This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (https://dspace.lboro.ac.uk/) under the following Creative Commons Licence conditions.

For the full text of this licence, please go to:
http://creativecommons.org/licenses/by-nc-nd/2.5/
Multimedia Services in a Distributed Office

by

Brendan Joseph Murphy

A Doctoral Thesis

Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of the Loughborough University of Technology

January, 1990
Supervisor: Prof JWR Griffiths

Department of Electronic and Electrical Engineering
University of Technology
Loughborough, England

© by Brendan Joseph Murphy, 1990
Abstract

This thesis is concerned with the provision of multimedia services (involving voice, video, text and graphics) in an office environment. The office of the future is expected to comprise a heterogeneous collection of workstations and multimedia components (including fileservers, voice and video codecs, document scanners, laser printers, etc) interlinked by a high speed (digital) local area network. Every office is likely to have one or more connections to a public Integrated Services Digital Network (ISDN) providing integrated access (involving various types of traffic) to a very large number of subscribers.

This thesis considers general issues relating to the design of such an office. Particular attention is given to the problem of the integration of media both at the network and user levels. Much of this discussion draws on practical experience gained during the Alvey Unison Project in which experimental multimedia offices were interconnected using a pilot ISDN. The architecture of the Unison network is discussed with particular reference to its suitability for the support of multimedia services.

The bulk of this thesis is devoted to a description of the design and implementation of a number of prototype multimedia applications, and to an evaluation of their performance over the network. The handling of slow-scan video and high resolution images have been particular areas of interest.

Much emphasis is placed on the problem of control in a distributed environment, and a model is presented for the management of control based on the use of a directory-like service. This model also provides a mechanism for locating an office service based on the name of the user to whom it belongs.
Acknowledgements

I would like to extend my special thanks to my supervisor, Professor JWR Griffiths, for his help and support during the course of this research. I would also like to thank his secretary, Sheila Clarson, for her help and friendship.

I would like to thank all my friends who have worked on the Unison Project at Loughborough, namely Vicky Hardman, Guojun Lu, Dave Parish, Tim Rodgers and Mahoob Siddiqui. Apart from contributing to many helpful discussions, they have provided a congenial working atmosphere which I shall long remember.

I would also like to thank my other colleagues, too numerous to mention, who have been involved with Unison and who have contributed, directly and indirectly, to the work described in this thesis. In particular, I would like to thank Chris Cooper from the Rutherford Appleton Laboratory for many interesting discussions during the course of this research, and for proof reading chapter 3 of this thesis.

I would like to acknowledge the financial support from the Science and Engineering Research Council (through the Alvey Unison Project) and the EEC (through the RACE Multimed Project).

Last but not least, I would like to thank my parents for their continued support and encouragement during these past years. My thanks also to Linda McQueen for her help, patience and good humour.

This thesis was prepared using the WordStar 4 word processor, the GEM Draw graphics package and the Xerox Ventura desktop publishing system all running on IBM PC compatibles. Final output was on an Apple LaserWriter Plus.
Symbols and Acronyms

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>Acknowledgement Block</td>
</tr>
<tr>
<td>ACM</td>
<td>Association for Computing Machinery</td>
</tr>
<tr>
<td>ADPCM</td>
<td>Adaptive Differential Pulse Code Modulation</td>
</tr>
<tr>
<td>ARM</td>
<td>Acorn RISC Machine</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>ATM</td>
<td>Asynchronous Transfer Mode</td>
</tr>
<tr>
<td>BASIC</td>
<td>Beginner's All-purpose Symbolic Instruction Code</td>
</tr>
<tr>
<td>B-ISDN</td>
<td>Broadband Integrated Services Digital Network</td>
</tr>
<tr>
<td>BBD</td>
<td>Basic Block Descriptor</td>
</tr>
<tr>
<td>BBP</td>
<td>Basic Block Protocol</td>
</tr>
<tr>
<td>BSP</td>
<td>Byte Stream Protocol</td>
</tr>
<tr>
<td>BT</td>
<td>British Telecom</td>
</tr>
<tr>
<td>CAD</td>
<td>Computer Aided Design</td>
</tr>
<tr>
<td>CBR</td>
<td>Constant Bit Rate</td>
</tr>
<tr>
<td>CCD</td>
<td>Charge Coupled Device</td>
</tr>
<tr>
<td>CCITT</td>
<td>Consultative Committee on International Telephone and Telegraph</td>
</tr>
<tr>
<td>CFR</td>
<td>Cambridge Fast Ring</td>
</tr>
<tr>
<td>CLI</td>
<td>Command Line Interpreter</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complimentary Metal Oxide Semiconductor</td>
</tr>
<tr>
<td>CR</td>
<td>Cambridge Ring</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct Memory Access</td>
</tr>
<tr>
<td>DoC</td>
<td>Domain of Control</td>
</tr>
<tr>
<td>DPI</td>
<td>Dots Per Inch</td>
</tr>
<tr>
<td>ECL</td>
<td>Emitter Coupled Logic</td>
</tr>
<tr>
<td>EOB</td>
<td>End of Block Marker</td>
</tr>
<tr>
<td>EOF</td>
<td>End of Frame Block</td>
</tr>
<tr>
<td>EOS</td>
<td>End of Still Image Block</td>
</tr>
<tr>
<td>EPROM</td>
<td>Electrically Programmable Read Only Memory</td>
</tr>
<tr>
<td>FAX</td>
<td>Facsimile</td>
</tr>
<tr>
<td>FIFO</td>
<td>First In, First Out</td>
</tr>
<tr>
<td>FRAMEACK</td>
<td>Frame Acknowledgement Block</td>
</tr>
<tr>
<td>FTAM</td>
<td>File Transfer and Access Management</td>
</tr>
<tr>
<td>GDP</td>
<td>Graphics Display Processor</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>ID</td>
<td>Identifier</td>
</tr>
<tr>
<td>IDA</td>
<td>Integrated Digital Access</td>
</tr>
<tr>
<td>IEE</td>
<td>Institute of Electrical Engineers (UK)</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electronic and Electrical Engineers (USA)</td>
</tr>
<tr>
<td>ILAN</td>
<td>Integrated Local Area Network</td>
</tr>
</tbody>
</table>
IP  Internet Protocol
ISA  Integrated Services Architecture
ISDN  Integrated Services Digital Network
ISO  International Standards Organisation
kbps  1000 Bits Per Second
kbyte  $2^{10}$ Bytes
kHz  1000 Hertz
LAN  Local Area Network
LVC  Lightweight Virtual Circuit
MAC  Media Access Control
Mbyte  $2^{20}$ Bytes
MCS  Multicast Control Service
MDS  Microprocessor Development System or Multicast Data Service
ms  $10^{-3}$ Seconds
OCR  Optical Character Recognition
OPEN  Virtual Connection Set-up Request Block
OPENACK  Virtual Connection Set-up Reply Block
ORL  Olivetti Research Laboratory (Cambridge)
PABX  Private Automatic Branch Exchange
PARC  Palo Alto Research Center (Xerox)
PC  Personal Computer
PCM  Pulse Code Modulation
PDU  Protocol Data Unit
PSS  Packet Switched Service
PTT  Post, Telephone and Telegraph
RAL  Rutherford Appleton Laboratory (Oxford)
RAM  Random Access Memory
RISC  Reduced Instruction Set Computer
ROM  Read Only Memory
RPC  Remote Procedure Call
SB-ADPCM  Sub-band Adaptive Differential Pulse Code Modulation
SCSI  Small Computer Systems Interface
SERC  Science and Engineering Research Council
SSP  Single Shot Protocol
SNA  Systems Network Architecture
SSPREQ  Single Shot Protocol Request Block
SSPRPLY  Single Shot Protocol Reply Block
STM  Synchronous Transfer Mode
SWI  Software Interrupt
TCB  Tube Control Block or Task Control Block
UDL  Unison Data Link
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMIST</td>
<td>University of Manchester Institute of Science and Technology</td>
</tr>
<tr>
<td>VBR</td>
<td>Variable Bit Rate</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very Large Scale Integration</td>
</tr>
<tr>
<td>WAN</td>
<td>Wide Area Network</td>
</tr>
<tr>
<td>WIMP</td>
<td>Windows, Icons, Menus and Pointers</td>
</tr>
<tr>
<td>XDR</td>
<td>External Data Representation</td>
</tr>
</tbody>
</table>
# Table of Contents

## 1 Introduction
1.1 Introduction 1
1.2 Background 1
   1.2.1 Personal Computers 1
   1.2.2 Professional Workstations 2
   1.2.3 Computer Communications 2
1.3 Current Developments 4
   1.3.1 Office Environment 4
   1.3.2 Multimedia Applications 4
   1.3.3 Integrated LANs 5
   1.3.4 Wide Area Networks 5
   1.3.5 User Level Integration 6
1.4 Aims 6
1.5 Context 7
1.6 Structure of this Thesis 7
1.7 Extent of Collaboration 8

## 2 Design Considerations for an Integrated Office
2.1 Introduction 9
2.2 Design of the Local Area Network 10
   2.2.1 Introduction 10
   2.2.2 General Properties of LANs 10
   2.2.3 An Integrated Services LAN 11
      2.2.3.1 The Spectrum of Traffic Types 11
      2.2.3.2 The Bandwidth Allocation Problem 13
2.3 Design of the Wide Area Network 14
2.4 The Office Architecture 15
   2.4.1 Introduction 15
   2.4.2 Arguments for a Distributed Architecture 16
      2.4.2.1 Performance 16
      2.4.2.2 Reliability 17
      2.4.2.3 Extensibility 17
      2.4.2.4 Cost 17
   2.4.3 Integration in a Distributed Environment 17
   2.4.4 Location of Functionality 18
   2.4.5 Location of Control 18
   2.4.6 Design of the Workstation 19
   2.4.7 Data Replication 22
   2.4.8 Synchronisation 23
3 The Network Architecture
3.1 Introduction
3.2 Background
3.3 Overview of the Network
3.4 The Cambridge Ring Architecture
  3.4.1 Introduction
  3.4.2 Basic Design
  3.4.3 The Hardware
  3.4.4 Protocols
    3.4.4.1 Introduction
    3.4.4.2 Basic Block Protocol
    3.4.4.3 Single Shot Protocol
    3.4.4.4 LVC Protocol
  3.4.5 Naming, Addressing and Routing in Universe
    3.4.5.1 Naming and Addressing
    3.4.5.2 Routing
3.5 The Cambridge Fast Ring Architecture
  3.5.1 Introduction
  3.5.2 General Design
  3.5.3 Hardware
  3.5.4 Protocols
    3.5.4.1 Introduction
    3.5.4.2 Associations
    3.5.4.3 Unison Data Link Protocol
    3.5.4.4 Unity RPC
  3.5.5 Naming and Addressing in Unison
    3.5.5.1 Naming Scheme
    3.5.5.2 Secretary
  3.5.6 Association Set-up
3.6 The Long-Haul Segment
3.7 The Unison Network Architecture
  3.7.1 Introduction
  3.7.2 Exchange Components
    3.7.2.1 Exchange
    3.7.2.2 Ramp
    3.7.2.3 Portal
  3.7.3 Addressing and Routing
  3.7.4 Multi-service Support
  3.7.5 Exchange Management
  3.7.6 Call Set-up Over Unison
3.7.6.1 Introduction  46
3.7.6.2 Association Set-up Across Unison  46
3.7.6.3 LVC Set-up Across Unison  47

3.8 Summary  48

4 The Cambridge Ring Multimedia Office

4.1 Introduction  50

4.2 Overview of the Office  51
  4.2.1 Introduction  51
  4.2.2 Office Components
    4.2.2.1 The A500 Workstation  52
    4.2.2.2 The Video Framestore  52
    4.2.2.3 The Telephone  54
    4.2.2.4 The Wideband Speech System  54
    4.2.2.5 The Tripos Server  55
  4.2.3 Office Services
    4.2.3.1 Video Services  55
    4.2.3.2 Voice Services  55
    4.2.3.3 The Electronic Blackboard Service  55
    4.2.3.4 Workstation Services  56
    4.2.3.5 Distributed Computing Facilities  56

4.3 The A500 to Cambridge Ring Interface  56
  4.3.1 Introduction  56
  4.3.2 Configuration  56
  4.3.3 Software Interface
    4.3.3.1 Ring Driver  57
    4.3.3.2 Tube Manager  58
  4.3.4 Operation  59
  4.3.5 Performance  59

4.4 The Video System  60
  4.4.1 The Background  60
  4.4.2 System Design  60
  4.4.3 System Implementation
    4.4.3.1 Framestore Software Development Environment  63
    4.4.3.2 A500 Software Development Environment  63
    4.4.3.3 The Framestore Software  64
    4.4.3.4 The Workstation Software  66
  4.4.4 The Protocols
    4.4.4.1 The Slow-Scan Video Protocol  67
    4.4.4.2 The Still Image Transfer Protocol  68
    4.4.4.3 The Framestore Print Protocol  70
    4.4.4.4 The Electronic Blackboard Protocol  70
5.3.4 Traffic Generator Control

5.4 The Workstation Window Environment

5.4.1 Introduction
5.4.2 Access to the Arthur WIMP Module
5.4.3 Detecting WIMP Events
5.4.4 Dispatching WIMP Events
5.4.5 Event Handler Run-time Environment
5.4.6 Window Support for the Console Handler
5.4.7 Operation
5.4.8 Performance
5.4.9 Evaluation

5.5 The Scanner System

5.5.1 Background
5.5.2 System Design
  5.5.2.1 Introduction
  5.5.2.2 Server Functionality
  5.5.2.3 Reading
  5.5.2.4 Locking
  5.5.2.5 Server Interface
  5.5.2.6 Client Access
5.5.3 Implementation
  5.5.3.1 The Scanner Hardware
  5.5.3.2 Device Driver
  5.5.3.3 Server
5.5.4 A Prototype Scanner Application
5.5.5 Performance
5.5.6 Evaluation
5.5.7 Providing a Fax Service

5.6 The Workstation Video System

5.6.1 Background
5.6.2 The ARM Video Module
5.6.3 Design
5.6.4 Implementation
  5.6.4.1 Access to the Video Module
  5.6.4.2 Transmission of Video Data
  5.6.4.3 Handling Multiple Streams
  5.6.4.4 Compression
  5.6.4.5 Termination
5.6.5 Operation
5.6.6 Performance
5.6.7 Evaluation

5.7 Multicast Video

5.7.1 Introduction
Chapter 1
Introduction

1.1 Introduction

Recent years have seen the emergence of a wide variety of information processing equipment based on digital computing technology. These systems are used for the capture, manipulation and display of primarily textual and numerical information. Current research is being directed towards incorporating other sorts of data including time-varying media such as voice and video. These multimedia systems promise to enhance the range of computing services available to the user and to improve the interaction between man and machine.

A parallel development has been in the application of digital techniques to the field of communications. There has been particular interest in the provision of integrated networks that can simultaneously support different classes of traffic including voice, video and conventional computer data. Such networks will enable the interconnection of multimedia systems to improve inter-personal communications and to extend multimedia services in general to cover a wider geographical area.

This thesis is concerned with the development of multimedia services in a networked office environment. The work is based on the practical experience gained from using experimental multimedia systems interconnected by a prototype integrated network. Although the emphasis has been on the needs of business, most of the ideas are equally applicable to other domains.

1.2 Background

1.2.1 Personal Computers

Information processing in the office has been transformed by the recent adoption of personal computers (PCs) for a wide variety of applications. The growth in the use of the PC has partly been due to continued advances in VLSI technology that have forced down the cost of the hardware. In computing terms, a PC is a fairly simple system typically featuring 512/640 kbytes of RAM, a 20 Mbytes hard disk, a floppy disk drive and a colour monitor. The Microsoft MS-DOS operating system has become the de facto standard, being first offered by the IBM PC and later by a multitude of clones. Although this is a fairly simple single-threaded operating system, it has provided third-party application writers with a common base on which to build their products.
A personal computer provides a user with a range of facilities that have traditionally only been available from other sources. For example, word processing packages now give individuals access to the service originally provided by the central typing pool. Likewise, desktop publishing systems provide facilities once the sole preserve of the printing industry. The ability to process information using equipment on one's own desk often provides benefits in terms of efficiency and economy. In addition, PCs have provided the worker with a range of new tools for the office. For example, spreadsheet applications provide a useful means of representing and manipulating complicated accounting information.

The interactive performance of a PC is far superior to that of traditional time-sharing computer systems. Furthermore, a PC is often equipped with specialised hardware such as a mouse and a high resolution graphics adapter. Office applications can take advantage of these features in order to provide a more sophisticated working environment than that normally available from a central computing utility. In particular, much emphasis has been placed on the use of graphics and windows to enhance the user interface.

1.2.2 Professional Workstations

In parallel with the growth of PCs in the office, there has been a steady increase in the use of workstations by the scientific and engineering community. A workstation is characterised by a powerful processor, at least 1 Mbyte of RAM, a large hard disk and typically a high resolution screen (1000 pixels \( \times \) 1000 lines). It is a more powerful (and therefore more expensive) computing engine than the PC. Unix is the popular choice of operating system for such machines, giving a de facto standard multi-tasking environment served by a mature range of third-party software products. Workstations are popular tools for research and development, and they are usually associated with applications such as CAD, software development and simulation.

More recently, advances in technology have meant that the distinction between a workstation and a PC has become increasingly blurred. Manufacturers are now offering PCs based on fast 32-bit microprocessors that boast the performance of more expensive workstations. These machines can run Unix and other multi-tasking operating systems, and they promise to provide the office user with an increasingly rich range of applications.

1.2.3 Computer Communications

Although the ubiquitous PC is the answer to a great many local data processing problems, there remains the need to share information between common interest groups. This requirement includes access to communal data and to communication services such as electronic mail and fax. Such access is usually limited to the simple exchange of charac-
Background

ters - the PC is often relegated to the role of a 'dumb terminal' by running a terminal emulation package.

Local communication has traditionally been provided using fixed serial (RS-232) links to a central computing system. The last decade has seen an increase in the use of local area networks (LANs) to provide improved access over a limited geographical area. LANs are characterised by a high quality communication medium covering a small geographical area, thus providing high transmission speeds. They can be shared by a number of stations to provide a more flexible communications environment than that available using fixed links.

