for repair or replacement.

8.6 Summary

A centralised tool for creating a distributed program was essential for experimentation with the checkpointing and recovery mechanisms in the ISLAND distributed PABX. When designing this kind of tool it is necessary to ensure that the distributed application is not dependent on the tool's operation, otherwise the system will be dependent on a single machine to guarantee its own availability. The *Master* server has also allowed a sophisticated operator interface to reside in a single machine, and thus to reduce the complexity of code within a replicated *machine set.*

Chapter 9

Evaluation

This chapter evaluates whether the ISLAND demonstration system will scale. Measurements of delays experienced by a single telephone are scaled in a simulation model in order to predict the service delays expected in a larger system. The same model is used to make a comparison between a Multicast protocol (implemented on a Cambridge Ring as a set of separate messages) and a Virtual Ring protocol, as a checkpointing mechanism, to ascertain their relative merits.

9.1 Telephone Usage Patterns

Studies carried out by telephone companies as long ago as the 1920's determined that a large population of subscribers generates calls in a random and independent fashion and also that the duration of a phone call is governed by a random process.

The effect of these processes are that the number of calls in progress over a period of time has a Poisson distribution. This gives rise to a useful mathematical model that can be used to make predictions about the system. A property of a Poisson source of telephone subscriber traffic is that the inter-call arrival times have a negative exponential distribution.

Queuing theory has been applied to the analysis of these systems, a detailed study of which is found in work by L. Kleinrock [Kleinrock 75], a more specific application of the techniques to telecommunications may be found in work by D. Bear [Bear 76].

The theory analyses systems that can be represented by a source of traffic, a queue, and a server. The source puts work into the queue and the server takes work out. Usually FIFO queuing disciplines are considered. The mean number of items in the queue and the mean service delay are typical predictions of the theory. An alternative approach to obtain these results is to use an event simulation model. A computer can be used to model the load functions and the data processing-rate of servers in a relatively condensed period of time.

Event → / Time ↓	Caller Off-Hook	Key Press (4 Events)	Remote Ringing	Remote Off-Hook	On-Hook (2 Events)
20ms Ticks	11	7 x 4	54	39	26 x 2
ms	220	140 x 4	1080	780	520 x 2

Table 9.1: Time taken by CFSM events.

In a conventional telephone system the theory can be used to determine the utilisation of a switch. Another useful prediction is the probability that a call will have to wait for a line before it can be connected. Utilisation is given the unit of the Erlang (named after the Danish mathematician who made major contributions to the subject of Queuing theory). Measurements of utilisation are quoted for the busiest hour of the day, usually about 3pm. Typical public telephones have a utilisation of 0.1 Erlangs. An office telephone has a higher usage and is usually considered to be between 0.15 and 0.2 Erlangs; 0.15 Erlangs is the figure used in the following sections. The mean call holding time is usually considered to be 180 seconds for the purpose of simulation and analysis.

The utilisation (ρ) can be defined in terms of the mean call arrival-rate (λ) and the mean holding-time ($\bar{x}$).

$$\rho = \lambda\bar{x} \qquad where \qquad \lambda^{-1} < \bar{x}$$

This assumes that an equilibrium exists between the arrival rate of work and the rate of data processing at the server. If the condition shown on the right does not hold the queue will be unstable and may grow to an infinite length.

9.2 Simulating the ISLAND System

In the ISLAND system the *Exchange* service is concerned with the call processing period of an association (a small percentage of the total call holding-time). Even though the holding time has a Poisson distribution the processing time of telephone events is deterministic. Furthermore, there is a high probability that the number of events associated with each call lies close to an expected value. However, this does assume that special features are hardly ever used. For example, a typical call may consist of the following events: handset off-hook, 5 key events, a second handset off-hook (for an answered call), and two handset on-hook events at the end of the call. The timings measured for a single telephone call can be used to constrain a model which simulates the arrival of many call events at the exchange using a queuing model.

The ISLAND demonstration system (with only 9 extensions) was used to measure these deterministic delays with a view to measuring the delay of service with up to 500 extensions. Table 9.1 shows the delays experienced for the various state transitions of a call determined by inserting monitoring code into a non-replicated

Server(s)	1	2	3	4	5
Max Ext.	90	190	265	365	490

Table 9.2: Maximum number of extensions before service is poor.

version of the exchange server. The processing time for events is measured to the nearest 20ms. The results shown are for a single *Ringphone* initiating a call with the *Exchange*, and are averaged over 10 trials conforming to the simple model of call set up.

To make a simple model of a call the average of these 9 events were taken to give the average time to process a telephone event. 409ms was calculated as the mean holding time for an event.

A batch of key presses were made on a push button telephone, in a manner which reflects normal dialling, this determined the mean separation of key events to be 500ms. The call model consisted of 7 events that were all 400ms long, each in a period of 500ms, and two others at an exponentially distributed period of time later with a mean of 180 seconds. Note that the initial model is for a single server with no checkpointing or monitoring events to be considered.

However the simulation model was taken further to consider busy extensions. Calls in progress are added to a *busy* set. The creation of a new call must come from an extension outside this set. A called party is chosen randomly and if it intersects with the *busy* set, the telephone call will terminate immediately.

A Poisson source can be simulated by using a random-number generator with a rectangular distribution and biasing the values returned with a logarithmic function. To measure the mean service delays the sample time must be long enough to smooth out the fluctuations that result from stochastic processes and have a lead time sufficiently large to allow the queuing model to reach a statistical equilibrium. This time period was determined experimentally by continuously averaging the time that a call event spent in the queue, and examining the relative values of successive means. A lead time of 10^7ms was used and an elapsed time of 10^8ms provided the termination condition. The results shown in figure 9.1 are the mean delays experienced by clients expecting service from a server with a population of between 0, 10, 50, 100, 200 and 500 extensions. Note the results are presented as smooth curves which have been derived from an interpolation (cubic) of 6 points. The intermediate points are not intended to be more accurate that 5% as the simulation is only a guide to the scaling factor (the simulation queue lengths were only ever stable to 5% of their measured size). The results given from 0-50 extensions are less accurate than this due to the small sample having an adverse effect on the distribution of engaged calls. Similar restrictions hold for figures 9.2 9.3 and 9.6.

In order to assess the quality of service, one second or more is defined to be an unacceptable service delay. If the service delay is under one second, it is considered to be acceptable. Using this criterion figure 9.1 implies that the maximum number of extensions supportable by one server is about 90. A point to note about the

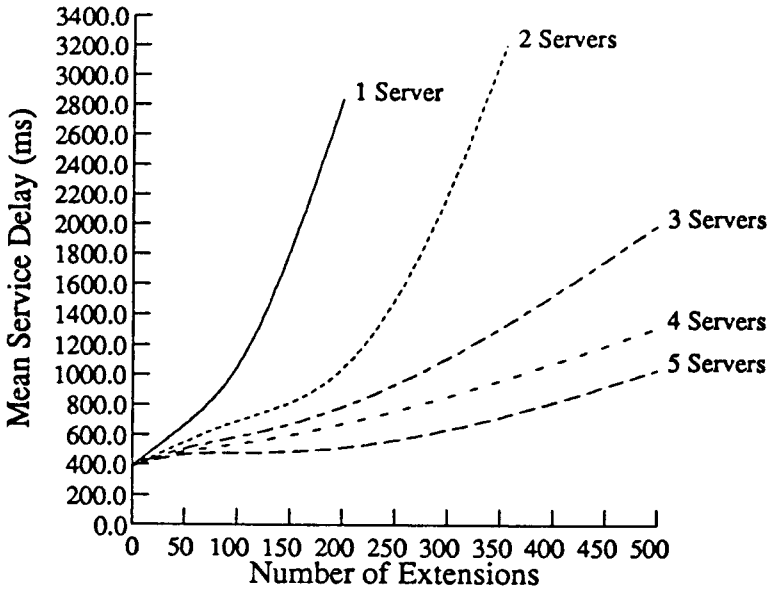

Figure 9.1: Service delays experienced with a variable number of servers

result is the rapidly increasing gradient of the graph, indicating an infinite delay at about 250 extensions. This is because the increasing number of extensions implies a decreasing mean inter-arrival period which approaches the processing time of an event. For 220 extensions the two quantities are almost equal and thus the utilisation approaches unity, the condition for instability.