LANs also offer the benefits of resource sharing in terms of information and physical peripherals. This has encouraged the move towards distributed computing facilities and away from the provision of centralised computer installations. Distributed systems are made up of a number of co-operating subsystems, each of which can be dedicated to a particular task. Distributed computing systems offer benefits in terms of performance, reliability and flexibility.

The incorporation of LAN technology has been particularly rapid by the workstation manufacturers such as DEC, Sun and Apollo. Fairly mature network support is now available for workstation users, providing such facilities as remote terminal access and file sharing. The IEEE 802.3 Ethernet has become the de facto standard workstation LAN, and networking extensions to Unix are now commonplace.

More recently, PC users have begun to discover the benefits of networking. There are several products now available (eg from Novell) that support remote terminal access and that provide resource sharing such as file sharing and remote printing. The new generation of PC operating systems (eg Microsoft OS/2) includes multi-tasking capabilities and in-built network support.

There remains the problem of sharing information across a wide area - that is, over an area that is too large to be served by a single LAN or mesh of LANs. This includes the requirement for terminal access to remote systems and for communication facilities such as electronic mail and file transfer. These requirements can be met by using leased lines between sites, or by using dial-up links over the public telephone network or the packet-switched service (PSS).

If local communication is based on a LAN, then a convenient way to provide remote access is to provide a LAN gateway to the wide area network (WAN). This can then be shared by all the hosts on the LAN. A typical office LAN might therefore include a collection of PCs and shared peripherals, together with a gateway for access to the corporate mainframe.
1.3 Current Developments

1.3.1 Office Environment

Current trends suggest that there will be a continued convergence of workstation and PC functionality. It is likely that office users will begin to demand the performance and facilities traditionally confined to workstations. In particular, there will be a growing emphasis on LANs in particular and on networks in general for communications and resource sharing. The ad hoc PC LAN products of today are likely to be replaced by an ISO or de facto standard environment similar to that currently emerging for workstations.

The proliferation of local area networks will generate an increasing number of distributed computing systems for the office. Just as fileservers and shared printers are commonplace on workstation networks, such components will appear in greater numbers in the office. Furthermore, there will be a new generation of distributed applications that can better utilise the combined computing power of the office to provide more sophisticated services to the user.

The increasing use of computer networks will mean that there will be a decreasing distinction between a communication device and a computer. Indeed, this reflects the general convergence of telecommunication and computer technologies. A PC will be both a processing engine for running standalone applications, and a communication device supporting electronic mail, fax, and more sophisticated services involving new types of media.

1.3.2 Multimedia Applications

The media that can be processed using currently available office software include text, line graphics, bit-map graphics and images. Much effort from the world of standards is being directed towards ways of incorporating and representing such media within a standard office document architecture [Hunter 89]. This defines a standard means of describing the structure (both physical and logical) of a multimedia document, thereby allowing it to be processed by generic software tools.

The new generation of applications will be able to handle time-varying media such as voice and video. The word processor of today will be supplemented by the multimedia editor of tomorrow to enable the creation and modification of voice- and video-annotated documents. Real-time media place great demands on workstation hardware in terms of speed, storage capacity, etc. A workstation that provides such support is referred to here as a multimedia workstation.

The arguments proposed in this thesis are for a distributed office architecture based on a collection of multimedia components and general computing peripherals. The term 'workstation' is a logical concept that may refer to a set of discrete components.
1.3 Current Developments

1.3.3 Integrated LANs

There has been much recent interest in the transfer of multimedia documents over existing computer networks (eg [Thomas 85], [Naftah 86]). The characteristics of the network are not an issue in these experiments since there are no timing constraints involved in the transfer. If the document contains real-time media, it can be 'played back' in real-time after it has been delivered.

The new generation of LAN promises to provide simultaneous support for traditional data traffic, distributed computing facilities and real-time multimedia communications. This requires that the LAN must be able to make minimum bandwidth guarantees for real-time media such as voice and video. Other necessary characteristics of the LAN are that media access times and transit delays must be short. If these claims can be realised, such a LAN meets the requirements of an integrated services network, and is referred to here as an integrated LAN. The slotted ring LANs described in this thesis have some of the properties of an integrated LAN.

1.3.4 Wide Area Networks

In the wide area context, the current installation of the integrated services digital network (ISDN) offers some scope for integrated services. The ISDN provides digital access to customer premises, and it can be used for the transfer of various types of traffic. The CCITT basic rate interface offers two (circuit-switched) 64 kbps B-channels and one (packet-switched) 16 kbps D-channel (the so called '2B + D' interface). The primary rate interface offers thirty 64 kbps B-channels and one 64 kbps D-channel.

The synchronous service (based on circuit-switching) is particularly suited to constant bit rate (CBR) traffic, such as PCM voice, since it provides fixed bandwidth guarantees. The (limited) asynchronous service (based on packet-switching) can be used by variable bit rate (VBR) traffic such as computer data. Note that the bandwidth available from the current ISDN is insufficient for real-time video without the use of compression techniques, the implementation of which are still expensive.

Recent research into asynchronous transfer mode (ATM) techniques [Vorstermans 88] suggests that ATM could be the basis for the next generation of broadband ISDNs. In ATM networks, the hybrid solution to the problem of integrated support adopted by the current ISDN is be replaced by an entirely packet-switched paradigm based on the use of a small, fixed sized 'cell'. The delay and jitter characteristics of the network approach those of a circuit-switched network. The adoption of ATM techniques could eliminate the distinction between telephony networks and computer data networks - instead, a single network should be able to meet the conflicting requirements of all types of traffic. The wide area network described in this thesis can be considered as a prototype ATM network.
1.3.5 User Level Integration

Given suitable network support, there is still the requirement for the integration of media at the user level. That is, high level support is necessary in order to be able to control, co-ordinate and present these media in a coherent fashion. An office that meets the requirement of user level integration is termed an integrated office. An office that has multimedia capabilities but does not provide user level integration is termed a multimedia office. The experimental offices described in this thesis fall into the latter category. However, the integrated office is clearly the desired objective, and the design of such an office is a major theme of this thesis.

The requirement for the integration of media might seem to be at odds with the proposals for a distributed office architecture. However, the argument in this thesis is that such a distributed architecture is the only realistic means of supporting the stringent demands of real-time media. Orchestration services are therefore required to co-ordinate the multimedia components making up the office. This is a complicated subject which has yet to see major research effort.

1.4 Aims

The purpose of this research was to investigate the provision of multimedia services in an office environment. Particular emphasis has been placed on the requirements of real-time services involving voice and video. The exchange of multimedia information between office users was seen as an essential requirement. Most of the work has been directed at the network and systems levels, although some consideration has been given to user level concerns.

The work has been conducted in an experimental multimedia office based on a local area network. A number of these offices have been interlinked using an ATM-like wide area network. The experimental office is based on a distributed architecture involving a collection of multimedia components. The control and co-ordination of such components has been a central theme throughout this research.

The overall intention was that the practical experience gained from these experiments would generate a set of design principles that would be applicable to future office developments. The desire for user level integration of a variety of media is seen as the ultimate goal, and a number of issues face the designer of an integrated office. It is hoped that the lessons learned from this research will go some way towards indicating a suitable design.
1.5 Context

The research work described here was conducted as part of the Unison Project [Clark 86]. Unison was a two million pounds collaborative research project within the Infrastructure and Communications Programme of the UK Alvey Directorate. The participants were Acorn Computers, the University of Cambridge Computer Laboratory, Loughborough University of Technology, Logica PLC, and the SERC Rutherford Appleton Laboratory. Considerable support was also provided by the Cambridge Olivetti Research Laboratory.

The objectives of Unison were to develop a network architecture suited to the expected communication needs of the ‘office of the future’, and to investigate the provision of multimedia services and distributed computing facilities in such an office environment. These objectives include the requirement for access to a variety of services at remote sites. This has led to the development of a wide area network which provides an ATM-like overlay on top of primary rate ISDN.

In parallel with the development of the network architecture, much work has been done in the design and implementation of multimedia components. This applications work was the main concern of participants at Loughborough, and it was here that the work described in this thesis was conducted. Colleagues at Cambridge and the Rutherford Appleton Laboratory were primarily concerned with the design and implementation of the network architecture. Acorn Computers provided the workstation hardware and some useful applications input. Logica PLC provided the project management.

1.6 Structure of this Thesis

Chapter 2 introduces the reader to the issues facing the designer of an integrated office. These issues relate to the characteristics of the network on which the office is to be based, and to the architecture of the office itself.

Chapter 3 outlines the Unison network architecture. It gives the necessary background for a full understanding of the systems described in subsequent chapters. It is primarily intended for those readers who are not familiar with Unison.

Chapter 4 describes an experimental office based on the Cambridge Ring local area network. A prototype slow-scan video system is described in some detail together with the window-based user interface. This office meets some of the requirements of an integrated office in terms of support for multimedia services, but performance is limited by the communications infrastructure and by the applications hardware.
Chapter 5 describes an experimental office based on the Cambridge Fast Ring. The exploitation of new hardware for the office components together with the improved communications infrastructure have provided a better basis for the provision of multimedia services involving real-time traffic. A video service which displays pictures within workstation windows is described in some detail, together with a high resolution still image service based on a document scanner.

Chapter 6 discusses the problems of control and access to services in a distributed office. A model is presented for the management of control, and its implementation is described for the Fast Ring office. This model also provides a convenient means of accessing a service in a distributed environment based on the name of the user with whom the service can be associated.

Chapter 7 presents the conclusions of this research. It also suggests the direction for future work in this area.

1.7 Extent of Collaboration

Chapter 2 is a general consideration of the requirements of an integrated office. It draws on material collected from a variety of sources, and from my own experience with the Unison Project. It has been compiled entirely by myself.

Chapter 3 describes the Unison network infrastructure, and is included to provide the background for the rest of the thesis. The installation, configuration and maintenance of the network infrastructure at Loughborough was entirely my responsibility.

Chapters 4 and 5 include a brief overview of the components and services making up the Cambridge Ring and Cambridge Fast Ring multimedia offices. This includes work done by a great many people, and references are made to more detailed accounts when necessary. However, the bulk of these chapters describes specific applications that are all my own work.

Chapter 6 discusses the problem of component control in a distributed office, and is based on 'hands on' experience gained with the experimental offices. The design and implementation of the model described in this chapter is entirely my own work.

Chapter 7 presents some personal observations and conclusions.
Chapter 2

Design Considerations for an Integrated Office

2.1 Introduction

An integrated office will meet all the communication and computing requirements of the next generation of office worker. Such an office will include facilities to process and distribute various types of media. Simultaneous support will be provided for existing telephony traffic, new types of real-time traffic such as video, and distributed computing transactions all within a common office architecture. The distinction between computer data and real-time traffic will become increasingly blurred as a single communications paradigm emerges involving all types of media. This environment will enable the development of new styles of multimedia applications and teleconferencing activities.

A basic argument presented in this thesis is that the integrated office will be based on some type of local area network (LAN). This is already the favoured networking technology for interconnecting computer systems over a limited geographical area. Furthermore, improvements in LAN technology suggest that it may be possible for a single LAN to meet the diverse requirements of the various types of traffic that might be expected in such an office. Such traffic includes real-time multimedia data such as voice and video, and distributed computing interactions between cooperating systems. This chapter outlines the general characteristics of a LAN, and highlights the specific properties that are required for the support of fully integrated services.

The interconnection of offices at remote sites in such a way as to provide wide area support for integrated office services requires that the performance characteristics of the WAN match those of the LAN. This prompts the adoption of a rather different approach to WAN design than that which has been normally taken in the past. This chapter goes on to outline this new way of thinking and to consider the necessary characteristics of such a network.

A fundamental characteristic of an integrated office is that integration must be provided at the user level. This demands that the office infrastructure itself must be capable of handling the various media in a constructive and coherent fashion. This chapter presents the arguments for a distributed architecture as the basis for the integrated office. The implications for the office infrastructure are discussed, concentrating on the implementation of the multimedia workstation. Various issues relating to the problem of integration in a distributed environment are also considered.
2.2 Design of the Local Area Network

2.2.1 Introduction

The traditional view of a computing system is one in which a collection of hardware is put under the control of a single operating system which then offers a variety of facilities to the user. Invariably, there are conflicting requirements facing the designer of such a system in terms of maximising the use of resources while maintaining an adequate level of service. This is particularly true for multi-user or multi-tasking systems which need to compromise between interactive response and overall throughput. The net result is often a system which handles a large number of tasks, but none of them particularly well. An attractive proposition is to build a number of dedicated subsystems which are designed to perform a small number of tasks efficiently. This can have the effect of reducing competition for resources and improving overall efficiency.

The introduction of LANs during the early 1970s presented the possibility of separating out individual tasks within a system and assigning them to separate processors. The LAN provides a high performance local communication medium for the interlinking of such systems. This development prompted much interest in the use of distributed processing techniques to provide a more efficient working environment based on a heterogeneous collection of components. Such an approach was adopted by the designers of the Cambridge Distributed Computing System in which a user is assigned a machine from a bank of processors according to his particular requirements (memory size, system type, etc). The processor bank machines draw upon various network services such as those provided by filing machines and printers. Refer to [Needham 82] for details.

Apart from supporting transaction-style interactions between distributed systems, LANs also offer the potential of carrying real-time traffic such as voice and video. LANs are also able to meet the (less stringent) requirements of traditional connection-oriented data communication services such as electronic mail and file transfer. As such, they are strong candidates for providing the communications backbone for the integrated office.

2.2.2 General Properties of LANs

LANs have a number of general properties that make them useful for interconnecting hosts within an office:

- Due to the high quality of the communications medium (often co-axial cable or optical fibre), transmission rates are fairly high ranging from hundreds of kilobits per second to tens of megabits per second.
2.2 Design of the Local Area Network

- Furthermore, bit transit delays are small compared with the average packet transmission time. This is a consequence of the short cable lengths and the lack of store-and-forward or switching nodes.

- Access times are also short allowing a station to access the network within a few average packet transmission times. This provides high connectivity between hosts.

- Another fundamental property is that bit error rates are low, usually less than 1 in $10^{10}$.

- Sequentially of packets is guaranteed. Although packets may be lost, those that are delivered will be in order.

It is important to distinguish between the network properties of 'high bandwidth' and 'low delay'. The bandwidth of a physical circuit is a measure of the information capacity of the circuit, and determines the maximum transmission rate for that circuit. The delay is a function of the bit rate and propagation delay of the medium (both of which are related to bandwidth), length of the link, and queuing delay in network nodes [Leslie 83]. A satellite channel is a good example of a high bandwidth link which suffers from long delays due to the large distances involved in transmission.

However, the concept of 'bandwidth' on a packet network is rather more difficult to define. The bandwidth of a virtual circuit is used here to describe the information capacity of the circuit, and relates to the availability of packets to transport that information. Delays are due to propagation delays mentioned above, switching delays in network nodes, and packetisation delays.

In both cases, insufficient bandwidth for a given stream results in an extra delay due to queuing at the transmitter. In many cases, this is the most significant factor meaning that delay can often be considered to be inversely proportional to bandwidth.

2.2.3 An Integrated Services LAN

2.2.3.1 The Spectrum of Traffic Types

In choosing a LAN to form the basis of an integrated office, other factors must be taken into consideration. The aim is to provide simultaneous support for traditional connection-oriented computing traffic as well as for distributed computing and real-time multimedia traffic. This requirement imposes conflicting requirements on the characteristics of the network. The basic design of the network must be flexible enough to support all these traffic types.
For example, packet voice makes particularly stringent demands of the network. In particular, bandwidth guarantees must be made for each call to reflect the time-sensitive nature of the traffic. Constant bit rate (CBR) voice, as generated by PCM, requires a constant bandwidth connection; variable bit rate (VBR) voice requires at least a minimum bandwidth guarantee. Packet delays must be very small (less than 2 ms is a target figure [Ades 87]); voice samples that arrive too late are of no use to a receiver. However, some degree of packet loss or corruption of data can usually be tolerated, since this merely degrades the service. Furthermore, high connectivity is not a requirement, since a voice connection is likely to last for several minutes.

Packet video has similar requirements to voice in terms of bandwidth guarantees and packet delays. However, whereas the bandwidth requirement of a voice stream is typically 64 kbps, a raw (i.e., non-compressed) real-time video stream requires about 140 Mbps. In practice, this means that real-time packet video awaits further advances in technology in terms of high bandwidth communications media and fast packet switches. [It is interesting to note that although raw video is fairly resilient to packet loss, compressed video can be very sensitive. As for voice, raw video contains a significant amount of redundant information making it insensitive to the occasional loss of data. On the other hand, compressed video contains far less redundancy making it rather sensitive to errors. Experience from the MAGNET project [Lazar 87] suggests that video places greater demands on a network in terms of both bandwidth and reliability than voice.]

Traditional connection-oriented computer traffic (such as file transfers and remote terminal access) requires a completely reliable communication path. There is no minimum bandwidth requirement, but this sort of bursty (VBR) traffic can often make use of as much bandwidth as is available for very short periods of time. High connectivity is not a requirement, since these sort of connection-oriented services typically last for at least a few seconds. Packet delays can usually be tolerated.

Distributed computing traffic makes yet another set of demands of the network. High connectivity is the major requirement, since many peer transactions are performed per second. Once again, high reliability is important, although neither bandwidth guarantees nor small packet delays are essential.

As explained above, most LANs meet the requirements of high connectivity, high bandwidth, low delays and low error rates. The choice of a suitable protocol suite can therefore provide support for the connection-oriented data transfer services and the connectionless distributed computing transactions identified above. A much more difficult requirement is that of providing minimum bandwidth guarantees for real-time traffic.
2.2 Design of the Local Area Network

2.2.3.2 The Bandwidth Allocation Problem

To support multimedia services, a network must be able to provide bandwidth guarantees for real-time traffic. This may include providing constant bandwidth connections for CBR traffic. A number of schemes have been designed to meet this requirement.

The MAGNET integrated LAN (ILAN) [Lazar 85] provides dynamic bandwidth allocation at the media access level under the control of a central expert system and local workstation bus controllers. It is based on an active slotted ring design to implement flexible sequential type access mechanisms. Global and local network utilisation patterns are monitored by the expert system and local bus controllers respectively. These parameters are recorded in a database and used to dynamically adapt the media access mechanism according to user requirements. The performance of this system awaits evaluation, but it is obvious that the basic infrastructure is complicated and expensive.

The Orwell Ring [Falconer 85] is a destination delete slotted ring that has been designed to support integrated services and switching applications. Individual stations monitor the load on the ring and use call blocking to prevent new circuits from causing overload. In this way, bandwidth allocated to existing circuits is guaranteed. Furthermore, individual stations maintain separate queues for different priority traffic. This ensures that real-time traffic is allocated bandwidth in preference to background data traffic. Although this is a more simple scheme than that used in MAGNET, its implementation is more complicated than that for a basic slotted ring.

Slotted rings have the inherent property that bandwidth is equally divided between hosts regardless of the load on the ring. This is a simple consequence of the fact that a station is not allowed to re-use a returning slot, thereby freeing it for use by the next station downstream. Furthermore, choosing a small packet size means that bandwidth sharing occurs at a fine granularity. These two properties are especially useful for voice. Assuming that the ring is not saturated, sufficient bandwidth will be available to support a voice connection. Furthermore, the small packet size keeps packetisation delays to a minimum. In addition, the number of samples in such a packet is sufficiently small to mean that occasional packet loss does not seriously degrade the service.

An economic implementation of a fully integrated LAN is still awaited. The minimum bandwidth requirement for real-time traffic and the high bandwidth requirement for real-time video are particularly difficult to meet using current technology. Furthermore, the design of a high performance network interface for video is still a major obstacle.
2.3 Design of the Wide Area Network

There are two basic styles of wide area communication network that are currently in use. Telephone networks use circuit-switching techniques to provide constant bandwidth connections for real-time voice. On the other hand, digital data networks use packet-switching techniques to accommodate the variable bandwidth requirements of bursty computer traffic. Telephone networks are analogue, digital, or a mixture of the two. Analogue telephone networks are characterised by long call set-up delays and by rather noisy (or error prone) circuits. Packet-switching techniques maximise network utilisation but are unable to offer constant bandwidth guarantees. These two styles of network represent the different approaches to communications taken by the telephony and computer industries.

Several packet-switched network architectures (eg DARPA's ARPAnet, IBM's SNA, CCITT's X25) were developed during the 1970s reflecting the contemporary requirements of computer communications. In particular, the low quality of the communication fabric meant that reliability was the major concern. This contrasts with the LAN environment which is based on an essentially error free high speed communication fabric. Heavyweight virtual circuits were the standard offering, with flow control and error recovery being provided on a hop-by-hop basis. Window flow control and block retransmission schemes were commonplace on such networks. Such an approach sacrifices speed and simplicity for the benefit of reliability.

The concept of a single communication medium meeting the conflicting requirements of various types of traffic (particularly voice and data) is currently receiving much popular attention under the banner of ISDN. The ISDN carrier services are based on high quality links (often using fibre optic technology) that are characterised by significantly lower bit-error rates than those associated with the conventional networks of the 1970s. Furthermore, the bandwidth that is being offered to PTT subscribers (2 Mbps primary rate) is approaching that available from the current generation of local area networks.

The problem with ISDN is that it is based on a primarily circuit-switched model - a consequence of its background in the telephony industry. Although ISDN offers the limited integration of voice and data, it cannot meet the requirements of a fully integrated network in terms of high connectivity for transaction-oriented services. Furthermore, a circuit-switched network makes inefficient use of network resources in general and network bandwidth in particular. The shortcomings of such an approach are explained more fully in [Turner 86].

There has been much recent interest in the use of asynchronous transfer mode (ATM) techniques as the basis for multimedia services. ATM aims to provide the bandwidth utilisation and connectivity benefits of packet-switching with the low delay and jitter characteristics of circuit-switching. A uniform transmission mechanism is provided based on the use of a small fixed sized packet (or cell). The network provides a
2.4 The Office Architecture

simple cell relaying service without flow control or error recovery. This is exactly the level of service required by real-time services such as voice and video; call blocking is the only realistic method of flow control for real-time traffic. Hosts requiring a completely reliable connection can engage in suitable protocols on an end-to-end and application dependent basis.

One challenge to the ATM designer is in the implementation of switching components that can meet the delay and jitter constraints of real-time traffic. Other problems relate to congestion control and bandwidth management. Out-of-band management techniques are favoured in order to maximise the performance of in-band components. This is because the use of out-of-band techniques obviates the need for in-band components to intercept management cells during the data transfer phase.

Much thought is now being directed towards the provision of a broadband ISDN (B-ISDN), and ATM is a favoured technology for such networks. However, the public networks that will be available for the foreseeable future will be based on ISDN circuit-switched technology. Before the arrival of native ATM WANs, one possibility is to superimpose an ATM overlay on top of the synchronous transfer mode (STM) service provided by the circuit-switched ISDN. This has been the approach adopted by the Unison Project, and the ATM aspects of the Unison network are described in [Tennenhouse 89]. Note that the use of an ISDN substrate for the provision of an ATM network raises the problem of the allocation of ISDN resources to meet ATM connectivity requirements. This is a complicated problem, and is discussed in relation to the Unison network in [Harita 89].

2.4 The Office Architecture

2.4.1 Introduction

The distinguishing feature of an integrated office is that the various media must be controlled and co-ordinated in a coherent fashion in order to provide user level integration. This requires an appropriate office infrastructure over and above the sort of communication infrastructure identified above. This section proposes a distributed architecture for such an office. The arguments for such an approach are presented, together with the implications for the office infrastructure. An attempt is made to resolve the apparent contradiction between the desire for integrated services and the proposal for a distributed architecture.
2.4 The Office Architecture

2.4.2 Arguments for a Distributed Architecture

2.4.2.1 Performance

A major incentive for employing distributed techniques in the office is that of performance. It is often advantageous to divide a complicated system into a number of co-operating subsystems. Each subsystem can be customised to provide one function well, rather than many functions badly. This is exactly the argument used to justify conventional distributed computing systems, but the argument is particularly valid when considering the design of a system to handle many types of media. The demands placed on resources by real-time media are unlikely to be met by a single system.

A number of issues are relevant to the performance of a multimedia system:

- network interface performance
- host processor bandwidth
- network driver design.

The performance of network interfaces is a problem common to many types of network. The maximum throughput between two machines is often limited more by the performance of the network interface than by the available network bandwidth. For example, although the maximum point-to-point bandwidth on the Cambridge Ring LAN is quite modest (about 1.3 Mbps), in practice the maximum achievable point-to-point throughput is significantly less than this (about 800 kbps) due to limitations in network interface performance. A single network interface would be stretched to simultaneously support one or more voice streams in parallel with, for example, a video stream or a file transfer.

The new generation of faster microprocessors and bespoke bit-slice implementations has eased this situation somewhat, but it often remains prudent to provide independent network interfaces for bandwidth-hungry or time-critical media such as voice and video.

Most host processors are simply not powerful enough to cope with a number of parallel data streams, especially of different media. A processor that is being interrupted every few milliseconds by a real-time voice stream is unlikely to be able to run a complex window-based application simultaneously, let alone handle a high bandwidth video stream. It seems sensible to dedicate separate processors to handle the different media, especially when real-time data streams are involved.

A further argument for this physical separation of data streams is that the transmission requirements for the different media and traffic types vary enormously, and this has implications for the network software that often extend right down to the network device driver. For example, traditional data traffic such as computer file transfers and electronic
mail requires heavyweight protocols where the emphasis is on reliability and not speed. At the other end of the spectrum, digital voice traffic requires lightweight protocols where the emphasis is on speed and not reliability. These requirements are so diverse that the network software handling the two media needs to be quite different.

2.4.2.2 Reliability

Distributed systems can offer the benefit of improved reliability. This is because the constituent components are, by definition, more simple than the entire system, and therefore tend to be less prone to failure. Furthermore, the failure of a single component is unlikely to be catastrophic for the rest of the system. An added advantage is that replicated components can be easily added thus making the system more fault tolerant by introducing redundancy. When a faulty component is detected, a standby component can take its place.

2.4.2.3 Extensibility

The modular nature of distributed systems makes them relatively easy to expand or reconfigure. For example, an extra fileserver can be added to improve overall disk access times.

2.4.2.4 Cost

A distributed approach offers the advantage of efficiency of implementation, and this can provide economic benefits. Furthermore, a distributed architecture is better suited to standardisation than a closed architecture. That is, distributed systems lend themselves to the sort of 'open' environment in which a fileserver can be bought from one supplier and a processor from another. The implementation of the particular subsystem is immaterial, as long as it conforms to a standard architecture. A standard environment would provide costs benefits to the user as a result of competition between suppliers.

2.4.3 Integration in a Distributed Environment

The requirement for user level integration seems to be at odds with the desire for a distributed office architecture. The aim is to present a variety of media to the user in a coherent fashion, and this is made more complicated if each of the media is handled by a separate component. A common architecture is required for the representation and manipulation of multimedia documents in which the various media may reside on, or originate from, separate components. This includes the need for synchronisation between related streams when 'playing back', say, a voice annotated message (see section 2.4.8 below). Furthermore, office users need to communicate using a variety of
media, and this involves the same problems of control and synchronisation. The design of such an architecture is an on-going topic of research [ANSA 89].

Support for a common (distributed) operating system by each of the office components would be one possibility for providing the desired level of integration. However, this is at odds with the aim of keeping the implementation of each component as simple as possible (see section 2.4.4 below). An alternative solution is to provide co-ordination, or 'orchestration', servers which add value to the basic functionality of individual components by combining their services in an integrated fashion. Such an approach fits in with the distributed architecture of the office, and permits the implementation of each component to be as simple as possible. Once again, the design of orchestration servers is a topic for further research.

2.4.4 Location of Functionality

Another design principle proposed here is that the functionality of each of the office components should be kept to a minimum. Instead, intelligence should reside in a separate server (or servers) dedicated to the task of providing extra functionality to the basic components.

The reasons for this are to do with cost and flexibility. If a component can be made simple, then its cost will be low. This is a particularly strong argument when there are likely to be a great many such components in the office (such as a telephone). Furthermore, simplicity usually correlates well with reliability. Benefits in terms of flexibility are possible because enhancing the overall service should simply be a matter of upgrading the dedicated server rather than altering the individual components.

This is exactly the approach advocated in the ISLAND project [Ades 87]. In this project, a distributed PABX is presented based on a local area network. The functionality of the individual telephones is kept to a minimum, but the facilities of a PABX (in terms of call redirection, etc) are provided by means of a dedicated server. New features can then be added by simply upgrading the software running on the server.

2.4.5 Location of Control

Another question related to the design of the office is the location of control. Given the sort of distributed workstation architecture proposed above, management of the control of the various components becomes an important issue. With a closely coupled configuration, the operating system provides the host processor with full control over each component. However, with a loosely coupled system, an entity (usually in the form of a workstation) must take responsibility for the control of the other components over the network.
In terms of control, components may be classified as being dependent, independent or a combination of the two. An independent component is one for which no external control entity is required for its use. An example of such a system might be a telephone with a handset; the handset can be used to control the telephone without outside intervention from a separate entity. On the other hand, a dependent component is one which relies on an external control entity for its use (similar in concept to a "server"). An example of such a system might be a telephone without a handset; this time, control information relating to call set-up and call termination must be supplied by the controller. Dependent components are characterised by having a well defined network control interface that can be exercised by the control entity. The third category of components includes those which are both dependent and independent. Such a device would be a telephone with a handset and a network control interface.

Independent components have the advantage that their operation does not depend on the existence of a third party (i.e., one can still use the telephone when the workstation is down); the disadvantage is that there is no scope for remote control over the network (i.e., one is unable to set up a telephone call from the workstation). The issue is that of location of control; should control be the responsibility of the component itself, or should control be distributed and possibly remote from the component. The argument supported in this thesis is for distributed control. This provides maximum flexibility in the design and location of the control mechanisms, with the added bonus that complexity can be removed from the component itself and placed elsewhere on the network. This keeps the implementation of the components themselves as simple as possible.

Note that the intention here is merely to supplement (rather than to replace) existing control mechanisms where they already exist. That is, workstation control mechanisms can exist alongside current control procedures when required. For example, there is no reason why a telephone should not possess both a control interface for workstation control and a handset for local control. The aim is to improve the user interface to office components by using the workstation (and suitable graphics, etc.), rather than to preclude their conventional use.

Accepting the arguments for distributed control raises the complicated question of the management of control. In practice, a workstation must take responsibility for its peripheral components by acting as the control entity. This implies the need for a binding between a workstation and the components that it controls. The whole question of the management of control is addressed more fully later in this thesis.

2.4.6 Design of the Workstation

The integrated office will contain one or more multimedia workstations to provide access to the full range of multimedia applications that will be available to the user. The term 'multimedia workstation' is used here to define a logical entity that is capable of han-
2.4 The Office Architecture

dling a number of media such as voice, video and text. The implementation of such an entity may be by means of a single system capable of handling all types of media, or by means of a collection of co-operating components each handling a particular type of medium. In practice, a range of possibilities exists between these two extremes categorised by the degree of coupling between the subsystems. A closely coupled system is defined here as being one in which the subsystems are interlinked by an internal processor bus. On the other hand, a loosely coupled system is one in which the subsystems are interlinked by the network alone.

A closely coupled workstation comprises a collection of processors and peripherals all based on a common bus architecture. Control and data share the same bus, and the system runs a real-time operating system. A diagram of a simple closely coupled workstation is shown in Figure 2.1. There are advantages with such an approach in terms of ease of integration at the user level, and several projects have adopted this design. For example, with a system in which a frame grabber is a processor bus peripheral, it is relatively straightforward to integrate video into window-based video applications. However, such a design lacks the benefits of a distributed approach.

An example of such a workstation is the EDDY workstation used by the MAGNET project [Lazar 87]. Voice, video and graphics are handled by a single unit under the control of a real-time executive. Although the internal design of the system is based on loosely coupled processors, the workstation offers few of the benefits of a system based on distributed network components.

![Figure 2.1: A Closely Coupled Workstation](image-url)
A loosely coupled workstation comprises a collection of multimedia components with the local area network as the only means of communication (Figure 2.2). This is consistent with the desire for a distributed office architecture, but there are problems in meeting the requirement for user level integration. For example, if video is handled by one component and graphics by another, it is difficult to combine the two on a single display. A possible solution to this problem is to provide analogue mixing of related streams for display on a single screen. This means that the digital data paths remain independent, but the analogue signals are combined for display.

A distributed approach to workstation design was taken by the PARC Etherphone project [Swinehart 83]. Rather than voice hardware and software being added to individual workstations, the Etherphone is a network peripheral. This provides voice capabilities to users over the full range of available workstations. Furthermore, the 'server' approach avoids the need for a new voice implementation for each type of workstation, and solves the performance problem of a single system handling voice alongside other forms of data. The full environment includes a telephone control server and a voice file server all based on this same distributed architecture.

In a loosely coupled system, there needs to be a rapid exchange of control information between components to enable co-ordination and synchronisation of related streams. In practice, this might not be possible over the network. A 'hybrid workstation' consists of

Figure 2.2: A Loosely Coupled Workstation
loosely coupled and closely coupled subsystems in which the closely coupled subsystems handle related streams (Figure 2.3). Such an approach represents a compromise between the two extremes identified above. It provides the benefits of a distributed design up to the point where practical constraints make this impossible.

An example of this hybrid approach has been taken by the Olivetti Pandora project [Hopper 88]. In this system, voice and video are handled by a single box which is a network peripheral. Analogue mixing is used to display video in windows controlled by the workstation. [For this purpose, the workstation and Pandora box are in fact coupled by an independent control link, although the data paths remain separate.]

2.4.7 Data Replication

The integrated office is likely to require some form of broadcasting or multicasting capability. This is useful for bulk data distribution and teleconferencing activities; the transmitter is spared the burden of replication at source, and this can have major performance benefits. A broadcast is a transmission to all receivers on a network, while a multicast is a transmission to selected receivers. Broadcast mechanisms are usually easier to implement, but they generally result in large amounts of redundant traffic. Broadcasts over WANs can result in an embarrassing amount of traffic which can swamp the network. However, some LANs have an inherent broadcast mechanism, for example by
reserving one particular port number for broadcast traffic. Clearly, the effects of broadcasting over a local area are less severe than over a wider area.

Wide area networks require some form of multicasting mechanism to obviate the need for source replication. Such a capability can also have benefits in terms of network bandwidth utilisation and reduced loads on bridging components. A possible design might comprise a collection of interworking multicast servers that co-operate to provide a wide area multicasting capability. Such a system requires knowledge of the network topology in order to realise potential bandwidth savings between sites. Another consideration is that the performance of the system must match that of the network and bridging components in order for it not to become a bottleneck for applications such as teleconferencing. The problem of how a client interacts with the system is another issue to be resolved. The general problem of multicasting is touched upon later in this thesis.

2.4.8 Synchronisation

The new generation of applications involving multiple time-varying media will require some form of mechanism to synchronise logically related streams. For example, a talking picture (or movie) requires synchronisation of voice and video streams.

One solution is by the use of temporal markers to control the playback of related streams. For example, for every marker in the video stream a receiver might block the playback of video until the corresponding marker is detected in the voice stream. If the granularity of the inserted markers is suitably fine and if suitable buffering is used, this technique can achieve the desired result. However, the problem is that if the voice and video streams are handled by independent components (as they would be using the distributed workstation architecture proposed above), these components have to interact first to insert the markers and then to decode them properly to achieve synchronisation. This is a non-trivial problem which appears to have no simple solution.

Another mechanism is based on the use of time-stamping. If voice and video packets are stamped with a common time reference, the receivers can use these time stamps to control playback. Synchronisation is then automatically achieved since the time stamps preserve the temporal relationship between the two streams. Although this scheme works even with distributed components, the problem is one of providing a network-wide absolute time reference. The problem of clock distribution across distributed systems has been much discussed in the literature [Marzullo 83].