The model can be extended to simulate the multiple server scheme adopted by ISLAND. When the simulation creates a call, a further uniformly random choice is made to select a server out of a defined server pool. Simulations were carried out for 1 to 5 servers, and these results are also illustrated in figure 9.1.

As expected, a greater number of servers reduces the service delay for a given number of extensions and thus more extensions can be added while maintaining low service delays. If the 'one second' rule of thumb is used, a maximum value for the number of extensions in each case can be read from the graph, and is shown in table 9.2.

As yet no consideration has been taken of the resilience mechanisms put into play for more than one server, namely monitoring and checkpointing. Such mechanisms can be modelled as events which are added to the server queues in the same way as telephone events.

9.3 Fault Monitoring

Inter-server monitoring is carried out by executing a remote procedure call (RPC) at regular periods of time. The delay experienced by a CLU RPC has been documented in G. Hamilton's original work [Hamilton 84]. Many changes have been

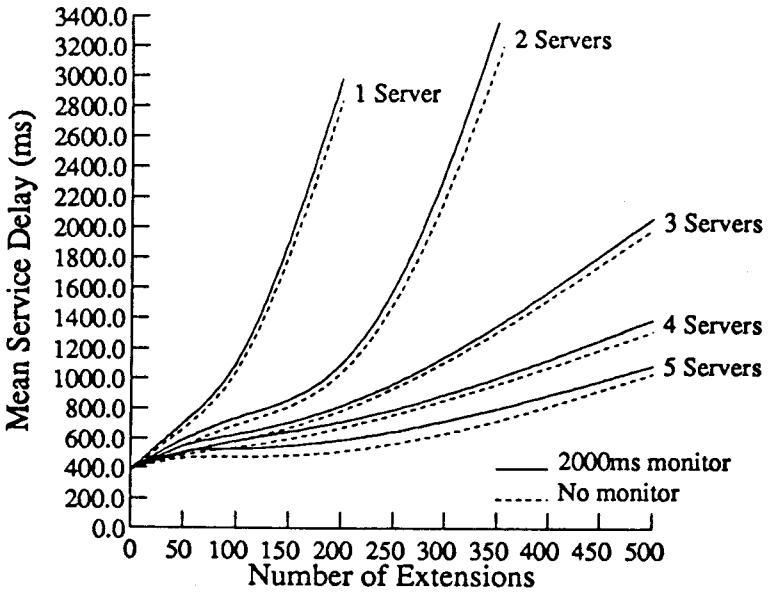

Figure 9.2: Service delays when using a 2000ms monitor probe

made since then, but it does allow a maximum mean delay to be estimated. For the benefit of a simple model, an RPC is considered as a message pair with each transmit and receive component taking 15ms of processor time. This value is certainly an upper bound on the time taken to process the kind of messages/data sent in the demonstration system. The processing time for a complete message is 2x15 = 30ms. In practice this is an overestimate —the round trip time for a null RPC may be as low as 12ms. All inferred results from the simulation are therefore conservative estimates.

Figure 9.2 repeats the results in figure 9.1 with additional information showing the effect of a 2000ms monitor-probe on the service delay. The mean delay in the server queue has only been calculated for telephone events because it is this aspect of service delay which defines whether or not its usage is tolerable.

The results show that the delays are not significantly increased. It is therefore interesting to determine the maximum rate of monitoring that can be carried out without significantly degrading the system performance.

Figure 9.3 shows the effect of monitoring with a period ranging from 100ms to 5000ms, for a 200 extension system with up to 5 servers. The monitor only has an effect when its period approaches the event holding-time. The graph shows that below a 750ms monitor period the delay is no longer constant but begins a steep upward-trend and thus adversely affects the event handling time.

Another factor which determines the minimum sensible time for the monitoring period is the time taken to recover from a server crash. If this is longer than the monitor period, a fast monitor will not be very effective. In the demonstration system recovery time is deterministic and proportional to the number of active

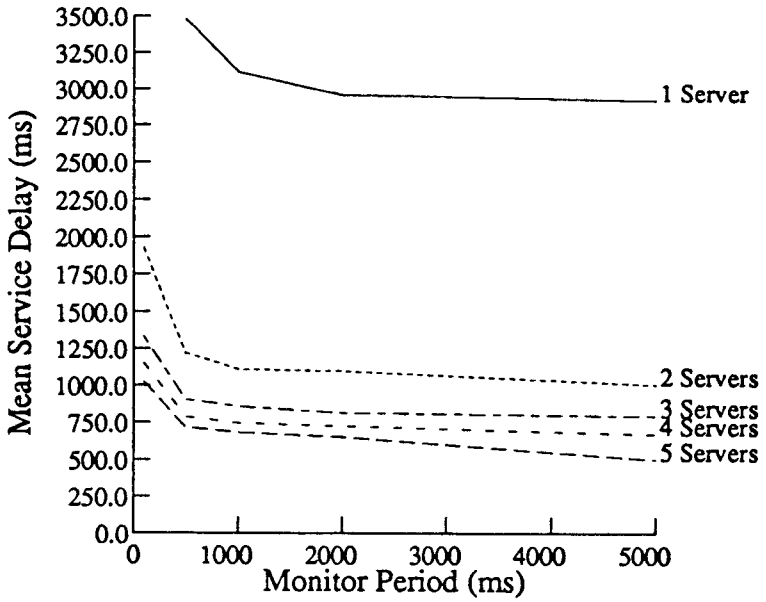

Figure 9.3: Service delay compared to monitor period

extensions, but inversely proportional to the number of servers. The mean time to take over the data structures associated with a call has been measured to have a lower bound of approximately 50ms. This value is taken from an experimental version of the recovery procedure. Given 200 extensions and 2 servers we can estimate that the monitor period should not be less than (200/2) x 50ms = 5 seconds. This is already considerably longer than the 750ms period derived from the simulation model.

In figure 9.4, the monitor time is defined in terms of the recovery time. The line labelled 'Min value' at 750ms indicates the lower bound based on performance loss. It can be seen that the limitations imposed by the recovery time is, in general, the dominating factor.

9.4 State Distribution

Two alternative schemes for checkpointing state were considered for the implementation of the ISLAND system: Multicast and Virtual Ring distribution. To determine which has the least impact on service delay, two separate simulations were carried out for these protocols.