A third solution involves the use of a collection of interworking synchronisation servers (or 'sync servers') that provide a network wide synchronisation service. Related streams would be connected to the local sync server which then inserts appropriate sync markers. The synchronised streams would be transmitted to a peer sync server, where they would then be decoded and passed on to the peer devices. The mechanism as-
sumes that there will be no loss of synchronisation during transmission between components and the local sync server; ie related streams are only expected to lose synchronisation across the wide area network (while passing through bridging components). As with the use of multicast servers described above, the main problems are to do with performance and the way that a client interacts with such a service.

2.5 Summary

This chapter has considered some of the issues facing the designer of an integrated office. These relate to the properties of the local area network on which such an office is likely to be based, the properties of the wide area network required to provide integrated inter-office communications, and the properties of the office infrastructure necessary to provide integration of services at the user level. A slotted ring technology is considered a strong candidate for forming the basis of the office LAN due to its relative simplicity and its inherent properties that make it suitable for integrated services. An ATM-based wide area network is proposed for interconnecting office LANs in the belief that this style of network has much to offer in terms of multi-service support. A distributed workstation architecture is favoured based on a collection of co-operating multimedia components that have been designed to handle one type of medium. A distributed control environment has been suggested based on the remote control of office components from the workstation.
Chapter 3
The Network Architecture

3.1 Introduction

This chapter gives an overview of the network architecture. It describes the Unison wide area architecture that was developed to interlink geographically dispersed local area networks (also referred to here as 'distribution networks'). It also describes the architecture of the two types of local area networks that were used during the project, namely the Cambridge Ring (CR) and the Cambridge Fast Ring (CFR).

Note that the Unison network architecture is quite independent of the architectures of the particular distribution networks. However, the Unison architecture has been designed to extend the features of a local area network with as much transparency as possible. A consequence of this is that a client's view of the network is one in which his local area network has been extended to include services at remote sites. The mechanisms for achieving this transparency are discussed for each of the distribution networks. A description of the client network architectures is presented in order to provide the necessary background for the following chapters on the Cambridge Ring and Cambridge Fast Ring offices.

This chapter takes a 'bottom-up' approach to the description of the network - the constituent parts are described first, followed by an overview of the complete architecture. The chapter begins by providing the background for the Unison work, and this is followed by an overview of the basic network configuration. The architecture of the two types of distribution networks is then described in some detail, followed by an overview of the terrestrial long-haul segment. The complete architecture is presented at the end of the chapter.

3.2 Background

There has been considerable interest during recent years in the use of local area networks (LANs) to support integrated services and distributed computing facilities. Such networks promise to meet the conflicting network performance requirements of a variety of traffic types. Distributed computing systems require high connectivity (i.e., the ability to access a large number of hosts within a very short period of time) and fast response times in order to support rapid transactions between peer applications. Real-time traffic requires bandwidth guarantees, fast access to the network, low delays and low jitter (the variation in delay).
One objective of the Unison Project was to build a wide area network with performance characteristics similar to those of a local area network. This is to allow applications that have traditionally been confined to the local area to be extended over a wide area. An additional aim was to investigate the problem of support for multi-service traffic including real-time and interactive data. The functionality required of such a multi-service network is quite different from that provided by existing internets. Data networks conceived in the 1970s use packet-switching techniques for the support of traditional computing facilities such as remote terminal access and file transfers. These techniques maximise network utilisation for the support of ‘bursty’ computer traffic, but they cannot meet the stringent requirements of real-time traffic identified above. On the other hand, networks that have been designed to support real-time traffic tend to be based on circuit-switching techniques and these cannot meet the connectivity requirements of distributed computing applications. The Unison architecture therefore represents a rather different approach to wide area network design.

Much of the Unison network architecture has been based on experience gained during the earlier Universe Project [Burren 89a]. This project used satellite technology (compared with terrestrial links used by Unison) to interlink geographically dispersed local area networks. The intention was to investigate the sort of services that might be provided over such a network particularly relating to distributed computing facilities. Experience with this project and other work on MAC level bridges (and routers) had already proved that LAN interconnection was quite feasible; the problem of multi-service support was an additional issue addressed by Unison.

Universe work was based on the use of CR LANs at the various sites. Much of this Universe infrastructure was inherited by Project Unison, and initial efforts were directed towards recreating the Universe architecture over Unison. Later on in the project, work migrated to the newer client CFR LANs and a new architecture was developed based on these networks.

3.3 Overview of the Network

The Unison network is based on the interconnection of distribution networks by an ISDN. Each site has a Unison exchange which switches intra-site and inter-site traffic between distribution networks of the same type. Any number of distribution networks may be connected to the site exchange, and remote access to peer distribution networks is via the ISDN. Note that there is no provision in the basic architecture for the protocol conversion necessary for the interconnection of distribution networks having different protocol stacks.

A simplified diagram of the network is given in Figure 3.1. The exchange is based on an internal CFR used for switching. The ramp provides access from an exchange
3.4 The Cambridge Ring Architecture

3.4.1 Introduction

This section outlines the architecture of the Cambridge Ring in terms of the hardware, the naming strategy and the adopted protocols. The LAN interconnection architecture presented here was developed during the Universe Project, full details of which can be found in [Burren 89a]. The Cambridge Ring itself is described in [Wilkes 79].

3.4.2 Basic Design

The Cambridge Ring is a 10 Mbps slotted ring around which a small number of slots continuously circulate. The basic unit of data transmission is the minipacket. A minipacket fits exactly into one slot; it contains source and destination addresses, 16 bits of user data, and a number of control bits. A station address is 8 bits long, giving a maximum of 256 unique addresses on a single ring. The structure of a minipacket is given in Figure 3.2.
A station wishing to transmit waits for an empty slot, marks it as full using one of the control bits, inserts 16 bits of data and sets the destination and source addresses. The receiver sets the two response bits to indicate the outcome of the transaction. When the slot returns to the transmitter, it is marked as empty so it can be used by the next station downstream. In addition, another slot must pass before this station retransmits. This allows the available bandwidth to be shared equally between active transmitters, preventing a single station from hogging the ring.

The two response bits are used to indicate whether or not the minipacket has been accepted. A receiver returns a 'busy' code to indicate that the station is occupied with a previous packet. This provides the lowest level of flow control. A receiver returns an 'unselected' code to indicate that it is not prepared to accept the minipacket from that particular station address. A receiver returns an 'accepted' code to indicate success.

The design of the ring includes an elaborate mechanism for maintaining the slot structure and identifying rogue stations. This is fully described in [Hopper 86a].

3.4.3 The Hardware

A connection to the Cambridge Ring comprises a repeater, a station and a host interface. The repeater permits physical access and regenerates signals on the ring. The station provides a standard logical interface to host devices usually based on a set of registers to permit transmission and reception. The host interface is a host dependent device that maps the general station interface to host processor i/o space. A diagram of a general ring interface is shown in Figure 3.3.
In practice, intelligent interface units have been provided for popular bus architectures such as Intel Multibus. These units often include a firmware implementation of a link layer protocol.

3.4.4 Protocols

3.4.4.1 Introduction

Minipacket level protocols are very simple and typically involve the transmitter backing off for a fixed number of ring cycles in the event of a 'busy' response from the receiver. This is automatically performed in hardware. Most conventional computer applications require a protocol immediately above this to provide a more useful service in terms of the size of the basic data unit and the addressing capability. The Basic Block Protocol (BBP) formats minipackets into blocks which are comparable to Ethernet or Token Ring packets.

Three basic types of protocol have been defined above the BBP. The Datagram Protocol specifies the minimum structure required to allow an unacknowledged block to be routed across the Universe network. The Datagram service was not used in the Unison office and it is not described further here. The Single Shot Protocol (SSP) is an exchange (or single-transaction) protocol in which a reply is returned in response to a single re-
3.4 The Cambridge Ring Architecture

The LVC Protocol is used to establish a *lightweight virtual circuit* [Leslie 83] which is a logical connection between hosts without flow control or error recovery. The lightweight virtual circuit (LVC) preserves one of the intrinsic properties of a LAN in that the order of blocks is preserved during transmission to a given receiver sub-address.

The various CR network protocols are summarised in [Adams 84].

3.4.4.2 Basic Block Protocol

A basic block consists of a sequence of minipackets containing header and routing information together with user data. Three types of basic block were used in the Unison office. A type 0 block comprises a header, a port number, a sequence of up to 1024 data minipackets, and a checksum. A type 1 block is the same except that the checksum is not used. A type 3 block comprises a header, a size minipacket (in bytes), a port number, a sequence of data minipackets, and a CRC checksum (see Figure 3.4). [A type 2 block fits into a single minipacket, but these are not supported by the network. They were used for such applications as packet voice on a single ring, but were predominantly used for interrupts].

![Figure 3.4: A Type 3 Basic Block](image)

Unqualified numbers denote bit lengths

A type 3 block is a more general version of a type 0 block (theoretically allowing more than 2 kbytes of user data) in which the ‘size’ field is a byte count instead of a word count. Type 0 and 3 blocks are the most commonly used by services requiring the maximum level of basic block support. A type 1 block is for the convenience of simple hosts for which the checksum calculation would be a significant overhead. In addition, services which can tolerate basic block errors (such as real-time voice) may prefer to use this type of block.

The header minipacket identifies the start of a block, and contains 10 bits of data to indicate the size (except type 3). The port minipacket provides a mechanism for sub-addressing within the receiver.

In line with the assumption that bit error rates are low (less than 1 in 10^{10}), the BBP includes no block acknowledgment scheme. Bridging components are designed to dis-
card blocks in the event of congestion; block loss caused in this way can only be detected by using a higher level protocol.

3.4.4.3 Single Shot Protocol

The single shot protocol provides a simple single-transaction service. An SSPREQ block is sent from the client to a service thereby initiating an operation and prompting the return of an SSPRPLY. The reply indicates the outcome of the operation. The format of these two Protocol Data Units (PDUs) is given in Figure 3.5.

The network address to which the SSPREQ is sent is determined by the nameserver lookup operation described below. The function code provides an additional mechanism for sub-addressing within a receiver. It is particularly useful for preserving buffer space in a simple service implementation by allowing the server to queue a single reception request on a single port number since incoming blocks can be filtered by function code. The receiver uses the reply port specified in the SSPREQ and the source station address passed up from the network interface to determine the network address for the SSPRPLY. Other SSP parameters (arguments and results) have meanings dependent on the particular transaction.

The SSP is commonly used for control purposes and for distributed computing applications. A service normally queues a reception request on a public port and waits to be contacted by a client. A client wishing to contact a service first allocates a private port for reception of the SSPRPLY, and a timer is started on this reception request (normal-
3.4 The Cambridge Ring Architecture

The client then obtains the network address of the service (by interaction with the nameserver) and initiates the operation by transmitting an SSPREQ to that address. If a reply is not received before the timeout has expired, the SSP transaction has failed. The problem of timeouts during nested transactions is addressed by the RPC mechanism discussed later in this chapter.

3.4.4.4 LVC Protocol

The LVC protocol is used to establish a lightweight virtual circuit without flow control or error recovery. To establish a connection, the client sends an OPEN block to a service thereby prompting any necessary action and the return of an OPENACK. As before, the reply indicates the outcome of the operation. The format of these two PDUs is given in Figure 3.6.

![OPEN/OPENACK Protocol Data Units](image)

Figure 3.6: OPEN/OPENACK Protocol Data Units

Unqualified numbers denote bit lengths

The only difference between the SSP and the OPEN/OPENACK exchange is that, for the latter, the receiver allocates a connection port for subsequent traffic on that connection. This reflects the fact that the result of a successful OPEN/OPENACK exchange is a lasting connection over which data flows until the connection decays through disuse.

Lightweight virtual circuits are used by real-time applications such as voice and video. As before, a service queues a reception request on a public port, and waits for incoming connection requests from clients. A connection is established by the successful exchange of OPEN and OPENACK blocks. The protocol that is then used is application dependent, and may include flow control and error recovery schemes. There is no explicit termination procedure; connections through bridges are simply left to time out.
3.4 The Cambridge Ring Architecture

3.4.5 Naming, Addressing and Routing in Universe

3.4.5.1 Naming and Addressing

For organisational convenience, the global Universe name space has been divided into a number of naming domains. The format of a full Universe name (either a machine or a service name) is \(<\text{domain}\>\*\<\text{localname}\>\). A unique domain name is allocated to each site, allowing local management of service and machine names within a domain. A domain may include one or more subnets.

The mapping between a Universe name and a Universe global address \(<\text{site}\>, <\text{subnet}\>, <\text{station}\>, <\text{port}\>) is performed by a nameserver located on every ring. A nameserver maintains a list of all the services available within a single domain. Nameservers interact to resolve names from foreign domains. To do this, each nameserver must have knowledge of every other naming domain and the address of a local bridging component giving access to each of these domains. The nameserver returns to a client a ring service address \(<\text{station}\>, <\text{port}\>, <\text{function}\>) which is required by the access protocols. The ring service address will refer to a bridge for a foreign service; the bridge maps the ring service address onto the appropriate global address.

Client access to the nameserver is by means of the SSP. A client fills in an SSPREQ with the service name to be resolved and the well known ring service address of the nameserver lookup service. The result of a nameserver lookup is then returned in the SSPRPLY. If the name has been successfully resolved, the results will include the ring service address of the specified service. Various flags are also returned giving miscellaneous information about the service such as the type of access protocol (SSP/LVC).

3.4.5.2 Routing

The nameserver is also a fundamental component in the routing mechanism. Nameservers interact with bridging components to set up paths across the network on behalf of a client. This operation is transparent to the client since a path is merely a ring service address on a bridge.

The Universe routing mechanism is based on the use of global addresses. The binding between a global address and a route is distributed among the nameserver and any bridging components through which the traffic flows. When a connection is required to a remote service, the nameserver makes an address insertion request to the appropriate bridging component. The bridge maps the global address of the specified service (supplied by the nameserver as a result of a successful lookup operation) to a local ring service address allocated by the bridge. The nameserver then returns this ring service address to the client. The bridge converts any initial connection block arriving at this address to a special initialisation block containing the global address supplied by the
nameserver. This block then traverses the network to the destination bridge where it is converted back to the original initial connection block. The return path has then been set up, and is simply a series of reply port mappings between bridges. The reply block then follows this path, and may set up a forward path (in the case of a virtual circuit set-up) as it returns.

3.5 The Cambridge Fast Ring Architecture

3.5.1 Introduction

This section outlines the architecture of the Cambridge Fast Ring distribution network as used during the Unison Project. The CFR is described in [Hopper 86b].

3.5.2 General Design

The CFR is a slotted ring developed from the earlier CR and is designed to work at speeds of up to 100 Mbps. The slot size has been increased to accommodate 32 bytes of user data. This increase reflects the higher clocking speed (and therefore the larger bit storage capacity) of the ring, and it is necessary to achieve reasonable point-to-point bandwidth over the faster ring. The exact choice of slot size is a compromise between providing a small number of rather large slots to maximise the available point-to-point bandwidth, and providing a large number of small slots to minimise slot wastage and provide rapid and equitable sharing of bandwidth. The address field has been extended to 16 bits for both source and destination addresses. The structure of a CFR slot is shown in Figure 3.7.

![Figure 3.7: Structure of the CFR Minipacket](image)

Unqualified numbers denote bit lengths

Chapter 3 - 34 - Network Architecture
The CFR has been designed with ease of ring interconnection in mind. A CFR node may be configured as a bridge between two adjacent rings. This is to provide support for a mesh of rings with a flat address space across the entire mesh. Hardware support, in the form of a 64 kbit lookup table, has been provided to aid the routing of minipackets through bridges. [This feature is exploited in the ramp.]

This same lookup table can be used in a different way by stations to provide a two-tier source selection scheme. As with the Cambridge Ring, a source selection register can be used to prevent reception from all locations on the ring or from all locations except that held in the register. However, an extension has been provided for the CFR in which the source selection register is set to accept minipackets from everywhere, but the lookup table (or 'hate list') is used to decide from which locations transmission is valid.

The CFR has been designed to support two media access strategies. The first is identical to that used by the CR in which a transmitter is not allowed to use a returning slot or the immediately following slot. This means that a transmitter can use one in every \( N + 2 \) slots, where \( N \) is the number of slots on the ring. This is called normal mode, and has the attraction of allowing bandwidth to be shared equally between stations regardless of the load on the ring. The second access strategy allows the returning slot to be used again by the same transmitter. This is called channel mode, and can be used to maximise throughput for bandwidth-hungry clients. A single bit in the CFR slot is used to identify the mode of transmission supported by that slot. A typical implementation will include a number of slots of each type to support both types of access. In this case, an automatic and transparent change to channel mode will take place if a transmitter is fast enough to make use of a channel slot.

Unlike the CR, there are no explicit response bits defined in a CFR slot. Instead, the last four bits of the CRC are used to denote two possible states. If a packet returns to a transmitter with the last four bits corrupted, this is taken to mean that re-transmission is not required. This may be because the packet has been accepted, or because the receiver is not prepared to accept a minipacket from that location. On the other hand, if the last four bits are intact, this indicates that re-transmission may be worthwhile. This would be the case if the packet is not received perhaps because the receiver is still busy, or maybe because a bit error has been detected as a result of the CRC check at the receiver.

Interpretation of the response is only a weak indication of whether or not a minipacket transmission has been successful. This is considered adequate since the design of the CFR has assumed the widespread adoption of ring meshes through which a response cannot pass.
The CRC is also used to convey control information from the transmitter to a receiver. This is required in channel mode to preserve the sequence of minipackets in the event of a minipacket transmission failure. When a transmitter discovers that it has replenished a slot for which the previous transmission had failed, it cancels the transmission by corrupting the CRC of the outgoing packet. The receiver then ignores the packet, but corrects the CRC indicating to the transmitter that the packet was rejected. The detection of a transmission failure in channel mode causes a reversion to normal mode. The source back-tracks to the original packet (double buffering is required for this purpose) before continuing transmission.

3.5.3 Hardware

A significant improvement over CR technology was the VLSI implementation of CFR station hardware. A serial CFR station comprises an ECL repeater chip and a CMOS station chip. The ECL chip provides the CMOS station chip with two 8-bit wide paths for outgoing and incoming data to and from the serial ring. The CMOS station chip provides a simple 8-bit register-based interface to the host, and an internal interface to the 64 kbit lookup table memory.