For the degree of recovery expected, checkpoints are only necessary when there is a change in association state between two callers. In the simplified Call Finite State Machine (CFSM) used for the simulation model there are four state changes: (1) Establish (2) Connect (3) Delete(caller) (4) Delete(remote), and thus four checkpoints are made. (1) and (2) when establishing the call, and (3) and (4)

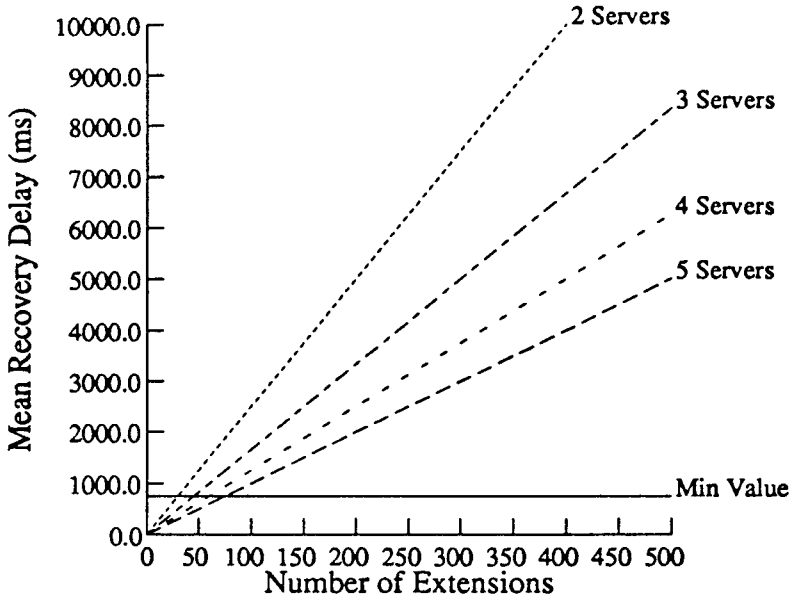

Figure 9.4: Monitor periods on the basis of recovery time

when each of the two parties close down the call. Multicast involves the server, which is executing the state transition, sending a checkpoint to every other server in the exchange. The Virtual Ring protocol causes the controller to send a single RPC to another server which then propagates it recursively on to the next server in a logical ring and so on until every server has been visited by the call. For the purposes of distribution of state, the termination condition is that the next RPC will be made to the originator of the checkpoint. The two methods generate the same number of message pairs during a back-up process. The only difference between them is the distribution of time spent by particular servers carrying out the checkpoints. The relative distribution of load for up to five servers is shown in figure 9.5.

Multicast and the Virtual Ring protocol have the same load distribution for two servers, and a similar one for three (although the greatest load falls upon the next server in the ring after the checkpoint initiator in the Virtual Ring scheme, rather than on the initiator, in the Multicast protocol). For more than three servers the distribution of load becomes linearly more biased against the checkpointing server in Multicast, whereas the Virtual Ring maintains a more even load over all servers. It is the server initiating a checkpoint which is most likely to be processing events at that time, and therefore to increase its work load during this period causes a reduction in performance. The Virtual Ring, however, distributes the load to servers which are more likely to be idle, thus reducing the processing delays. A simulation was carried out to find out if this reduction was significant for the system configurations used by ISLAND.

Figure 9.6 shows that the mean delays for a Virtual Ring are certainly less than with Multicast, but that the service delay is not significantly changed by either. However, there is a trend which becomes noticeable at high loads with 5 servers.

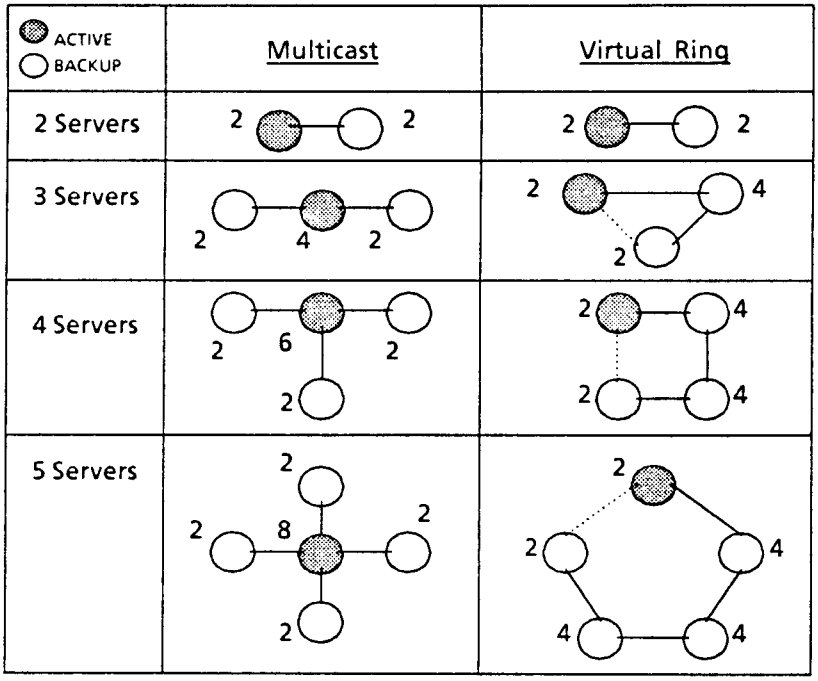

Figure 9.5: Message distribution in the Multicast and Virtual Ring protocols

As the number of servers increases the benefits of the Virtual Ring are more significant.

On the assumption that a Virtual Ring will have a linear advantage over Multicast when increasing the number of servers, figure 9.7 shows the extrapolated improvement of the Virtual Ring protocol over Multicast. A load of 500 extensions with between 2 and 5 servers is used to predict the improvement at up to 10 servers. The graph is expected to be linear because the work load on the controller has a linearly increasing overhead in Multicast, but in the Virtual Ring distribution-scheme the overhead is constant. The graph allows us to predict that a 10% improvement can be obtained with 6 servers and a 20% improvement with 11. This number may be considered an overkill in added redundancy in order to take advantage of the necessary gains in performance. The conclusion must therefore be that for the purpose of performance in the ISLAND system, it makes little difference which scheme is implemented. A further consideration is the ease of implementation and ease of error recovery. The novel recursive RPC did, however, turn out to be a very concise way of implementing a state distribution procedure, and if similar control systems are designed with larger numbers of control servers (i.e. greater than 10), it may be possible to benefit from this technique.

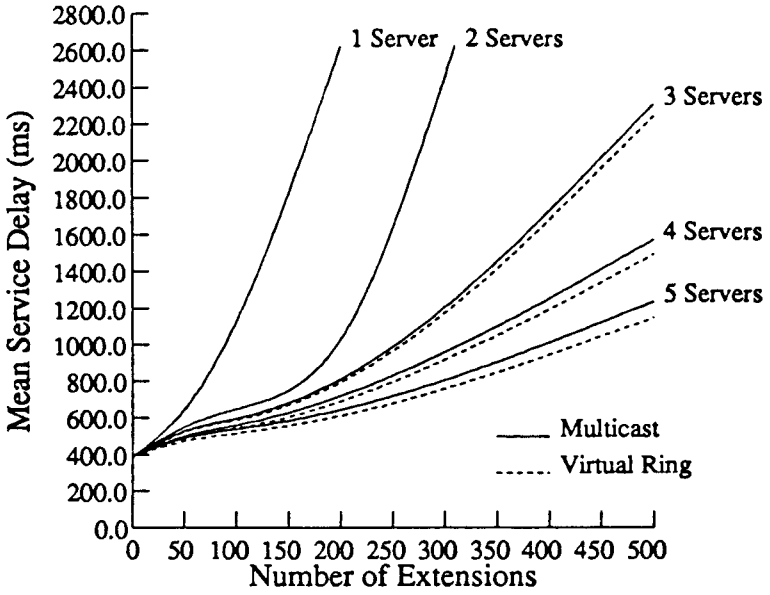

Figure 9.6: A Comparison of Multicast against a Virtual Ring protocol

9.5 Inaccuracies in the Queuing Model

Strictly speaking, a server running under the Mayflower Operating System does not process events in a FIFO queuing order. When events arrive they are handled by processes which in some cases queue work to be done, but in others simply wait on monitor locks or semaphores for the scheduler to activate them when the corresponding lock is released or the semaphore notified. The scheduling scheme is not necessarily a FIFO. Also a scheduler will perform time slicing between active processes. Mayflower currently uses 100ms per slice. Events which take longer than this time will be pre-empted in favour of another. At first sight this looks very detrimental to the model of telephone events taking 400ms. However, the current implementation assumes a global monitor lock for all phone objects and thus **phone** CFSM transitions cannot pre-empt each other. Further, checkpoints on **phone** objects also need to acquire the lock and are FIFO serialised. On the other hand, it is possible that incoming monitoring calls may be scheduled out of sequence and ahead of events already queued, making delays slightly longer than predicted by the model. Provided that the model is not being used to predict delays from the very heavily loaded sections of the graphical results, the model should still produce valid conclusions about this implementation.

9.6 Conclusion

A typical business PABX may be expected to support 200 extensions. An ISLAND PABX could not meet this demand using only a single server. The server used to

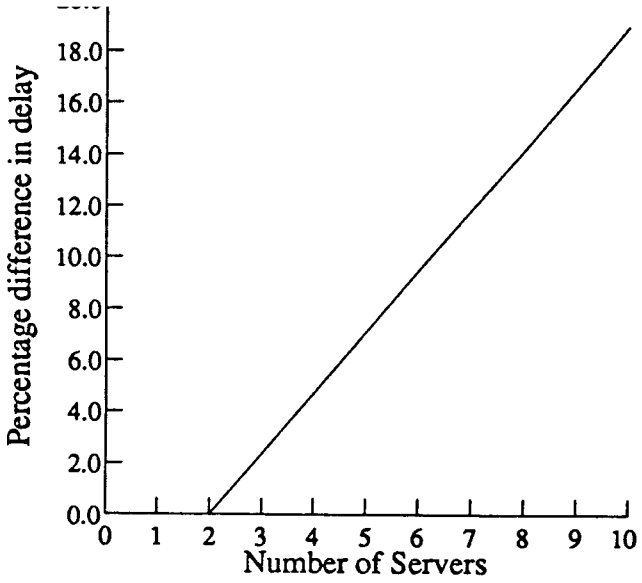

Figure 9.7: Projected benefit of Virtual Ring performance over Multicast

provide the timing measurements for this simulation was one of the original servers used in the CMDS processor-bank. It utilises an 8MHz 68000 with an intelligent network interface, the MACE. The communication path to the *Ringphones* was through two ring-ring bridges. Although the bridges are efficient they will certainly have increased the transit time of SSPs. In addition, no time has been spent optimising the event handling routines. It can be concluded that the timings could be considerably improved upon.

Two servers only just provide a reasonable quality of service, but three servers meet the delay requirements fairly easily. Three servers and 200 extensions give rise to a mean delay of about 800ms and a redundancy capable of surviving two complete server failures. Information about the performance of commercial systems is difficult to obtain. It is likely that such systems will respond faster in a closely coupled processing environment. However, if the delays for a distributed system are low enough, the service will be acceptable. It is worth remembering that the model only considers loading at the busy hour and that the response time will be much faster throughout the rest of the day. Also with three servers a single failure will still result in a tolerable system response.

Another factor affecting the validity of these simulations is that the simulation model has not represented the effects of a client interacting with the special features that are provided by the *Exchange*. It may be that the improved interface and remote *Ringphone*-control from a workstation will increase the usage of features and thus generate a greater load on the *Exchange*. In this case the assumed traffic patterns may be rendered invalid. This possibility can only be clarified by practical experience derived from a larger implementation.

Chapter 10

Conclusion

In recent years there have been a great many proposals for the implementation of integrated communication systems. Without direct experience of the benefits of an integrated system it is difficult to assess their usefulness. The ISLAND demonstration system has provided an environment in which voice and data have been integrated at the physical level. Additionally, the ISLAND distributed management scheme provides highly available control of network telephones while also allowing clients to write their own control application-programs. The mechanisms described for checkpointing and recovery of associations have, as yet, only been demonstrated in a limited way and have not been included in the current demonstration system. Apart from the physical benefits of using a single network for integrated media, it is an environment in which multimedia applications of the future can be designed and implemented.

10.1 Integration Issues

It is the view of this thesis that multimedia applications are part of the natural evolutionary path of merging the Local Area Network with the PABX. To date only a limited number of organisations use distributed systems which are based on networks suitable for the integration of real-time media. This results mainly from the proliferation of Ethernet as the industrial standard (IEEE 802.3). In chapter 2 several systems were discussed which demonstrate the new ways in which designers are beginning to visualise the PABX.

Undoubtedly some features invented by PABX manufacturers are created to make their products look competitive, others are there as the genuine result of client demands. There are many examples of PABXs that are sold with custom software to suit the needs of a particular organisation. Features such as specialist on-line directories and automatic call-routing between departments, are ways in which an organisation can increase the productivity of its workers. After all, idle time spent on the telephone when multiplied by a large work force costs a considerable amount of money. Changes in the custom software require further interaction with

the manufacturers. It would be far better if a system could be purchased and then tailored by its clients to satisfy their own needs. Integration of the PABX into a computing environment can provide this kind of control.

Both the centralised PABX and distributed LAN technologies are contenders for a position as the physical medium for transporting integrated voice and data. The centralised switch technology has dominated the commercial market. This is mainly because of its cost effectiveness and the large research and development investment made by telephone companies to produce custom-made devices that perform specialised coding, multiplexing, and switching functions.

Currently, LAN technologies are becoming worthy competitors for integrating data, voice and video. Firstly, bandwidth is becoming plentiful and less costly, that is to say, technologies for producing fast VLSI components are becoming cost effective and are being applied to network design (the Cambridge Fast Ring is an example). LANs also have the advantage that bandwidth is shared out to clients actually requesting service, rather than permanently dividing up bandwidth between clients, some of whom may rarely use it. It is the access rules of a network that define the time constraints by which data can be transported. Only a subset of LANs have properties that are suitable for the transport of voice as well as data. The Cambridge Fast Ring (CFR) has properties that allow bandwidth to be shared equally between requesters at a granularity which satisfies the requirement for the maximum transport delay of voice. Moreover, it is designed to have a data rate of 100Mbs^{-1} and if clocked at that speed would be capable of supporting 600 active telephone conversations (on the assumption that voice is the only medium on the network). A community of 200 extensions (chapter 3) are supported more easily in an integrated environment and will still leave sufficient bandwidth for data applications to operate comfortably.

The CFR is built from a two-chip set: a 100Mbs^{-1} serial-bus interface and an 8-bit parallel-bus component performing the station function. The latter is designed in such a way that it allows a cluster of station chips to be connected together using an 8-bit parallel bus. The cluster can then be connected to the serial ring through a single network attachment. Fast networks will almost certainly use fibre-optic cable as the communication medium, and require a high-precision component to tap the network. Reducing the number of network taps makes the system more economical. For example, a single network tap in a corridor may provide for several network-telephone extensions in its connecting rooms, with only a small incremental cost for each one.