The transmit FIFO buffer appears as a single 8-bit register, and successive writes to it cause it to be filled and then transmitted. Likewise, the receive FIFO buffer is mapped onto a single 8-bit register, and the buffer may be emptied by successive reads. A selection of interrupts are available to signal such events as 'transmit buffer empty', 'transmission failed' (the minipacket is 'thrown on the ground' or TOGed), and 'receive buffer

![Figure 3.8: A General CFR Interface](image-url)
3.5 The Cambridge Fast Ring Architecture

full'. Refer to [Acorn 87a] for a full chip specification. A simplified diagram of CFR hardware is given in Figure 3.8.

Another benefit of the CFR chipset is that the per-minipacket CRC calculation is performed in hardware. On the CR, the CRC calculation was sometimes a major processor overhead. This feature of the CFR chipset therefore offers a significant improvement in performance.

3.5.4 Protocols

3.5.4.1 Introduction

As with the CR, the minipacket protocol defines a simple scheme for the transmission of minipackets between stations. This protocol is implemented by the CFR chipset. A minipacket transmission failure results in a fixed number of hardware retries separated by a suitable delay. This retry strategy is programmable in terms of the maximum number of retries and the interval between retransmissions. This provides a low level mechanism for a slow receiver to exert back pressure on a fast transmitter, although it is expensive in terms of network bandwidth.

Unlike the CR, a host is not returned definitive information on the outcome of a minipacket transmission. This reflects the fact that a minipacket response can only indicate the outcome of the first hop of what may well be a multi-hop route. A host is simply informed of whether or not a retransmission would be worthwhile, but a higher level protocol must be used to ensure correct end-to-end delivery.

All interactions over the CFR are based on the notion of an association between the hosts involved. An association is characterised by a couple of \(<\text{station}, \text{port}\>\) doublets that refer to each end of the association. As such, an association on the CFR is very similar in concept to an LVC on the CR.

A Unison Data Link (UDL) protocol has been defined to provide a link level service over an association. It provides a service similar to that provided by the BBP on the CR. This layer is sub-divided into M-Access and UDL sub-layers. The lower M-Access sub-layer is implemented by the CFR chip set as described above. The upper UDL sub-layer provides service users with the ability to exchange data blocks which are larger than a single CFR slot. In fact, segmentation and re-assembly account for most of the value added by the UDL service.

In practice, most applications require some form of higher level protocol to provide at least end-to-end flow control. In particular, a UDL service user has no guarantee that transmitted data is being properly received by the intended recipient if the route passes through a bridging component; in fact, the receiver may not even be listen-
ing. In Unison, Unity RPC represents such a UDL service user that was widely adopted by applications requiring a higher level of service. In particular, RPC provides the basis for most of the distributed computing facilities developed for the CFR office. On the other hand, the voice service does not require this level of support and it uses single minipacket UDL blocks directly.

3.5.4.2 Associations

An association is the fundamental requirement for the exchange of data between CFR hosts. Associations are established out-of-band by an independent network service (the Unison mechanism is explained in section 3.5.6 below). This contrasts with LVC set-up on the CR in which the OPEN was sent out-of-band (to a well known public port), but the OPENACK was returned in-band (to the privately allocated reply port). Hosts on the CFR do not require knowledge of well known public ports; such knowledge is confined to the association set-up service.

3.5.4.3 Unison Data Link Protocol

The UDL protocol provides a lightweight data link service to higher level network services. It defines the way in which blocks of data are transferred between peer users over an association. UDL is not concerned with flow control or error recovery; the intention was to provide minimal protocol overhead in order to support a wide variety of transport level services. These transport services can then be tuned to handle the particular requirements of the various types of traffic using the network. The protocol is defined in [Tennenhouse 86].

The UDL PDU comprises 4 bytes of protocol information including the port number and various re-assembly parameters; this leaves 28 bytes for user data in each minipacket. Note that although the 'port' component (which refers to the underlying association) is embedded in the UDL protocol data unit, the notion of an association should be considered quite independent of the segmentation and re-assembly service provided by UDL. The association is the fundamental concept; protocols other than UDL could run on top of an association if desired.

3.5.4.4 Unity RPC

The Unity RPC mechanism [McAuley 87] defines a way of performing remote procedure calls over a network. It is based on the ideas discussed in [Hamilton 84]. It can be decomposed into a marshalling layer and data transport layer. The marshalling layer is concerned with the language and machine independent representation of procedure call parameters. The data transport layer is responsible for the actual exchange of data between host systems on the network. These two functions map on to the ISO Presentation and Transport layers respectively.
The marshalling layer is responsible for the packing and unpacking of call parameters to and from network buffers. Such data is encoded using the Sun external data representation (XDR), thus making it machine and programming language independent. The processes of marshalling and unmarshalling often account for the largest overhead when performing an RPC, especially if byte swapping is required.

The transport layer is responsible for getting the data in these buffers to the required destination. It requires support from a data link service to permit the exchange of 'reasonable' size blocks of around 1 kbyte.

Unity defines three different semantics for RPCs requiring three different transport layer services. The At Most Once semantics ensure that the remote procedure is executed no more than once. The Exactly Once semantics ensure that the remote procedure is executed only once (disregarding server crashes). The At Least Once semantics allow multiple invocations of the remote procedure. The Exactly Once semantics are closest to those of a local procedure call, but such a call places the greatest demand on the transport service. Better performance might be expected using the other types of call. At Least Once calls are only good for idempotent operations whose completion is necessary. At Most Once calls provide the lowest level of service and should be used for operations whose successful completion is not essential.

![Diagram](https://via.placeholder.com/150)

**Figure 3.9: The General RPC Mechanism**
The RPC mechanism provides a convenient programming environment in which the details of the underlying network are largely hidden from the application. Network exceptions (such as lost packets or network component failures) can be handled in a uniform manner. Another benefit is that the RPC mechanism alleviates the problem of the choice of timeouts - particularly important for nested transactions. This is achieved by allowing the client to monitor the progress of a server operation using alive packets.

RPC data transmitted over the network contains no typing information - it is implicit in the RPC function definition. It is therefore essential that the client and server agree on the type and number of parameters being exchanged over the network. A convenient solution for identical client and server programming environments is to define the RPC interface in terms of an interface definition language and then to use this to generate client and server stubs. A stub comprises a sequence of calls to handle the marshalling and unmarshalling of data, and to perform the RPC itself. These can be bound to the client and server at compile time, and therefore guarantee that there is agreement on the RPC data to be exchanged. The role of the RPC stubs is illustrated in Figure 3.9.

3.5.5 Naming and Addressing in Unison

3.5.5.1 Naming Scheme

A Unison location name takes the simple form `<domain>·<localname>`. Domain names are global within the network, but each domain represents a privately managed name space. A domain may contain an arbitrary collection of stations, but is constrained to be no larger than a single CFR (since there is not concept of a 'sub-net' in Unison). In practice, there is usually a one-to-one mapping between a domain name and a CFR.

3.5.5.2 Secretary

The Unison secretary is the entity that resolves a `<service>, <location>` name to a `<station>, <port>` network address instance. Unlike the situation with the CR nameserver, this binding between name and address is dynamic. The secretary is at a well-known station address (`FF00H`), and this is the only address that needs to be built into host software. There is a one-to-one mapping between a secretary and a domain. Secretaries interact to resolve remote service names.

3.5.6 Association Set-up

The fundamental role of the secretary is in the establishment of associations between peer users. In Universe, call set-up was a two stage process involving firstly using the nameserver to resolve the service name to the corresponding (public) network ad-
dress and secondly using an initiation protocol between peers to exchange (private) ports for the connection itself. Furthermore, the initiation protocol uses an asymmetric procedure in which the call request is signalled out-of-band (to the public network address) but call confirmation is received in-band (on a private reply port). In Unison, association set-up is a single stage process involving the use of the secretary to resolve the service name and to exchange private port numbers, all performed entirely out-of-band. In this way, the secretary combines the role of a nameserver with that of a 'call set-up' server.

At the start of day, every host must create a special S-association with the secretary. This involves a single minipacket exchange with a universal port number on the secretary (42) during which private port numbers are exchanged. S-association set-up is one of the few occasions when a host needs knowledge of a well known public port (another occasion is for booting). All further communication between secretary and host is via the Unity RPC mechanism over this S-association.

A server must advertise its services with the secretary in order for them to be accessible to other network users. Unlike conventional nameservers, the secretary maintains a dynamic list of services available; when an S-association breaks, any registered services are automatically deleted.

---

**Figure 3.10: Local Association Set-up**

Numbers represent the order of events
To create an association with a named service, a client makes an association.create RPC to the secretary quoting the name and location of the desired service. If the secretary is able to resolve the \(<\text{servicename}>, \text{location}\>\) doublet to an address within its domain, it attempts to establish the connection by forwarding the request to the corresponding server down its S-association. If the service accepts the call, a successful response code is returned to the client via the secretary, together with the \(<\text{station}>, \text{port}\>\) doublet that identifies the association. See Figure 3.10.

3.6 The Long-Haul Segment

The long-haul segment of the Unison network is provided by an ISDN. The exchange at each site provides primary rate access to a 2 Mbps slotted frame carrier defined by the CCITT G.732 Recommendation. The primary rate interface is defined by the CCITT I-series of Recommendations for an ISDN [CCITT 85]. This interface is configured as thirty 64 kbps timeslots (known as B-channels) for user data, and one 64 kbps D-channel used for signalling to control the B-channels. For the duration of the project, the ISDN was a pilot network commissioned by the Alvey Directorate and known as the Alvey High Speed Network. This had a simple star-shaped topology in which the limbs were BT Megastream links to a central switch in London. More recently (August, 1989), the availability of the BT Multiline IDA (Integrated Digital Access) service has enabled the use of a public network. [Although the Multiline IDA service uses a different signalling protocol to that specified by the CCITT, eventual convergence to the standard is expected.]

3.7 The Unison Network Architecture

3.7.1 Introduction

This section describes the wide area architecture that was developed to interconnect distribution networks. It begins by describing the components that make up the exchange. This is followed by a discussion of the addressing and routing mechanisms, and a description of the exchange management functions that implement these mechanisms. Exchange support for multi-service traffic is then considered, followed by a description of the call set-up procedures between distribution networks.

3.7.2 Exchange Components

3.7.2.1 Exchange

Site access to the ISDN is provided by the Unison exchange. The exchange is based on a CFR for internal switching. The CFR packet (carrying a 32-byte payload) is
3.7 The Unison Network Architecture

the basic unit of transmission across the Unison network. The choice of a slotted ring for the exchange was partly due to the fast access and good bandwidth sharing properties of such a LAN. Together with the small slot size, these properties make the CFR a good choice for the switching of integrated services traffic. Although other LANs (such as Orwell [Falconer 85]) offer better theoretical performance for this purpose, implementations of these schemes are not readily available.

3.7.2.2 Ramp

Exchange access to the ISDN is provided by the ramp [Burren 89b]. This device maps CFR packets from the exchange ring onto the timeslot frame structure of the ISDN. The combination of two ramps and the ISDN makes up an inter-site bridge. The ramp allows a number of 64 kbps B-channels to be grouped together into a U-channel to enable the bandwidth of an inter-site connection to be multiples of 64 kbps. The bandwidth allocated to a U-channel can be varied dynamically [Harita 89], and U-channels can be created and deleted according to inter-site connectivity requirements. There may be multiple ramps on a given exchange.

3.7.2.3 Portal

Client LAN access to the exchange is provided by a portal. A portal accepts data from the distribution network and transmits it to a peer portal for injection into the destination network. Inter-portal traffic flows over a set of associations (which extend end-to-end for the special case of CFR distribution networks). Portals serving distribution networks other than than the CFR may need to segment packets longer than a CFR slot; such portals must run a link layer protocol (such as UDL) on each inter-portal association to enable re-assembly. There may be multiple portals on a given exchange. Note that protocol conversion is not a portal function; the facility to communicate between dissimilar client networks is a value added service to be provided by a separate network entity.

The implementation of a portal is dependent on the particular distribution network that it is intended to serve. One half must conform to the Unison exchange architecture in order to interact with management services and provide access to peer distribution networks. The other half must conform to the architecture of the distribution network itself. The portal makes use of the services of the exchange to allow distribution network traffic to be carried over the Unison network. For example, the CR portal maps each LVC on the CR LAN onto an appropriate association over the Unison network.

Since the architecture of the CFR distribution network is essentially the same as that of the exchange, the functionality required of the CFR portal is relatively simple. CFR minipackets are simply relayed between client and exchange CFRs; associations exist end-to-end between distribution network hosts.
3.7 The Unison Network Architecture

The portal and exchange effectively provide the relaying functions of the ISO Network Layer using the short packets and low link layer overheads of the CFR. This level of support is adequate for applications that require lightweight connections without flow control or error recovery. Clients requiring flow control and error recovery must engage in a suitable end-to-end protocol. Indeed, ISO Transport and upper layer protocols can be exercised if required on an end-to-end basis. In this way, support is provided for a range of traffic types including real-time voice and conventional data.

3.7.3 Addressing and Routing

To support inter-site addressing, the 16-bit CFR address space is partitioned into windows by splitting the address field into equally sized window and station components. Devices on an exchange CFR have the same local window value (FFH, the same for all exchanges) but unique station components. Other window values are used to access remote sites. The restriction on the number of stations on a given exchange is not considered too severe since most of the stations will be portals, each of which serves an entire client distribution network.

A window is a portion of a site's address space that is temporarily assigned to support communication with a remote site. Window values are only assigned when connectivity is required to a remote site; they are reclaimed when no longer required. Window management is a function entirely local to each exchange; window values are intended to be exchanged at ISDN call set-up time. Peer window services co-operate to effectively knit together the CFR address spaces of their respective sites. [Note that the dynamic arrangements for window management were not implemented during the project; window values were allocated manually as required.]

The ramps are dynamically configured to extract selected packets from the CFR exchange and to map these packets onto corresponding U-channels. The ramp examines the destination window value for each packet and extracts those that are destined for foreign sites. The ramps also perform the function of mapping window values between sites. The source address of an outgoing packet contains the local window value, and the destination address contains the window value of the peer exchange. The source address of an incoming packet is mapped onto the peer exchange's window value for the originating exchange, and the destination address is mapped onto the local window value.

A consequence of the dynamic allocation of address space to windows is that Unison addresses are transient values used to support peer portal communications. No attempt has been made to support global addressing. This reflects the desire to support connectivity to a large number of sites, as will be the case in a large public network. Although the number of simultaneous connections is limited, the number of potential connections is unlimited.
3.7.4 Multi-service Support

As a first step towards providing multi-service support, the Unison architecture includes a simple two-level priority scheme which operates on a per-association basis at the portal and on a per-window basis at the ramp. Priority packets get preferential treatment in terms of delay and congestion control. Clearly, the distribution network architecture must include some priority mechanism in order to be able to make use of this facility. For example, the CFR distribution network architecture includes such a scheme, and priority can be requested on a per-association basis at association set-up time. Packets arriving at the CFR portal on a priority association are mapped on to a suitable priority window (supplied by exchange management at association set-up time).

3.7.5 Exchange Management

Exchange management functions are responsible for name resolution, communication resource allocation and routing. This includes the establishment and management of U-channels to remote sites and the dynamic mapping of window values to U-channels. Access to, and interactions between, exchange management functions are entirely based on the Unity RPC mechanism.

A common feature of the Unison exchange management functions is that they operate out-of-band with respect to the data flows that they control. Although the services are built up in a layered fashion, they do not conform to the OSI model in which high level protocol elements are embedded in those of the lower layers. The use of out-of-band techniques reflects both the desire to support small packets with minimal protocol overhead (especially for voice) and the requirement to maximise bridge performance by minimising per-packet overheads.

The exchange secretary service handles all inter-portal association requests. Association requests to services within the same site can be satisfied by simply routing the request to the appropriate portal on the same exchange. Association requests to remote services (ie off site) require support from underlying exchange services to provide access to the ISDN.

The window service is responsible for establishing windows between sites and for binding those windows to appropriate U-channels and ramps. The window service is intended to provide clients with an ISDN bandwidth allocation facility according to the bandwidth requirements specified at call set-up time. This is the predictive component of the bandwidth management strategy described in [Harita 89]; if such bandwidth requests cannot be met, call blocking must be employed to prevent excess traffic from flooding the network.
The channel service provides the necessary control for the establishment and management of U-channels over the ISDN. The channel service is responsible for supplying the reactive component of the bandwidth management strategy described in [Harita 89]; the channel service needs to monitor ramp queue lengths and adjust U-channel sizes accordingly. The first client to access a service at a particular site experiences a start of day delay associated with connection establishment (exchange management overhead and ISDN slot synchronisation delay [Burren 89b]). Further accesses to the same site are not so delayed.

The signal service is provided by the ramp and implements the necessary signalling protocols to manage calls over the ISDN.

A full description of the various exchange management functions that provide inter-site routing and bandwidth management is beyond the scope of this thesis. For a more detailed overview, refer to [Tennenhouse 87]. For a comprehensive review of the design and performance of the Unison exchange, refer to [Tennenhouse 88].

3.7.6 Call Set-up Over Unison

3.7.6.1 Introduction

This section describes the mechanism for association set-up between CFR distribution networks over the Unison network. It then goes on to outline the mechanism for LVC set-up between CR distribution networks over the Unison network. It is worth mentioning that an experiment was conducted between Loughborough and Cambridge during which an IP connection was set up between hosts on Ethernet distribution networks. This is not relevant to the work described in this thesis, and it is not considered further here. However, it does illustrate the ability of the Unison network to support the interconnection of a third type of distribution network.

3.7.6.2 Association Set-up Across Unison

If the location parameter in the client association.create RPC specifies a foreign domain, the client secretary must interact with exchange management services in order to be able to set up the association. The association.create RPC is forwarded to the portal down the portal's client-side S-association. The portal then passes on the request to the exchange secretary down it's exchange-side S-association. The exchange secretary then attempts to establish the connection to the specified domain.

If this is a domain on the same exchange (ie within the same site), the exchange secretary simply forwards the request to the appropriate portal for that domain where it is handled by the secretary in that domain. The reply is relayed back via the exchange secretary through the portals. This situation is shown in Figure 3.11.
If this is a remote domain, the exchange secretary requests a window to the peer exchange secretary. The `association.create` RPC is then forwarded across the network, where it is processed by the peer secretary in the same way as described above. This sequence of events is shown in Figure 3.12.