System reliability must be of a high standard if distributed systems are to be considered as practical alternatives to supporting telephony applications. At the physical level, the network and hence the services, are vulnerable to disruption from simple connectivity failures. In chapter 3, the self-healing and braided-ring structures are described as a means of providing redundant connection paths. Although these designs have been documented for some time they have rarely been implemented in practical systems. Integrated media systems may well be the catalyst for the implementation of highly-reliable commercial-networks of this

kind. One example is the FDDI ring which has been designed to carry voice and data and contains a self-healing mechanism to address this very problem. The FDDI ring is currently being proposed as a MAN standard in IEEE 802.6.

The ISLAND project has demonstrated how a PABX may be functionally split up and distributed across a Local Area Network. Emphasis has been placed on the simplicity and cost effectiveness of the telephone, while at the same time improving its human interface. A visual aid in the form of a single-line display has been the main addition. The reaction obtained from a small sample of clients that used the system was that a visual display of options is far more desirable than a 'summary card' or manual which accompanies many of the conventional PABXs. The display would have been even more useful if it could have been extended to contain two or three lines of text. This kind of display is now much more available than at the time when the *Ringphone* was designed.

The essential philosophy behind the ISLAND telephone is that it is simple, it executes a single voice protocol, and is controlled by a small set of primitives which configure the telephone in a variety of operating modes. In itself it contains no inherent knowledge or state vital to the configuration of the surrounding system. In the ISLAND environment applications wishing to control voice do not have to know about the real-time properties of the media.

The two principle operations a client may wish to carry out on voice is (a) storage and playback and (b) conferencing. Two servers were designed to perform these functions: the *Translator*, which is used to provide a voice interface to a network file server, and the *Conference Server* which sums several voice-streams together and then retransmits the result to the parties involved. Both of these servers require a control interface similar to the telephone. The conference service was not built for the demonstration system, mainly because of the problems foreseen in achieving the required data-transfer rates through the network interface of a $10Mbs^{-1}$ Cambridge Ring. This conclusion was confirmed by the difficulties encountered in coercing an 8MHz 68000 processor to execute the voice protocol, and service both the voice digitising hardware and the network interface at the required speed.

Future implementations will be easier to develop on the CFR than on the original Cambridge Ring. A single CFR packet can transport an ISLAND voice packet, containing 2ms of voice samples, and incur only a small fraction of the protocol overhead found in the $10Mbs^{-1}$ Cambridge Ring (CR). This is because the CR can only send 2ms of voice by assembling 14 di-bytes into a buffer and transmitting it as a *Basic Block*. Further enhancement of the operation can be achieved by using extra hardware to buffer voice samples from a CODEC into complete 2ms blocks before interrupting the processor to initiate a handling routine. Chapter 4 suggests two architectures which may be suitable to perform this function. The advantages of improving efficiency at this interface lead to the possibility of connecting several telephones to one network station. This is a concession to the original architecture because it reduces availability of service when a single station fails. However, it is considerably more practical to implement a large system in this way.

The voice protocol (chapter 3) produced acceptable voice quality between two *Ringphones*. The corrective action taken by the playback pointers in the event of erroneous block transfers, generally went unnoticed. The automatic synchronisation of voice sample-clocks proved to be unnecessary. The software-controlled crystal-oscillators used in the demonstration system turned out to be relatively stable and could easily be adjusted to $1{:}10^4$ of the required sample rate. The relative drift of pointers between two telephones would slowly (in the order of minutes) manifest itself as a buffer pointer consistently out of place for the arrival of several successive voice blocks. This state is recognised and corrected by the same mechanism employed in the case of lost blocks.

10.2 Management and Control

For the purposes of controlling a network telephone, the Cambridge SSP protocol was used. Functions available at the telephone's network interface permit simple operations to be carried out such as: start sending voice to a station, receive voice from a station, display some text, activate a tone. This simple message request/reply mechanism was found to be lightweight, fast and well suited for executing these simple remote actions. To prevent telephone control by unauthorised parties, the network interface has been protected. A valid unique identifier (UID) must be presented with any request before it will be executed. UIDs are known only by authorised controllers.

A telephone call can be represented by a finite state machine. Each state transition requires several operations to be carried out on the telephone. In the demonstration system the actions making up the state transitions were predominantly SSP calls. For a human operator their execution was fast enough to make the overall state transition appear to be a single event.

Telephone control, by default, originates from either a reliable exchange service, or from an application running on an arbitrary client or service machine. In both cases, the controlling party has no knowledge of the voice protocol. By encapsulating control of voice behind the telephone network interface, modularity is maintained, and control can be dynamically transferred between machines. The management of voice is therefore considerably simplified.

Three interfaces were designed for the purpose of *Ringphone* control. Firstly, a simple interface to the telephone which allows a session machine to remotely cause key presses. Secondly, the ability of a machine to register interest in a telephone; the *Exchange* will then send commentary information to the registered party enabling, for example, a workstation to present the details of an incoming call on its own display. The third scheme requires a greater time investment by would-be programmers and constitutes a general mechanism for passing control of telephone resources to applications. These may be experimental or fixed services outside the function of the *Exchange*. By adding a new number to the directory with a *service* attribute (see chapter 5), the service machine can be dialled up

by a telephone. The resulting association is created without establishing a voice channel. While the service remains the current association, it may take over control of the telephone through the remote interface and, for example, change the *Ringphone* key event-vector to send key-presses to its own station. Control reliability is ensured in the event of a service crash by the use of the '*' key which always redirects control back to the *Exchange* service. It was felt that these three mechanisms provided for the various degrees of control needed by applications.

To date, only simple applications using the first kind of interface have been implemented. Using relatively simple shell scripts written on machines running the Tripos operating system and on others running UNIX, it has been possible to dial numbers from a command shell as if they were being physically dialled on a local *Ringphone*. All the menu features described in chapter 5, are accessible in this way. There are numerous ways in which these scripts can be used, for example in diary programs to place a call at a particular time, or in association with personnel files to allow dialling by name.

10.3 Management and Reliability

Considerable effort was put into designing a resilient *Exchange* service, for the following reasons. Most people expect the telephone to be a reliable piece of equipment and to provide a highly available communication service. If the new systems are to be adopted they must prove themselves to be dependable and advantageous to use. There is always a certain amount of prejudice against a new system and this has to be won over. The human interface and feature content of an integrated system will generally convince most people of the advantages. However, the systems are complicated and contain many component dependencies. The result is a reliability problem and if it is not solved, the system will lose favour.

In chapter 6 several 'reliable systems' were reviewed. Universally the solution to poor system-component reliability has been to replicate the system dependencies. The two mainstream approaches to the problem are: (1) Parallel replicated systems and (2) Hot Stand-by systems. There are many variations on the way that these approaches can be used.

Availability is one of the primary attributes associated with a telephone service. On the assumption that a LAN can be made into a highly available network, and that telephones are sufficiently simple that they rarely fail, providing the required service amounts to ensuring that the control mechanism is also highly available. A LAN/PABX has the advantage that the telephone controller can fail but the existing virtual connections continue to operate. It is only the state changes in telephone associations that are prevented by the failure. This property allows a recovery task a greater amount of time to perform its operation before the client community notices the problem.

The *Exchange* service needed to be extensible in both its fault tolerance and per-

formance characteristics. A variation on the Hot-standby approach was considered to be the best solution for this service. By replicating the *Exchange* with all its replicated servers actively offering service, the client load can be shared out between them. This mechanism increases performance when more servers are added. However, to achieve this kind of distributed control there must also be some inter-server interaction. When several calls handled by different server components converge on one *Ringphone*, synchronisation is required. The four-function association interface (ECHD), described in chapter 5, made it possible to build a simple remote-interface (chapter 6) to extend control in this situation. In some Hot-standby systems there is a danger of dormant faults in the back-up system which do not show themselves until recovery occurs. By using all the servers in an active mode this possibility is reduced.

This kind of parallel operation provides highly available access to the *Exchange* service. However, each server contains essential state information and, in the event of its own failure, it must copy its state to a place from where it may be safely recovered. To avoid the dependency of a network filing service, typically an expensive resource and not replicated, state can be copied to the other control servers. In this case another server is able to take over the operation of a failed *Exchange* component.

Two state distribution schemes were considered (chapter 7): Multicast, and a Virtual Ring protocol. The impact of these different approaches is assessed in chapter 9. In the context of the ISLAND system using a maximum of five servers, both methods produced similar performance characteristics. However, the results did imply that there would be a more significant benefit of the Virtual Ring protocol over Multicast if a system were to use as many as ten servers. For the purposes of failure detection the servers were ordered in a logical ring giving each server the responsibility of monitoring the next server in the ring. An evaluation of the monitoring period concluded that it could be as low as 750ms, without detrimental performance loss. However, the recovery period would generally be significantly longer than this, making a fast monitor unnecessary.

The evaluation predicts that a population of 200 telephones, assuming conventional telephone usage patterns, can be supported by 3 servers with a mean service delay of less than one second when handling events. In this configuration loss of a single server will not increase the mean delay above 1 second. Clearly more servers can be used to decrease the service delay and increase the fault tolerance.

When managing a distributed program a central control-point is essential for coordinating its component parts. A server suitable for managing ISLAND's distributed application is described in chapter 8. An important aspect of its design was that it should not become a dependency for the reliable distributed system. The server was used as an operator's interface to the PABX, and to provide management of the service's resilience factor.

To further enhance reliability a *Watchdog* timer could have been designed to locally monitor each control server. A *Watchdog* timer is a device which is connected

to a processor bus and, in normal operation, expects to communicate with the processor at regular intervals. If the *Watchdog* timer expires, it responds by resetting the processor. Throughout the design of the ISLAND system failures have been assumed to be fail-stop in effect. In the rarer cases of failure this may not be true. Some errors may cause a machine to malfunction in a way which cannot be determined by remotely monitoring its operation. A *Watchdog* timer may detect some of these cases and convert them to the fail-stop case.

In all cases the measures taken to prevent failure of a system are only effective against its expected failure modes. There will always be cases which have not been taken into account. System design can only try and optimise fault tolerance against the likely causes of failure.

10.4 Future Work

The ISLAND project has shown that reliable integrated systems are feasible and have interesting applications. However, ISLAND was a small development restricted by network bandwidth and the resources available to build network telephones. A future system could be designed around a Cambridge Fast Ring which would benefit from an order of magnitude speed-up in both the network data rate and data throughput in the interface hardware. If a sufficiently large system were implemented it could be put to use in serious day-to-day applications in a laboratory environment.

In order to expand the system on the scale required an abundant supply of network telephones would be needed. The *Telephone Concentrator* described in chapter 4 would be a necessary part of this new design, allowing several telephones to be connected to a single network-station and controlled by one processor.

Applications for the immediate future developed with the existing ISLAND system include a voice mail-service, whereby voice can be sent in a similar way to electronic mail. Every client would have a voice mail-box from which voice could be played back at a telephone in the same way that we use telephone answering machines today. The voice mail-service, however, would be an integral part of the system.

A popular aim of integrated system designers is multimedia 'Hypertext', that is, the non-linear representation of a document which may contain a combination of text, graphics, voice, or even video. A Hypertext document [Conklin 87] is designed to be browsed by a program that may display attributes of a topic that a reader is interested in. This saves a reader having to search through an amorphous body of information. The approach also allows the association of useful concepts as a means of choosing different paths by which a document may be read. Hypertext will therefore represent information in a way more closely related to the way we think. An example of such a system is the THOTH-II Hypertext browser at Bell Laboratories [Collier 86]. Some research is being carried out at the University of Cambridge into the representation of different media in documents which could

be extended for use in a Hypertext system. For this purpose, the ISLAND control primitives are being used to integrate voice into documents displayed on a Xerox workstation [Anupundi 86]. This project is still in its early stages.

Further work needs to be done to consider how video may be managed within the system. Real-time video is only just becoming feasible for integration into a LAN. The new generation of $100Mbs^{-1}$ LANs combined with cost effective video-CODECs should encourage research in this area. Currently the project has one video-framestore connected to a Cambridge Ring for the purposes of capturing and storing images. It will soon be joined by a second with the intention of carrying out research into suitable mechanisms for transferring real-time video across a LAN. This research involves the design of new coding techniques to compress video without introducing excessive delay and maintaining the essential picture quality [Nagioff 87].

In the future it may be possible to develop a video-phone using this system combined with a Cambridge Fast Ring network. The management of video streams will also need to be reliable and highly available. It will be possible to control video connections with the same reliable management policies that ISLAND has used for voice.

It is only when Local Area Networks and the services they support have proven themselves to be dependable alternatives to conventional facilities, that the new integrated systems will be adopted. If the primitives for managing the various media are well chosen, many client programmers will create multimedia applications for their own purposes. We can look to the future for interesting and novel applications.

References

[Ades 86] S. Ades, R. Want, R. S. Calnan,
Protocols for Real Time Voice Communications on a Packet Local Area Network,
Proc. IEEE ICC 86 Conference, Toronto, June 1986.

[Ades 87a] S. Ades, D. C. Swinehart,
Voice Annotation and Editing in a Workstation Environment,
Proc. AVIOS 86 Voice Applications Conference,
San Fransisco, April 1987.

[Ades 87b] S. Ades,
An Architecture for the Provision of Integrated Services,
Ph.D. dissertation, Tech. report No. 114, Univ. of Cambridge, September 1987.

[Anupundi 86] S. Anupundi,
Presentation and Annotation of Multi-media Documents,
Ph.D. Thesis Proposal, Univ. of Cambridge, October 1986.

[Ayache 82] J. Ayache, J. Courtiat, and M. Diaz,
REBUS, A Fault-Tolerant Distributed System for Industrial Real-Time Control,
IEEE Transactions on Computers, Vol. c-31, No. 7, July 1982.

[Bacon 87] J. M. Bacon, K. G. Hamilton,
Distributing Computing with RPC: The Cambridge Approach,
Proc. IFIP conference on Distributing Processing,
Amsterdam, October 1987.
(Also Univ. of Cambridge Tech. report No. 117.)

[Barrett 85] V. P. Barrett,
Integrating Local Networks — the AT&T approach,
Proc. Localnet '85, Online Publications, Pinner UK,
pp325–336, 1985.

[Bartlett 81] J. F. Bartlett,
A NonStop Kernel,
ACM Proc. of the Eighth Symposium on Operating System Principles, pp22–29, December 1981.

[Bear 76] D. Bear,
Principles of Telecommunication-Traffic Engineering,
Peter Peregrinus Ltd, 1976.

[Birrell 83] A. D. Birrell, B. J. Nelson,
Implementing Remote Procedure Calls,
Xerox PARC Tech. report CSL-83-7, December 1983.

[Birman 85] K. Birman,
Replication and Fault Tolerance in the ISIS System,
Proc. of 11th Sym. on Op. Sys. Principles,
ACM Operating Systems Review, 19, No.4, pp79–86, 1985.