### 3.7.6.3 LVC Set-up Across Unison

In the case of the CR, the portal must map address insertion requests and initial connection blocks onto suitable associations between peer portals. In this environment, associations only extend as far as the portals that serve the distribution networks; LVCs exist end-to-end and map onto suitable associations over the Unison network.

For this to operate dynamically, the CR portals would require support from exchange management functions to resolve names, allocate windows, manage associations, etc. However, lack of management services on the early 'pilot' exchange meant that such
mappings were static for the CR portal. These mappings were actually defined in a file read by the portal at initialisation time.

3.8 Summary

This chapter outlines the architecture of the Unison network. The intention was to design a wide area network with similar characteristics to that of a local area network. This has been made possible by the recent provision of digital public bearer services providing a quality of service approaching that available from the current generation of local area networks (in terms of bit-error rate if not bandwidth). The network is
based on an ISDN which allows variable bandwidth circuits to be switched between sites. Clients on local distribution networks are presented with a lightweight packet-relaying service between sites. In order to meet the requirements of real-time traffic, high speeds and low delays have been the major design criteria.

The architecture of the Cambridge Ring and Cambridge Fast Ring client networks has also been described in the context of Unison. The CR architecture was essentially that inherited from the Universe Project. The CFR architecture encompasses a rather different approach to call set-up by using out-of-band techniques involving the secretary.

It should be apparent that the Unison network shares many of the characteristics required of an ATM network. An ATM-like overlay has been superimposed on the primary rate synchronous transfer mode service of the ISDN. A small fixed sized cell (the CFR minipacket) is the basic unit of transmission, and a simple cell-relaying service is all that is provided by the network. This is consistent with the desire to support the full range of multimedia services, each of which places different demands on the network. Clients requiring flow control and error recovery must exercise suitable end-to-end transport protocols on an application dependent basis. The relevance of Unison to ATM research is highlighted in [Tennenhouse 89].
Chapter 4

The Cambridge Ring Multimedia Office

4.1 Introduction

This chapter describes an experimental multimedia office based on the Cambridge Ring (CR) local area network. A number of such offices were interconnected by a pilot ISDN as part of the Unison Project. Many of the office applications described here were inspired by previous experiments conducted during Project Universe [Burren 89a] and by distributed computing techniques developed on the CR at Cambridge [Needham 82]. The Universe Project was concerned with extending the facilities of a local area network by interconnecting sites using a satellite. The Unison Project provided the opportunity for further research into office-oriented multimedia applications based on a network of offices interconnected by terrestrial links. Research into real-time services, such as voice and video, was of particular interest especially with regard to the suitability of the communications architecture for the support of fully integrated services.

Much of the office infrastructure was inherited from the Universe Project, although considerable effort was directed towards replicating the video framestores during Project Unison. The CR office provided the testbed for applications development during the first half of the Unison Project. Later on, it was superseded by the Cambridge Fast Ring office described in the following chapter.

The bulk of this chapter describes the work done in providing a flexible video system based on the new batch of video equipment. This exercise proved to be an interesting introduction to network server design in general, and to issues of component control and user interface design in particular. The important lessons learned from this experience have influenced the design of later systems described in this thesis, particularly relating to the management of control.

The chapter begins with a brief overview of the network components making up the office (these are described more fully in [Murphy 88]). It goes on to describe the implementation of the CR interface for the office workstation. The video system is then described in some detail, followed by a brief account of the experiments conducted over the wide area network.

I am indebted to Tim Rodgers who supervised the replication of the video framestores, and who therefore made my work possible.
4.2 Overview of the Office

4.2.1 Introduction

The Cambridge Ring office comprises a collection of networked multimedia components which can collectively provide a variety of services to the user. These services include distributed computing facilities such as remote file handling, connection-oriented computer data services such as file transfer and remote terminal access, and new types of services such as real-time voice and slow-scan video. The CR office meets many of the requirements of the integrated office in terms of the variety of traffic which it can support. However, the most important shortcoming is that there are no mechanisms for the user level integration of these media.

The design of the office is based on the sort of distributed workstation concept discussed in chapter 2. Each office component has an independent network interface and handles a particular type of medium (e.g., voice, video, text). In principle, a central

![Diagram of the Cambridge Ring Multimedia Office](image)
workstation acts as the controller for these components, although in practice most CR components were autonomous in terms of control. Most of the components were designed during the Universe Project, except for the A500 workstation which became available early on in the Unison Project.

A schematic diagram of the office is given in Figure 4.1, and the general layout of the office is shown in Photograph 4.1.

4.2.2 Office Components

4.2.2.1 The A500 Workstation

The A500 is a prototype workstation in the Acorn Archimedes range of microcomputers. It is based on a custom designed RISC (Reduced Instruction Set Computer) microprocessor, the Acorn RISC Machine (ARM). The ARM processor has three support chips: the Memory Controller (MEMC), the Video Controller (VIDC), and the Input/Output Controller (IOC). The MEMC is responsible for memory management, the VIDE controls the screen display and the sound system, and the IOC controls external peripherals. The A500 features 4 Mbytes of RAM, one floppy disk drive, one 20 Mbyte hard disk, and an expansion bus. Access for external peripherals to the expansion bus is via the Podule hardware interface. Refer to [Chambers 86] for details on the architecture of the ARM series of workstations.

The A500 workstation forms the heart of the office. It provides the services expected of a standard office microcomputer such as word processing and spreadsheets. It also offers a limited number of network services such as access to the network laser printer. More interestingly in this context, it performs the additional role of providing the user interface to those office components that are controlled over the network. In order to perform this role, a Cambridge Ring network interface was required for this machine. The design of such an interface is described in section 4.3 below.

4.2.2.2 The Video Framestore

The video framestore is a Multibus system comprising an Intel 8086 based 86/30 single board computer [Intel 82], a Logica VMI-1 Cambridge Ring intelligent interface [Logica 81a], and custom built video controller, memory and analogue boards. The system uses a Logica Polynet Cambridge Ring node [Logica 81b]. Each framestore is capable of both image capture and display. The framestore has two $512 \times 512 \times 16$ bit memory planes, giving it the ability to work in full-duplex mode (one plane for grabbing the outgoing image, the other for displaying the incoming image). It includes a powerful graphics display processor (GDP) which supports vector graphics and windowed displays. The framestore deals with YUV (luminance, chrominance) signals which provide a better basis.
This photograph shows an early version of the Cambridge Ring office. The 'workstation' is the BBC microcomputer in the centre of the picture; the screen displays the original menu-driven framestore control software. The screen on the right displays the framestore output including an electronic blackboard image in the bottom right quadrant. The electronic blackboard is driven by the graphics tablet to the right of the keyboard. A head-and-shoulders image of the office worker is captured by the camera to the right of the workstation monitor. The screen on the left displays the output of the still image framestore; this store is driven by the overhead camera mounted on the stand on the extreme left. The toll-quality telephone handset is on the right of the desk. The wideband speech microphone is the flat object on top of the workstation monitor, and one of the loudspeakers is at the back of the desk behind the telephone.

In later versions of the office, the BBC microcomputer was replaced by the A500 workstation. In other respects, the appearance of the office was essentially unchanged.
for quantisation and coding than RGB (red, green, blue) signals. The video framestore is described more fully in [Lee 86a].

The framestore can grab frames in real time, but lack of processor bandwidth restricts the useful data rate to about 500 kbps. By simplifying the framestore receiver software (for example, by removing support for multiple streams), it is believed that data rates approaching the limit of the VMI-1 (about 800 kbps) would be achievable.

In common with most Cambridge Ring basic block protocol implementations, the VMI-1 is essentially a half-duplex device at the basic block level. That is, only one basic block can be received or transmitted at any one time, so all data is interleaved at this level (up to 2 kbytes of data per basic block). When a framestore is used in full-duplex mode (ie transmitting and receiving simultaneously), apart from the performance suffering, the appearance of the incoming picture is rather jerky due to this interleaving effect. Therefore, in practice a framestore is dedicated to be either a transmitter or a receiver. A transmitter framestore takes in a video signal from a camera, digitises it, and processes it for transmission over the network. A receiver framestore takes image data off the network, reconstructs the digitised frame, and displays it on an attached monitor.

The framestores are fully programmable in terms of the transmission protocol used, the compression algorithm, the image resolution, the frame update rate, etc. However, lack of processing power prevents the effective use of an image compression algorithm, and the time taken to transmit a full size, full resolution image is about eight seconds (see section 4.4.6.1). A framestore may have multiple connections (referred to later as 'channels') to other stores, and a receiver framestore can display incoming data from different channels in separate windows on the framestore display.

4.2.2.3 The Telephone

A real-time voice system has been inherited from the Universe Project. This system provides the functionality of a conventional telephone system in terms of dialling, ringing, etc. The system comprises a custom-built 64 kbps codec board (which includes silence detection hardware), together with a Z80 processor board and CR interface. The speech transmission protocol uses an 'adaptive delayed time' algorithm which attempts to compensate for jitter by adjusting for network delays. Refer to [Adams 85] for details.

4.2.2.4 The Wideband Speech System

The wideband speech system uses a sub-band adaptive differential pulse code modulation (SB-ADPCM) coding scheme to compress 7 kHz of analogue voice input down to the standard 64 kbps data rate associated with PCM speech. The system uses the same Z80 processor board as the Universe telephone described above. The wideband
4.2 Overview of the Office

The speech system does not attempt to emulate a conventional telephone system. Instead, a microphone and loudspeakers are used in place of a telephone set for voice input and output. Refer to [Yin 86] for details.

4.2.2.5 The Tripos Server

The Tripos server is a general purpose VME [VME 82] 68000 machine running the Tripos operating system. Tripos ([Richards 79], [Knight 82], [Wilson 85]) is a message-based real-time multi-tasking operating system developed at the Cambridge Computer Laboratory. The server is used in the office for driving the network printer and loading some of the office components at boot time. Tripos also provides the environment for most of the distributed computing facilities available in the office.

4.2.3 Office Services

4.2.3.1 Video Services

The video service allows still images, slow-scan video, and electronic blackboard data to be transmitted across the network. A user may interact with the video service by invoking the control software on his workstation. The video service is fully described in section 4.4.

4.2.3.2 Voice Services

The Universe telephone system uses a familiar loop-disconnect push-button telephone set for call set-up. Control is therefore independent of the workstation. The service provided is essentially that of a conventional telephone system.

The wideband speech codecs take speech from a microphone, and the amplified speech output drives a loudspeaker. The original intention was that control would reside in the workstation for initiating call set-up, controlling the volume, etc. However, a shift to the Cambridge Fast Ring environment pre-empted this work. For the purposes of development, fixed network connections were built into the software.

4.2.3.3 The Electronic Blackboard Service

The prototype blackboard service allows a workstation user to open a blackboard window on a remote framestore. Co-operating users can superimpose blackboard windows to provide a very simple shared blackboard facility. Alternatively, a user can annotate a still image displayed on a remote framestore. Simple graphics primitives have been implemented for drawing basic shapes, for pattern filling specified areas, for typing text from the keyboard, and for free drawing arbitrary shapes. A limited range of colours are also available. Refer to [Lu 89] for details.
4.3 The A500 to Cambridge Ring Interface

4.2.3.4 Workstation Services

As well as controlling other office components, the workstation can be used for traditional personal computing facilities such as word processing, running spreadsheets, etc. Local files may be printed on the networked laser printer.

4.2.3.5 Distributed Computing Facilities

During the early part of the 1980s, the Tripos operating system was the basis for much of the distributed computing research work conducted in the Cambridge Computer Laboratory [Needham 82]. The distributed computing facilities available in the office were based on the Tripos environment inherited from Cambridge. These facilities included remote printing, remote terminal access, file sharing, etc.

4.3 The A500 to Cambridge Ring Interface

4.3.1 Introduction

Although the A500 was an attractive candidate for the role of office workstation, the problem was that a Cambridge Ring network interface was not available for this machine. One solution would have been to design and build a Cambridge Ring interface podule, and to write a Cambridge Ring network device driver. However, this was ruled out due to lack of resources. The solution adopted was to make use of an Acorn Master microcomputer to provide access to the network (the interface hardware and network device driver already existed for this machine), and to link the Master and A500 by means of an Acorn proprietary processor expansion bus (the Tube). A Tube podule and Tube podule device driver were available for the A500. This approach required minor alterations to the Master 6502 network device driver, and the development of 6502 Tube management code to handle the link. Although the performance of this interface was fairly low (about 200 kbps maximum throughput), it was quite adequate for the modest requirements of the traffic using it (control and electronic blackboard data).

4.3.2 Configuration

The hardware configuration for this interface is shown in Figure 4.2. The CR interface code is based on a version of the Universe 6502 ring driver [Cooper 84] modified to run in an Acorn Master microcomputer. The ring driver implements the basic block protocol. The 6502 processor in the Master is solely concerned with running the ring driver, and is referred to below as the ring processor. The A500 processor is referred to here as the host processor. The Master is interfaced to the CR via the
4.3 The A500 to Cambridge Ring Interface

Acorn '1 MHz Bus' connected to a BBC/Seel interface card. This in turn connects to a Seel station and repeater.

Workstation access to the ring driver is by means of the Acorn 'Tube'. This is a bus which allows high speed data transfers between processors. The host and ring processors are tightly coupled via the Tube.

4.3.3 Software Interface

4.3.3.1 Ring Driver

The software interface to the ring driver is by means of a control block known as a basic block descriptor (BBD). The structure of this descriptor is shown in Figure 4.3, and it is fully defined in [Murphy 87].

The ring driver is entered by making a Master system call (OSWORD) which is unrecognised by the Master's resident operating system (MOS). The MOS polls service ROMs for a claimant service routine, which in this case is supplied by the ring driver. The driver is passed a pointer to the BBD, which must be in ring processor address...
4.3 The A500 to Cambridge Ring Interface

<table>
<thead>
<tr>
<th>0</th>
<th>Port</th>
<th>Station</th>
<th>Timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Action</td>
<td>Flags</td>
<td>Callback routine</td>
</tr>
<tr>
<td>8</td>
<td>Host buffer address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Max buffer len</td>
<td>Actual buffer len</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.3: The Basic Block Descriptor
Numbers represent byte offsets

Across the Tube. Finally, it is necessary to copy back the modified BBD to the host. It is the Tube Manager layer of code (running on the ring processor) that provides these services.

4.3.3.2 Tube Manager

The basic mechanism for communicating over the Tube is provided by the Tube driver relocatable module running on the A500. The Tube driver is activated by typing "BBC". Thereafter, any command prefixed by '@' is passed (via the Tube driver) over the Tube, and interpreted by the command line interpreter (CLI) of the attached machine.

<table>
<thead>
<tr>
<th>0</th>
<th>Master buffer address</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Host buffer address</td>
</tr>
<tr>
<td>4</td>
<td># Bytes</td>
</tr>
<tr>
<td>6</td>
<td>Direction</td>
</tr>
</tbody>
</table>

Figure 4.4: The Tube Control Block
Numbers represent byte offsets

Master buffer address - is the 2 byte 6502 address to/from which data is to be copied
Host buffer address - is the 4 byte host (A500) address to/from which data is to be copied
# Bytes - number of bytes to copy
# Pages - number of pages (256 bytes) to copy
Direction - direction of transfer
Action - Tube Manager control byte

The MOS command 'LINE' can be used to run 6502 assembler code routines indirected via the MOS user vector (USERV). This mechanism is used to run the Tube management code referred to above. The procedure is to first load the Tube management code into Master memory, and then to point the USERV to its entry point by running a small 6502 assembler routine from disk. This is all done at start of day. Subsequent @LINE commands to the host CLI then invoke the Tube management code on the Master.

The interface to the Tube manager is by means of a control block.
known as a tube control block (TCB). The structure of the TCB is shown in Figure 4.4. This is set up at a fixed location in host address space. When the Tube manager is entered, the TCB is copied across the Tube and decoded. The TCB contains a control byte which determines the action to be taken. In the case of the action.copy command, the rest of the TCB specifies the amount of data to be copied (the size of the BBD), the direction of data transfer, and the source and destination addresses. In the case of an action.driver command, a call is made to the ring driver handing over a pointer to the Master's copy of the BBD. Note that interrupts are temporarily disabled on the Master during all Tube transfers so as not to lose interrupts from the ring interface hardware (this would cause the loss of incoming blocks).

4.3.4 Operation

The full procedure for calling the ring driver from the A500 is now explained. The required BBD is set up somewhere in A500 address space. The TCB is set up (at a fixed location) to instruct the Tube manager to copy the BBD from host to ring processor address space. The first call is then made to the Tube manager (using a @LINE system call) to perform the action.

Having set up a copy of the BBD in Master address space, the ring driver must be called to perform the required action. To do this, the TCB is set up with the appropriate action code, and the Tube manager is called for the second time.

Once the ring driver operation is complete, the updated BBD must be copied back into the original host location to allow the application to examine the outcome of the transaction. The TCB is set up for this action, and the Tube manager is called for the third and final time. This completes a single ring driver transaction.

4.3.5 Performance

Many such transactions are required for a simple operation such as performing an SSP exchange: a reply port is allocated for the SSPRPLY, a reception request is submitted for the SSPRPLY, the SSPREQ is transmitted, the driver is repeatedly polled for the completed reception of the SSPRPLY, the BBD is copied back from the driver's private pool into 6502 address space, and finally the reply port is de-allocated. This requires the exchange of about 400 control bytes across the Tube, plus of course the SSP data itself.

Despite this high overhead, the interface is fairly efficient. Using a BASIC ring handler running on the A500, almost exactly three SSP transactions (with the nameserver) can be performed per second. This was improved by approximately 20% using a ring handler written in the C language. This performance is quite adequate for the modest requirements of control information and electronic blackboard data.
4.4 The Video System

4.4.1 The Background

The video system is based on a collection of framestores located around the network, and on one or more workstations necessary for their control. The framestores were designed at Loughborough during Project Universe, and a couple of prototypes were built and successfully demonstrated working between Loughborough and Geneva at the Telecom 83 show [Burren 89a]. These were replicated during Project Unison, giving several more reliable systems for multipoint experimentation. The workstations were provided by Acorn Computers Ltd during Project Unison, and interfaced to the Cambridge Ring by the author (section 4.3 above).