[Brady 68] P. T. Brady,
A Statistical Analysis of On-Off Patterns in 16 Conversations,
Bell System Technical Journal, Vol. 47, pp73-91, Jan 1968.

[BT 83] **An Explanation of the Loss of Service Requirements of Strategic Network Plans TP:I01/4 and IS:I01/4,**
British Telecom Standards, NS2.2.3,
Memorandum 109, Issue 1, October 1983.

[Bullington 59] K. Bullington, J. M. Fraser,
Engineering Aspects of TASI,
The Bell System Tech. Journal, Vol. 38, March 1959.

[Calnan 87] R. S. Calnan,
ISLAND: A Distributed Multimedia System,
Proc. of Conference Globecom '87,
pp744–749, Tokyo, Japan, Nov. 15-18 1987.

[CCITT 72] **Pulse Code Modulation (PCM) of Voice Frequencies,**
Recommendation G.711 (Geneva 1972), CCITT Red Book,
Eighth Plenary Assembly, 1984.

[Chen 78] L. Chen, A. Avizienis,
N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation,
Dig. of FTCS-8, Toulouse, France, pp3-9, 1978.

[Chor 85] B. Chor, and B. A. Coan,
A Simple and Efficient Randomized Byzantine Agreement Algorithm,
IEEE Transactions on Software Engineering,
Vol. SE-11, No.6, June 1985.

[Christodoulakis 86] S. Christodoulakis et al.,
Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A Model and a System,
ACM Trans. on Office Information Systems,
Vol. 4, No. 4, October 1986.

[Clark 86] P. F. Clark et al.,
Unison –
Communications Research for Office Applications,
Electronics and Power, pp665–667, September 1986.

[Cooper 85a] E. C. Cooper,
Replicated Distributed Programs,
Proc. 11th Sym. on Op. Sys. Principles,
ACM Operating Systems Review, 19, No. 4, pp63–78, 1985.

[Cooper 85b] R. C. B. Cooper, K. G. Hamilton,
Preserving Abstraction in Concurrent Programming,
Tech. report No. 76, August 1985.
Univ. of Cambridge.

[Collier 86] George Collier,
THOTH-II: A Hypertext Browser,
Bell Comm. Research, Tech Memorandum,
TM-ARH-002762, May 1986.

[Conklin 87] J. Conklin,
Hypertext: An Introduction and Survey,
IEEE Computer, pp17–41, Sept 1987

[Craft 85] D. H. Craft,
Resource Management in a Distributed Computing System,
Ph.D. dissertation, Tech report No.73,
Univ. of Cambridge, March 1985.

[Craft 85] D. H. Craft,
Writing Servers in Mayflower,
PERG Meeting No. 73, Project Mayflower Document,
Univ. of Cambridge, October 1985.

[Davidson 85] P. J. Davidson,
Review of the CCITT Recommendations for Integrated Service Digital Networks (ISDN),
British Telecom journal, Vol. 3, No. 4, October 1985.

[DES 77] **Data Encryption Standard**
Federal Information Processing Standard (FIPS),
Pub. 46, pp651-670, U.S. Department of Commerce Jan 1977.
Also found in C. H. Meyer and S. M. Matyas, **Cryptography: A New Dimension in Computer Data Security,**
Appendix A. Wiley Interscience Pub.

[Dion 81] J. Dion,
Reliable Storage in a Distributed System,
Ph.D. dissertation, Tech. report No. 16,
Univ. of Cambridge, February 1981.

[Dolev 81] D. Dolev,
The Byzantine Generals Strike Again,
Report. No. STAN-CS-81-846, March 1981,
Univ. of Stanford, California.

[Ellis 84] C. Ellis,
The PABX as a Local Area Network,
Proc. Networks '84,
Online Publications pp445–450, London, July 1984.

[Falconer 85] R. M. Falconer, R. L. Adams,
Orwell: a protocol for an integrated services local network,
British Telecom Tech. Journal, October 1985.

[Falconer 83] R. M. Falconer,
A Study of Techniques for Enhancing the Reliability of Ring Local Area Networks (LANs)
Proc. IFIP WG6.4 University of Kent Workshop,
September 1983.

[GEC 81] **The Monarch 120 Call Connect System: Technical Description Part 1,**
General Electric Company, 1981.

[Garnett 83] N. H. Garnett,
Intelligent Network Interfaces
Ph.D. dissertation, Tech. report No. 46, May 1983
Univ. of Cambridge.

[Gibbons 80] J. J. Gibbons,
SSP – A Single Shot Protocol for the Ring,
Systems Research Group Note, September 1980
Univ. of Cambridge.

[Gifford 84] D. Gifford,
The Space Shuttle Primary Computer System,
ACM Vol. 27, No. 9, September 1984. pp874-936.

[Girling 83] C. G. Girling,
Representation and Authentication on Computer Networks,
Ph.D. dissertation, Tech. report No. 37,
Univ. of Cambridge. April 1983.

[Gray 78] J. N. Gray,
Notes on Database Operating Systems,
Operating Systems: An Advanced Course,
Springer–Verlag, pp393-481, 1978.

[Greenway 85] J. Greenway,
The Next Step: An Integrated PBX/LAN,
Localnet '85 USA, Online Pubs UK pp97–108, 1985.

[Gruber 83] John Gruber, Leo Strawczynski,
Judging Speech in Dynamically Managed Voice Systems,
Telesis 1983 Two (Bell Northern Research), pp30–34, 1983.

[Hamilton 84] K. G. Hamilton,
A Remote Procedure Call System,
Ph.D. dissertation, Tech. report No. 70, Dec. 1984,
Univ. of Cambridge.

[Hoare 74] C. A. R. Hoare,
Monitors: An Operating System Structuring Concept,
Communications of the ACM Vol. 17 No. 10 pp549–557, 1974.

[Hopper 78] A. Hopper,
Local Area Computer Communications Networks,
Ph.D. dissertation Tech. report No. 7, April 1978,
Univ. of Cambridge,

[Hopper 86] A. Hopper, R. M. Needham,
The Cambridge Fast Ring Networking System (CFR),
Tech. report No. 90, June 1986,
Univ. of Cambridge.

[Hull 84] R. Hull, F. Halsall, R. L. Grimsdale (Univ. of Sussex UK)
Virtual Resource Ring: Technique of Decentralised Resource Management in Fault-Tolerant Distributed Computer Systems,
IEE Proceedings, Vol. 131, Pt. E, No.2, March 1984.

[IEEE 83] IEEE Project 802, Local-Area Network Standards,
Draft IEEE Standard 802.5, Token-Ring Access Method and Physical-Layer Specifications,
Working Draft, September 1983.

[ITU 85] **I-Series Recommendations,**
International Telecommunication Union,
CCITT VIIIth Plenary Assembly, Fascicle 111.5, 1985.

[Johnson 81] M. A. Johnson,
Byte Stream Protocol Specification,
Systems Research Group Doc., April 1981,
Univ. of Cambridge,

[Kay 83] P. M. Kay,
A New Distributed PBX for Voice/Data Integration,
Proc. Localnet '83, Online Pubs UK, pp485–493, New York 1983.

[Kleinrock 75] L. Kleinrock,
Queuing Systems Vol1: Theory
John Wiley & son Inc., 1975.

[Lamport 80] L. Lamport, R. Shostak, M. Pease,
The Byzantine Generals Problem,
ACM Transactions on Programming Languages and Systems,
Vol 4, No 3, pp382–401, July 1982. (Originally Pub. Nov. 1980)

[Lazar 85] A. A. Lazar et al. (Univ. of Columbia),
An Optical Fiber-Based Integrated LAN for MAGNET's Testbed Environment,
IEEE Journal on Selected Areas in Comms.,
Vol. 3, No. 6,pp872–882, November 1985.