The Project Universe framestore software offered the user a point-to-point slow-scan video service between framestores on the network. A block-conditional coding scheme was implemented to economise on network bandwidth. Refer to [Griffiths 84].

Further software was written early in Project Unison to support a quadrant display on the receiver framestore (Photograph 4.2). One quadrant could display a local live image; a second quadrant could display a remote live image. In addition, support was added for the transfer of a still image from the 'workstation' (then a BBC microcomputer), and for the transfer of electronic blackboard data from a graphics tablet attached to the BBC microcomputer. The still image was displayed in a third quadrant, and the electronic blackboard in the fourth. All network connections and transmission parameters were ‘hard-wired’ for the purpose of testing, and operation was confined to a single Cambridge Ring. Refer to [Lee 86b] for details.

The concept of the new video system was one in which a single framestore would be able to support multiple connections and multiple windows of live video, still pictures and electronic blackboard data. Control would be entirely with the workstation user on a per-call basis, and not hard-wired into the server.

4.4.2 System Design

There are three traffic streams to consider:

- slow-scan video
- still image (snapshot) transfers
- electronic blackboard.

A raw video stream is characterised by the continuous flow of high volumes of data between two framestores. Occasional data loss can be tolerated for the sake of speed.
Photograph 4.2: The Framestore Quadrant Display

This photograph shows a close-up of the framestore quadrant display (on the rightmost monitor). The two left quadrants display slow-scan video: the lower quadrant shows the local image from the camera on the desk; the upper quadrant shows the remote image from another framestore on the network. The top right quadrant shows an image captured from the overhead camera attached to the still image framestore (the output from this framestore is shown on the screen on the left). The bottom right quadrant shows an electronic blackboard image generated using the graphics tablet shown to the right of the keyboard.

The video system described in this chapter allows any number of arbitrary sized windows containing video, still images and electronic blackboard data to be displayed on the screen.
A still image transfer is characterised by the bursty flow of high volumes of data between two framestores. Reliability is more important than speed.

An electronic blackboard stream is characterised by the bursty flow of low volumes of data between workstation and framestore. Once again, reliability is more important than speed.

In all cases, the aim is to provide flexible call set-up procedures so that the user has complete control over the parameters relating to each stream (such as size and position of a still image picture, or speed of a video transmission).

A description of the video system relies on the notion of a channel. A channel is a simplex connection based on a virtual circuit between two machines on the network. It defines the mapping of a data structure (say a video frame) between transmitter and receiver. Associated with each channel is a data structure referred to as a channel descriptor. Identical copies of the channel descriptor reside in the machines at each end of the channel. There can be different types of channel, each type corresponding to the different type of stream identified above. Each channel descriptor is built up from mandatory parameters common to all channels, and other parameters relevant only to the particular type of channel. The three types of channel descriptor are defined in Appendix A.

The protocol used for data transmission on a channel depends on the type of channel; that is, the transport class is determined by the channel type. For example, a very lightweight protocol is used for the transmission of video data. In this case, it is preferable to sacrifice reliability for speed. On the other hand, a more robust reliable stream type protocol is used for a single image transmission. In this case, speed of transfer is less important than reliability.

At least one channel must exist between two machines before data can flow. In general, there will be any number of channels of any type between any number of machines. The workstation is responsible for initiating the setting up of all channels; workstation software prompts the user for the various parameters required to build each pair of channel descriptors. In a similar way, the workstation software is responsible for the orderly closing down of redundant channels. Channels may be set up or closed down at any time.

Having set up a channel, the workstation can initiate data transfer when required. When instructed to transmit on a specified channel, a machine will decode the relevant channel descriptor and call an internal transmission routine appropriate to the channel type. A pointer to the descriptor is passed to this routine. The descriptor will tell the machine how to transmit the data. For example, in the case of a video channel,
the descriptor will define the area of the frame to be transmitted, the speed of transmission, etc.

In general, a machine will maintain a list of channel descriptors corresponding to each channel handled by that machine. A transmitter will cycle through its list of descriptors and transmit one unit of information (for example, a frame in the case of a video transfer, or a basic block, in the case of an electronic blackboard transfer) per channel. It then moves on to the next active descriptor, and the process is repeated. The list of channel descriptors is circular.

The receiver first filters incoming blocks on the basis of block type (SSPREQ, OPEN, Video data, etc). For data blocks, the sub-address (port number) of incoming blocks is then used to index the appropriate channel descriptor. The descriptor instructs the receiver on how to process the incoming data (e.g., which window to use to display a block of video data).

4.4.3 System Implementation

4.4.3.1 Framestore Software Development Environment

All framestore software was developed on an Intel Ethernet-based Microprocessor Development System (MDS) and blown into EPROM. The bulk of the code is written in Pascal, except for some low level library routines that are written in 8086 assembler.

Blowing EPROMs to evaluate new code proved to be a very tedious and time consuming affair. A better approach might have been to have programmed a bootstrap EPROM, allowing new framestore code to be loaded across the network. This is also the preferred solution for the distribution of software updates, possibly to remote sites. [In practice, such updates were made by posting a new set of EPROMS!] The problem was that this would have required an 8086 implementation of a suitable boot protocol [Wilbur 83] which would have diverted effort. Furthermore, there remains the problem of transferring the bootable code from the MDS to the network bootserver (over an RS-232 link); the time saved over blowing new EPROMs would have been minimal.

4.4.3.2 A500 Software Development Environment

During the period of video system development work, the A500 workstation was running the Acorn Arthur operating system (initially version 0.066, and later version 1.2). Arthur is a simple ROM-based single-threaded operating system sharing many of the characteristics of earlier BBC microcomputer operating systems. Although support was later provided for a good range of programming languages including C and Fortran, the BASIC interpreter was the only tool available for early program development. The Acorn BASIC implementation is fairly efficient and sophisticated, and many Acorn system programmers...
still use BASIC for commercial applications development. BASIC was the language used for the development of the workstation control and user interface software.

Arthur supports the concept of 'relocatable software modules' which can provide extensions to the basic operating system (much as ROMs could be plugged into the previous generation of BBC microcomputers to enhance the basic machine). An Arthur module may be a filing system, a language, or a general utility such as a window manager or an assembler code debugger. The Arthur window manager (WIMP) module proved to be an invaluable tool for user interface design.

4.4.3.3 The Framestore Software

The structure of the framestore software is shown in Figure 4.5. The framestore supports two interfaces to the outside world:

- the terminal interface
- the network interface.

The terminal interface supports the input (from the terminal to the framestore) of single character instructions, and the output (from the framestore to the terminal) of variable length strings. The single character instructions are chosen from a menu of options that have been provided for testing purposes. For example, the user can display colour bars, grab a video frame, clear the screen to a specified colour, toggle between memory planes, etc. This aspect of the terminal interface allows video hardware debugging without the framestore needing to have a working network connection. The display of variable length strings is a very useful debugging aid for developing network software. Note, however, that the terminal interface is intended purely as a debugging tool, and it is not fundamental to the normal operation of the framestore. Note, also, that a framestore automatically enters 'stand-alone mode' if it is unable to access the network at initialisation time.
The network interface supports the reception and transmission of variable length basic blocks. The network interface supports the full control and data interface to the framestore. This framestore software interface is defined in Appendix A.

The framestore software is built around a central polling loop which listens to both the terminal and network handlers. A framestore starts up in this listening state, and always returns to it after any transaction. Incoming characters from the keyboard are decoded by the terminal handler and used to select a debugging function. Incoming blocks from the network are decoded by the network handler and passed either to the channel handler (in the case of control information), or to the graphics display processor (GDP) driver (in the case of data). For example, if the block is an SSP control block, the channel manager SSP service routine is called to decode the function code and perform the relevant action. If the block is a video data block, a call is made to the GDP driver to display the data (with reference to the appropriate channel descriptor). All network reception requests are double-buffered. Whatever the action, the framestore always returns to the listening state after completion.

The call set-up procedure is quite straightforward. The transmitter is sent an SSPREQ instructing it to make a connection with a named receiver. The channel parameters which go to make up the channel descriptor are supplied as arguments to the SSPREQ. The transmitter duly sets up the connection (or informs the workstation of a failure), passing the channel parameters to the receiver as arguments in the OPEN block. The receiver acknowledges with an OPENACK, prompting the transmitter to return an SSPRPPLY to the workstation. The SSPRPPLY returns two handles on the new channel, one allocated by the receiver, the other by the transmitter. These handles are known as channel IDs. Having been informed of a successful channel connection, the workstation can instruct the transmitter to start transmission on that channel.

Any number of additional channels can be added at any time, and existing channels can be modified or closed down. The procedure for modifying an existing channel (i.e., changing the pair of descriptors) is rather involved. Having collected the new channel parameters from the user, the workstation suspends transmission on that channel from the relevant transmitter. The workstation then sends the new set of channel parameters to both the transmitter and the corresponding receiver. The original channel descriptor parameters are substituted for the new values, and the transmitter is then instructed to resume transmission.

The framestore software includes code for the printing of images on a networked PostScript [Adobe 87] printer. To initiate a print, the workstation instructs the framestore to open a virtual connection to a named print service, and then to transmit a specified area of the image to the print server. This is done by the workstation sending an SSPREQ to the framestore with parameters including the name of the print service, the area of the image to be printed (for the benefit of the framestore), and a PostScript
4.4 The Video System

prologue (for the benefit of the printer). The framestore then sets up a connection to the print server by means of the usual OPEN/OPENACK exchange, passing on the PostScript prologue as part of the OPEN request. The PostScript prologue informs the printer of the size, position and orientation (portrait or landscape) of the image to be printed. The specified area of the image is then transmitted to the print server using a reliable protocol (refer to section 4.4.4.3 below). The print server processes the raw data from the framestore and generates a PostScript file to be interpreted by the printer.

4.4.3.4 The Workstation Software

The structure of the workstation software is shown in Figure 4.6. The software is distributed between the A500 host processor and the 6502 ring interface processor.

The workstation software supports two interfaces to the outside world:

- the user interface
- the network interface.

The user interface is built on the Arthur WIMP module. The WIMP module handles keyboard and mouse input, and drives a colour window-based display. The video control software provides access to the channel manager layer which is responsible for the setting up of channels between framestores (in the case of video), or between the workstation and framestore (in the case of the electronic blackboard). The WIMP interface also provides access to the blackboard software described in [Lu 89].

The ring handler implements the SSP and OPEN/OPENACK protocols, and handles the transmission of variable length basic blocks on behalf of upper layers. The Tube manager runs on the 6502 ring proces-
4.4 The Video System

sor and handles the transfer of network driver control blocks (basic block descriptors) between the host and ring processors. The 6502 ring driver implements the basic block protocol.

In a similar way to the framestores, the workstation code is built around a central polling loop that continually waits for WIMP events. Mouse clicks are interpreted by decoding the mouse position in the context of the current state of the software. The user interface is a complex state machine, and mouse clicks may or may not cause a state transition.

4.4.4 The Protocols

This section describes the protocols used for transmission on the three types of channel that form the video system. For each channel type, call set-up is by means of the OPEN/OPENACK protocol, and the data transfer protocol is built on the basic block protocol.

4.4.4.1 The Slow-Scan Video Protocol

This is a lightweight protocol using a simple flow control mechanism with no high level error correction or retry strategies. For simplicity, there is no image compression, so no coding scheme is supported.

In general, the area of the frame to be transmitted comprises \( p \) pixels (in the x direction) and \( l \) lines (in the y direction). The area can be sub-sampled in either the x or y direction (the frame resolution is 512 pixels by 512 lines). In general, the pixel pitch will be \( dx \), and the line pitch will be \( dy \). A pitch value of 1 gives no sub-sampling, a pitch value of 2 gives two times sub-sampling, etc. Each pixel is coded to 16 bits (2 bytes). The quantity of data to be transmitted per frame is then \( p \cdot 2 / dx \) bytes per line, multiplied by \( l / dy \) lines. These parameters are stored in the transmitter part of the channel descriptor.

The protocol data units (PDUs) are shown in Figure 4.7. A video data block contains a two byte header followed by an integer number of frame lines. Each line requires \( p \cdot 2 / dx \) bytes preceded by 2 bytes specifying the line number relative to the area origin (top left corner). The block is terminated by an end of block (EOB) marker. The block size is as large as possible given the two constraints that frame lines are not split between blocks and that the maximum basic block length is 2048 bytes.

When instructed to start transmission, the framestore will transmit the specified area of the frame in a sequence of blocks using the above scheme. An end of frame (EOF) block is transmitted to mark the end of the sequence. The receiver replies to an EOF
by returning a frame acknowledgement (FRAMEACK) block. The transmitter is then free to start transmission of the next frame.

The sequence of data blocks is unacknowledged, and for each block the framestore makes only one attempt at transmission. The framestore will make repeated attempts to transmit the EOF and FRAMEACK blocks in the event of block transmission failures. Failure to transmit an EOF after a maximum number of retries will lead the framestore to suspend transmission on that channel.

The flow control mechanism operates at both the block and frame levels. At the block level, the transmitter waits for a specified time between block transmissions (another parameter specified in the channel descriptor). At the frame level, a transmitter will only resume transmission on reception of a FRAMEACK.

4.4.4.2 The Still Image Transfer Protocol

This is a more reliable protocol in which each data block is acknowledged and retries are made in the event of block transmission failures.
The PDUs are shown in Figure 4.8. The data PDU is similar to that described for the slow-scan video protocol except that a ‘block number’ field has been added. After transmitting a block, the store waits for the receiver to return an acknowledgement block (ACK) before transmitting the next. Failure to receive an ACK within a given time leads the framestore to repeat the transmission (either the original data block or the ACK were lost, but in either event there is no penalty for retransmission since video data is idempotent). After a maximum number of retries, the transmitter gives up and moves on to the next block. In the event of successive block transmission failures, the framestore suspends transmission on that channel (either the network connection has broken, perhaps due to a bridging component failure, or the receiver framestore has crashed).

The sequence of blocks is terminated by an end of still image (EOS) block which is acknowledged by a FRAMEACK.

---

<table>
<thead>
<tr>
<th>DATA</th>
<th>ACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Header (6A90H)</td>
<td>Header (6A10H)</td>
</tr>
<tr>
<td>Block Number</td>
<td></td>
</tr>
<tr>
<td>Line Number</td>
<td></td>
</tr>
<tr>
<td>One Line's Worth of Pixel Data</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Line Number</td>
<td></td>
</tr>
<tr>
<td>One Line's Worth of Pixel Data</td>
<td></td>
</tr>
<tr>
<td>End of Block (1111H)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.8: The Still Image Protocol Data Units
Unqualified numbers denote bit lengths
4.4.4.3 The Framestore Print Protocol

This protocol is a reliable transfer protocol similar to the still image protocol described above. The PDUs are shown in Figure 4.9. The data PDU contains a block number field and an integer number of lines of data, and each block is acknowledged with an ACK. The print service masks out the 6-bit luminance component of each pixel, and performs a linear mapping to an 8-bit value for the benefit of the printer. Successive bytes are simply copied to the printer to generate a PostScript data file (the 'end of line' delimiter in the data PDU is not strictly necessary). The operation is terminated by exchanging END blocks.

4.4.4.4 The Electronic Blackboard Protocol

This is a fairly lightweight protocol with block retries but no block acknowledgements. The PDU is described in detail in [Lu 89]. Flow control is not an issue since blackboard traffic is generally light and sporadic.
4.4 The Video System

4.4.5 System Operation

4.4.5.1 Introduction

Using the video system introduces the notion of ‘logging on’ to a framestore. This is the process by which the user gets a ‘map’ of a framestore screen on his workstation, and is thereafter able to control that framestore by means of the WIMP system. That is, a window is opened for every channel that currently exists on the framestore, and by manipulating these windows, existing channels can be modified, and new channels can be established. Logging on to a framestore in this way gives a user the illusion that he has somehow connected to that store; in fact, this is a purely logical association maintained by the workstation software. See Figure 4.10.

The process of logging on is achieved by the workstation instructing the framestore to return its current state (ie its list of channel descriptors). Each descriptor is decoded in turn to yield the channel parameters corresponding to the connection, and a window is opened on the workstation screen to indicate the size and position of the corresponding video window. In this way, a map of the framestore is built up on the workstation display. The user can inspect additional channel parameters by clicking his mouse over the relevant workstation window.

In the case of a receiver, the map on the workstation corresponds exactly to the images that are being displayed on the screen of the framestore. In the case of a transmitter, the map represents the area(s) of the sampled image(s) that is/are being transmitted.

To give the reader some idea of how the system is used, a number of simple procedures are described below. The general appearance of the user interface is depicted in Figure 4.11 and in Photograph 4.3.
4.4 The Video System

4.4.5.2 Setting up a Video Channel

The user clicks the video icon on the main menu bar with his mouse, and is then prompted for the name of the framestore that he wishes to act as transmitter. [Clearly, this step requires that the user knows something about the network configuration. In particular, he needs to know which framestores have cameras attached and which can therefore act as transmitters. More on this later.] The user is then logged on to his chosen framestore, and he is able to inspect any existing channel connections by clicking the mouse over the workstation windows.

By clicking the new channel icon, a new window pops up on the screen. The user positions and scales this window as required in order to specify the area of the sampled image that he wishes to transmit. By clicking his mouse over the new channel window, he is allowed to override various default parameters by way of sub-sampling factors and delay times. Once he is satisfied with all these parameters, he clicks the OK
Photograph 4.3: The User Interface to the Video System

This photograph shows the screen of the A500 workstation running the video system control software. The user interface is based on the Arthur WIMP module. The main menu bar is shown on the right of the screen, and status messages are shown at the bottom. This user is currently 'logged on' to the framesetore called 'Plexus' at Loughborough. This framesetore has one active channel represented by the single 'channel window' in the bottom right-hand corner (partially obscured by the 'channel information' window. The size and position of this channel window corresponds to the size and position of the video window on the framesetore itself.