[Le Lann 77] G. Le Lann,
Distributing Computing - Towards a Formal Approach,
Proc. IFIP, Toronto, Canada, 1977.

[Leslie 83] I. M. Leslie,
Extending the Local Area Network,
Ph.D. dissertation, Tech. report No. 43,
Univ. of Cambridge, Feb. 1983.

[Leslie 84] I. M. Leslie, R. M. Needham, J. W. Burren, G. C. Adams,
The Architecture of the Universe Network,
ACM Sigcomm '84 Symposium, Montreal, Canada, June 1984.

[Limb 82] J. O. Limb, C. Flores,
Description of Fasnet —
A Unidirectional Local-Area Communication Network,
The Bell System Technical Journal, Vol. 61, No. 7, Sept. 1982.

[Liskov 81] B. Liskov et al. (MIT),
CLU Reference Manual ,
Lecture Notes in Computer Science No. 114,
Springer-Verlag, Heidelberg 1981.

[Liskov 82] B. Liskov et al.,
Guardians and Actions: Linguistic Support for Robust Distributed Programs,
Conf. Record of the Ninth Annual ACM Symposium on Principles of Programming Languages, pp7–19, 1982.

[Musa 75] J.D. Musa,
A Theory of Software Reliability and its Application,
IEEE Trans. Software Eng., SE-1, 3, pp312-327, 1975.

[Motorola 82] **MC68000 16-Bit Microprocessor User's Manual,**
Prentice Hall, 1982.

[Nagioff 87] O. Nagioff,
Image Encoding,
Ph.D. Thesis Proposal, March 1987,
Univ. of Cambridge.

[Needham 82] R. M. Needham, A. J. Herbert,
The Cambridge Distributed Computing Sytem,
Addison-Wesley, London 1982.

[Newman 88] P. Newman (Univ. of Cambridge),
A Broad-band Packet Switch for Multi-Service Communications,
To be published in *IEEE Infocom '88*, March 1988.

[Nicholson 83] Robert T. Nicholson,
Integrating Voice in the Office World,
BYTE Publications Inc., Dec. 1983.

[Ody 84] N. Ody,
Terminal Handling in a Distributed System,
Ph.D. dissertation, August 1984,
Univ. of Cambridge.

[Peterson 72] W. W. Peterson, E. J. Weldon.
Error-Correcting Codes,
MIT Press, 1972.

[Porter 86] J. D. Porter,
MAC Layer Interconnection of Homogeneous LANs,
Thesis Proposal, October 1986,
Univ. of Cambridge.

[Randell 78] B. Randell et al.,
Reliability Issues in Computing System Design,
Computing Surveys 10, 2, pp123-165, June 1978.

[Redman 87] B. E. Redman,
A User Programmable Telephone Switch,
EUUG Conference, May 1987,
On ship M/S Mariella voyage Helsinki–Stockholm.

[Richards 79] M. Richards et al.,
TRIPOS —
A Portable Operating System for Minicomputers,
Software - Practice and Experience, June 1979.

[Ritchie 74] D.M. Ritchie,
The UNIX Time Sharing System,
Communications of the ACM Vol.17, No.7, pp365-375, 1974.

[Robin 84] G. Robin,
Customer Installations for the ISDN,
IEEE Communications Magazine, No. 22 No.4, April 1984.

[Ross 86] F. E. Ross,
FDDI - a Tutorial,
IEEE Communications Magazine, Vol.24, No.5, May 1986.

[Rossi 84] B. C. Rossi, M. Coronaro,
Integrated Office Communication System
Localnet '84, Online Publications, Pinner UK, 1984.

[Schmandt 85] C. Schmandt, B. Arons,
Phone Slave:
A Graphical Telecommunications Interface,
Proc. of the Soc for Information Display, 26(1):79-82. 1985.

[Melliar-Smith 82] P. M. Melliar-Smith, R. L. Schwartz,
Formal Specification and Mechanical Verification of SIFT: A Fault-Tolerant Flight Computer,
IEEE Trans. on Computers, Vol C-31, Number 7, pp616–630 July 1982.

[Shoch 82] J. F. Shoch, Jon A. Hupp (Xerox PARC),
The 'Worm' Programs—
Early Experience with a Distributed Computation,
Comm. ACM, pp172-180, March 1982.

[Shooman 79] M. L. Shooman,
Software Reliability,
Ch. 9, pp355-422, Computer Systems Reliability,
Authors: T. Anderson and B. Randell,
Cambridge University Press, 1979.

[Smith 85] G. A. Smith, Neil Brewer,
Age and Individual Differences in
Correct and Error Reaction Times,
British Journal of Psychology, 76, pp199-203, 1985.

[Swinehart 83] D. C. Swinehart, L. C. Stewart, and S. M. Ornstein,
Adding Voice to an Office Network,
IEEE GlobeCom '83 Conference, November 1983.

[Swinehart 86] D. C. Swinehart et al. (Xerox PARC),
A Structural View of the
Cedar Programming Environment,
ACM Trans. on Prog. Languages and Systems, October 1986.

[Swinehart 87] D. C. Swinehart et al.,
An Experimental Environment for
Voice System Development,
IEEE Knowledge Office Engineering, 1(1):39-48, February 1987.

[Temple 84] **The Design of a Ring Communication Network,**
Ph.D. dissertation, Tech. report No. 52, 1984.
Univ. of Cambridge.

[Tennenhouse 86] D. L. Tennenhouse,
The Unison Data Link (UDL) protocol specification,
Unison Project(Alvey Programme), Doc. Ref. UC022, Oct. 1986.

[Tennenhouse 87] D. L. Tennenhouse, I. M. Leslie, and R. M. Needham et al.,
Exploiting Wideband ISDN: The Unison Exchange,
InfoComm '87, San Francisco, March 1987.

[Terry 87] D. B. Terry, D. C. Swinehart,
Managing Stored Voice in the Etherphone System,
Globcom '87, pp48-61, Tokyo, Japan, November 1987.

[Thomas 85] Robert. H. Thomas, Harry C. Forsdick et al.,
Diamond: A Multimedia Message System Built Upon a Distributed Architecture,
IEEE Computer, November 1985.

[Walker 78] R. D. H. Walker,
Basic Ring Transport Protocol,
Systems Research Group Note,
Univ. of Cambridge, October 1978.

[Wendsley 85] J. Wendsley,
August Systems Industrial Control,
Presented in *Resilient Computer Systems* by T. Anderson,
Collins, Chapter 13, pp232-246, 1985.

[Wilbur 85] S. Wilbur,
Local Area Networks,
Presented at *4th IEE Vac. School on Data Comm.*, Sept 1985.

[Wilkes 79] M. V. Wilkes, D. J. Wheeler,
The Cambridge Digital Communication Ring,
Local Area Communications Network Symposium,
Boston, USA, May 1979.

[Wilson 85] D. Wilson,
The Stratus Computer System,
Presented in *Resilient Computer Systems* by T. Anderson,
Collins, Chapter 12, pp208-231, 1985.

[Woodbury 87] L. Woodbury et al.,
A Modular Integrated Communications Environment, (MICE): A System for Prototyping and Evaluating Communications Services
ISS-87 Conference, March 1987.

[VMEbus 85] **The VMEbus Specification: Revision C.1**
Motorola Series in Solid State Electronics.
Printex, October 1985.