By clicking the mouse over the channel window, the user has opened up the channel information window shown just left of centre. This window gives additional information about the channel. For example, Plexus is receiving video data from the transmitter called 'Nexus' (also at Loughborough). The pixel pitch value of 2 means that the image is being stretched in the x direction by a factor of 2.
icon on the main menu bar. He is then prompted to select the name of the store that is to be his receiver.

As before, the user is logged on to the selected framestore, and his workstation displays a map of the screen of the receiver. Once again, he may position the new channel window to cover the area that the new image is to occupy. He can choose to enlarge, reduce, stretch or squash the image by an integer factor. (This alters the pixel and line pitches on the receiver side). When he is happy, he clicks OK, thus completing the user dialogue.

The workstation then goes about the business of setting up the connection. The parameters that the user has supplied are passed to the transmitter as part of the call set-up SSPREQ. The transmitter makes the connection, passing these same parameters to the receiver as part of the OPEN request. The receiver sets up an identical copy of the transmitter's channel descriptor. The receiver channel ID is passed back to the transmitter in the OPENACK, and the transmitter and receiver channel IDs are returned to the workstation in the SSPRPLY. By default, the transmitter is instructed to start transmission as soon as the connection has been established.

4.4.5.3 Setting up a Still Image Channel

The procedure for setting up a still image channel is very similar to that described above. The difference is that once the channel has been established, the image is transmitted only once before the channel goes dormant. Transmission can be restarted at any time provided that the network connection is still open (ie provided that the connection through bridging components has not timed out, typically a time in the order of minutes).

4.4.5.4 Setting up a Blackboard Channel

The procedure for setting up a blackboard channel is much the same as that described above. This time, however, the workstation itself is the transmitter, and the user is prompted only for the name of the receiver. As before, the blackboard can be arbitrarily scaled and positioned. The blackboard software supports line graphics, square and circle generation, flood fills, ASCII text, etc. Refer to [Lu 89] for details.

4.4.5.5 Modifying an Existing Channel

It is quite likely that a user will wish to modify an existing channel. For example, he might wish to speed up a transmission, or he might want to reposition or rescale an image on the receiver. He can do this by selecting the modify channel icon over the appropriate channel window. A new modified channel window appears which replaces the previous channel window. This can be used as before to set up the new
channel parameters. When happy, the user clicks OK, and the workstation sends the necessary instructions to the framestores causing the pair of channel descriptors to be modified (see Appendix A for details of these SSPREQ formats).

4.4.5.6 Printing an Image

To print an image, the user clicks the print icon on the main menu bar. A print window appears on the screen, and the user is prompted to position this window over the area to be printed. When happy, the user clicks OK and the workstation generates the necessary SSPREQ to the framestore. By default, the workstation uses a pre-defined print service name. Also, by default, the image is scaled and oriented to appear as large as possible on an A4 sheet.

4.4.5.7 Miscellaneous Procedures

There are other simple procedures for suspending transmission, restarting transmission, closing down channels, grabbing frames, generating colour bars, clearing areas of the screen, resetting the framestore, etc. For details, see the framestore interface specification in Appendix A.

4.4.6 System Performance

4.4.6.1 Slow-scan Video Performance

The performance (in terms of frame rate) of the slow-scan video system is dependent on a number of parameters. The processor bandwidth of the receiver framestore is the limiting factor on a single ring, and the maximum achievable data rate is about 500 kbps. Although the transmitter framestore can run at higher rates (the VMI-1 can operate at about 800 kbps), the receiver cannot match this speed (blocks are simply discarded and the image becomes distorted). Furthermore, the CR portal cannot handle data at rates greater than about 500 kbps - see section 4.5.] In order to slow down the transmitter, a fixed delay (in the order of milliseconds) is inserted between consecutive basic blocks during video transmission. The size of this delay is specified at call set-up time, and this value clearly affects the overall frame rate. An excessively large value will slow down the transmitter more than is necessary, whereas an insufficiently large value will lead to image distortion at the receiver.

The other factor that influences the frame rate is the quantity of data per frame. A full size, full resolution picture represents $512 \times 512$ pixels, each pixel coded to 16 bits. The transmission time for this sort of picture is about 8 seconds. However, the frame size and resolution (at the transmitter and receiver) can all be specified at call set-up time, and in practice higher frame rates are achievable. A popular 'trick' was to sub-sample the entire transmitter frame by a large factor (say 8) but to display the data
in a small (but full resolution) window on the receiver. This has the effect of improving the frame rate by a factor of 8 while still maintaining a reasonable level of quality. Further improvements in frame rate can be achieved by transmitting only a section of the transmitter frame or by further increasing the sub-sampling factor, but there is clearly a trade-off between frame rate and picture quality.

4.4.6.2 Still Image Transfer Performance

The same comments with respect to image size and resolution apply to the still image transfer service. However, the still image service uses a more reliable communication protocol involving block acknowledgements rather than fixed delays between blocks, and this means that frame transmission times are longer. A full size, full resolution still images requires about 12 seconds transmission time between framestores on a single ring. This time is not significantly increased for transmission over the wide area network (in particular, the portal has negligible effect).

4.4.6.3 Electronic Blackboard Performance

The electronic blackboard service relies on the occasional transmission of fairly short basic blocks (a few hundred bytes every few seconds) and there are no serious performance issues (either locally or remote) relating to such volumes of traffic.

4.4.6.4 Protocol Performance

The slow-scan video protocol is a simple lightweight protocol which seemed to perform quite well. As expected, during times of great network or processor activity, occasional video data blocks were lost, thereby degrading the appearance of the displayed image. Lost blocks were quite obvious due to missing sections in the image.

The more robust still image transfer and image printing protocols proved to be more temperamental. The basic scheme is that a transmitter sends a block of data and waits for a reply (ACK) before sending the next. If a reply is not received within a certain time, the transmitter repeats the transmission. Duplicate blocks are ignored by the receiver. After a fixed number of successive block transmission failures, the transmission is abandoned. Block numbers are used, since, in general, the data is not idempotent (eg when printing).

The problems experienced were due to the choice of timeout values. If the transmitter timeout for receiving an ACK is too short, the transmitter retransmits unnecessarily. The receiver ignores repeated blocks, and the transmitter will eventually receive the original ACK. The protocol therefore works, but it makes inefficient use of network bandwidth. This is the case when the transmitter ACK timeout is too short with respect to the receiver block processing time (which itself will depend on receiver activity).
4.4 The Video System

Upshot of this is that the choice of the ACK timeout value is critical to the performance of the protocol.

Another timing problem was observed when conducting experiments through the portal. The ACK for the first data block was always lost (for reasons still not understood). If the receiver data block reception timeout is shorter than the transmitter ACK reception timeout, the receiver will abandon reception before the transmitter repeats the transmission. This leads to a fatal communication failure. If, however, the first data block is successfully retransmitted, temporary deadlock results since the transmitter will be waiting for an ACK that the receiver has already given. The deadlock is broken when the transmitter eventually gives up on the first block, and transmission continues as normal from the second block onwards. Once again, this illustrates the sensitivity of this protocol to the value of timeouts, and its vulnerability to block loss. The upshot is that this is a poor reliable transmission protocol; a better alternative is the Unity RPC transport protocol mentioned in chapter 3 and implemented in the CFR office described in the following chapter.

4.4.6.5 Printing Performance

As mentioned above, the framestores have a remote image printing capability based on the use of a network PostScript laser printer (the Apple LaserWriter Plus). The problem was that printing an image was a time consuming business. One reason for this was that the interface between the Tripos printer server and the printer was via an RS-232 serial link operating at a speed of 9600 baud. A full size, full resolution image is 512 pixels by 512 lines by 6 bits of luminance data per pixel (expanded to 8 bits for the benefit of the printer), representing 256 kbytes of raw data. The PostScript language is entirely ASCII based, so 512 kbytes of ASCII data needs to be transmitted down this link for a full size image print. This requires over seven minutes of transmission time. In practice, total printing time for a full size image is about ten minutes partly due to the network transmission delay between framestore and print server, but mostly due to processing overheads in both the printer server and the printer itself.

In an attempt to improve performance, the Tripos printer device driver was modified to drive the RS-422 synchronous serial port on the printer. The processor board used in the printer server features a Thomson Z8530 serial communications controller which can be configured to drive an RS-422 line at speeds of up to 64,000 baud. Although not documented by Apple, the LaserWriter features the same controller, which likewise may be configured to run at the same speed.

Successful tests were conducted using the new printer driver, but unfortunately they gave disappointing results. Although the line transmission speed was increased by a factor of nearly seven, the overall printing time was virtually unchanged. It appears that the printing speed is as much limited by the printer processor bandwidth as by the
speed of the communication link; the data was being transmitted in short bursts followed by long delays while it was being processed by the printer. The only solution would be to use a more powerful printing engine in conjunction with a faster communication link.

4.4.7 System Evaluation

An early version of the video system was demonstrated at the Alvey UMIST demonstration, and there were several lessons learned from ‘hands on’ experience with the system and from user comments.

4.4.7.1 User Interface Design

The general feeling seemed to be that a window-based user interface was an improvement over previous menu systems. It is generally accepted that the use of a mouse and icons is preferred to the use of a keyboard and menus, especially by ‘computer laymen’.

One criticism was that simple procedures (like setting up a video connection) were made too complicated. This seems to be an almost inevitable consequence of providing full flexibility to the user; the result is that the user is required to specify a large number of parameters, some of which may be of no particular interest. The other extreme is to remove this choice from the user and to build in fixed values for these parameters. A better compromise would be to make sensible assumptions for the default value of all parameters, while preserving the user’s ability to make alterations if required. This could be achieved by using a series of forms or templates which are pre-filled with default values, but which can be edited by the user if desired.

One point to emphasise is that although a workstation window can be used to define the size and position of an image on a framestore, the video data itself is not displayed inside the workstation window. Video data is displayed inside a video window on the (separate) framestore display in a position corresponding to that of the workstation window. A system which includes video data within workstation windows is described in the next chapter.

4.4.7.2 Error Recovery

The initial framestore software (as demonstrated at UMIST) was very intolerant of network failures. In fact, a framestore would completely reset itself if it detected a protocol violation or a transmission failure on any of its active channels, with the unfortunate result that all other active channels were lost. This proved an embarrassment when procedural errors were made by the user during call set-up (eg trying to connect to
a non-existent framestore), and when there were network component (especially portal and ramp) failures.

The solution adopted was for a framestore to simply abandon transmission or reception on an invalid channel, but to continue to process all other channels. This means that in theory a framestore can get clogged up with redundant channel descriptors, although in practice this is not a problem.

Another general comment is that the user is not informed of any component communication failures. That is, once a channel has been successfully established, the user receives no notification of its status. It would be possible to add an error reporting capability, but this raises the question of to whom the error should be reported. The question of ownership is common to all office components and is considered in chapter 6.

4.4.7.3 Control and Management

The framestore control strategy is one of potential anarchy. There is no logical binding between framestores and workstations; there is no concept of 'ownership' or 'domains of control'. Any workstation on the network can log on to any framestore and take control. This presented problems at the UMIST demonstration when a user at one site could take over (and even reset) a framestore at the remote site. The fragile control strategy relies on a 'gentleman's agreement' over who should control what, and nothing is enforced in software. There is an obvious need to somehow manage the flow of control; this is a general problem in a distributed office and it is addressed in chapter 6 of this thesis.

4.4.7.4 Distribution Transparency

Another issue is that of distribution transparency. The video system described here requires a user to have knowledge of the location and attributes of framestores on the network. (For convenience, this knowledge is actually built into the workstation control software, but this is clearly not where it belongs.) For example, to start transmission to a peer user at site camb, the local user needs to know the name of the peer user's receiver framestore (ie one with an attached monitor) at Cambridge in order to be able to construct the required peer service name (of the form camb<service><machine> - refer to Appendix A for information on the general structure of service names).

A higher level of abstraction requires network support for binding based on logical instead of physical names [Chapman 89]. A physical name relates to network topology and is usually the basis for routing. On the other hand, a logical name may be arbitrarily selected on the basis of user name, geographical location, or organisation-
4.5 Experiments over the Wide Area Network

A network service (often provided by a 'nameserver') is then required to map the logical name space to a physical name space.

Location transparency can be achieved by using the CR nameserver to map an appropriate logical service name (say, of the form `<service>-<TimsFramestore>`) to the corresponding physical network address, but only by accepting a static mapping between logical and physical names. There is an apparent need for a network service to provide some measure of migration transparency. This would allow client software to locate a framestore service using a logical name (say based on the owner of the framestore) without needing to be bothered about its current whereabouts on the network. Once again, this is a general requirement in a distributed office, and the problem is considered further in chapter 6.

4.5 Experiments over the Wide Area Network

The CR multimedia office was successfully demonstrated at the 1987 Alvey exhibition at UMIST, Manchester. It was shown interworking with a similar office at Loughborough over the Alvey High Speed Network. Applications that were demonstrated included slow-scan video, still image transfers, the electronic blackboard service and the real-time voice system. The video system was under the full control of the workstation, and the user interface to this and the electronic blackboard system was implemented in the Arthur WIMP environment.

At the time of this exhibition, the wide area network infrastructure was not fully in place. In particular, the 'pilot' Unison exchanges that were available lacked the exchange management services described in chapter 3. The CR portal was an adaptation of the CR bridge described in [Leslie 83], and its performance turned out to be rather limited by its CR interface. In practice, the portal bandwidth was limited to about 500 kbps, less than half the maximum point-to-point bandwidth available on a single ring, and well below the maximum wide area bandwidth available from the ISDN.

A further shortcoming in the original implementation of the portal (that was particularly significant for wideband speech) was its poor handling of large numbers of small basic blocks (due to a deficiency in the CR driver software). This meant that although the data rate could be quite low (nominally 64 kbps for wideband speech), communication was severely disrupted due to the large number of (small) packets involved. This problem was rectified after the UMIST exhibition. [Although plans were afoot to further improve the performance of the CR portal (by using a superior CR interface [Tripp 85]), the diversion of attention towards the emerging CFR office environment pre-empted such a move.]
On subsequent occasions, applications experiments were conducted between Loughborough and Cambridge, and Loughborough and the Rutherford Appleton Laboratory. These experiments included a successful demonstration of the wideband speech system using improved portal software. Remote logins were accomplished between sites, and text files and bit-mapped images (from the framestores) were printed remotely. Furthermore, many tests were conducted at Loughborough over the wide area network in 'loopback mode' (this allowed experimentation without suffering the inconvenience of transporting equipment between sites).

Once the early implementation of the portal had been improved, the performance of most applications over the wide area network was comparable to that over a single ring (assuming that sufficient ISDN bandwidth had been allocated - an insufficient U-channel size resulted in serious delays due to back-pressure being applied by the exchange). Although no single application generated data at a rate greater than the maximum portal bandwidth, the portal was certainly the overall performance bottleneck when conducting experiments involving parallel streams through the exchange. Apart from this, real-time voice was the only application that was significantly affected by operation over the network due to the delays imposed by the bridging components and by the ISDN itself.

4.6 Summary

This chapter describes an experimental multimedia office based on the Cambridge Ring local area network. It is based on a loosely coupled collection of components with the network providing the common communication medium. As such, it is based on the open (or distributed) workstation architecture proposed in chapter 2. Although the CR office meets many of the requirements of an integrated office in terms of the media it supports, it lacks any coherent mechanisms for the integration of these media at the user level.

A video system comprising a collection of framestores under the control of a workstation has been described in some detail. This supports the display of slow-scan images, still pictures and electronic blackboard data in windows on a framestore display. The user interface to this system is based on the use of windows on the workstation which map onto those of the currently selected framestore. Use of the video system highlighted the problem of the management of control in a distributed office, and the problem of locating services based on the name of a peer user. These two problems turn out to have a common solution based on the definition of 'domains of controls' and the dynamic binding of office users to such domains. These ideas are further discussed in chapter 6.
Chapter 5
The Cambridge Fast Ring Multimedia Office

5.1 Introduction

During the course of the Unison Project, there was a shift in development effort from the original Cambridge Ring (CR) to the newer Cambridge Fast Ring (CFR) as the basis for the multimedia office. This was made possible by the availability of serial CFR equipment (from Caman Systems Ltd) allowing client CFReS to be installed at each of the sites. The adoption of CFR technology in the office provided network performance benefits for real-time services in particular and office applications in general. It also provided an ideal opportunity for developing a new local area network architecture drawing upon the experience gained with the Cambridge Ring. An early version of the CFR office was demonstrated at the 1988 Alvey ITEX exhibition in London where it was shown interworking with a similar office at Loughborough.

Much development work was necessary to take advantage of this new environment. The network architecture initially conceived for the exchange CFR was extended to include the new generation of client CFReS. A CFR portal was designed and implemented (based on Inmos Transputers) at the Rutherford Appleton Laboratory (RAL) in order to provide client access to the Unison network. The secretary service was expanded to include client networks, and this became the universal mechanism for establishing associations, both local and remote, across Unison.

Apart from the network infrastructure developments, a new generation of office components was required for applications experiments. A new telephone system was developed at RAL, and wideband speech and video framestore systems were developed at Loughborough. All of these office components were based on the Inmos Transputer, with Occam being used as the programming language. Given that Unity RPC was the chosen mechanism for interacting with the secretary, a major milestone in the development of the CFR infrastructure was the RAL implementation of Unity RPC in Occam. This was necessary for the portal and the all the Transputer-based application components.

This chapter gives a brief overview of the CFR multimedia office. It then goes on to consider the issue of control in this new environment, and outlines the work done to provide workstation control of the office components. Given that the workstation has a more fundamental role (in terms of control) in the CFR office than in the CR office, the design of the user interface to these components assumes a greater importance. This chapter describes the design and implementation of a workstation window environment intended to form the basis for this user interface. It goes on to describe a document scanner system and a workstation video system, both of which make use of
the windowing facilities available on the workstation. Some video multicast experiments are then discussed as an illustration of the role that a multicast mechanism might play in an integrated office. Finally, the performance of the office over the wide area network is discussed.

The majority of the Tripos software environment described below is due to the work of Ian Wilson, formerly of the Cambridge Computing Laboratory and now of the Cambridge Olivetti Research Laboratory (ORL). In particular, he is responsible for the ARM Tripos system, the Tripos implementation of Unity RPC, the Tripos Remote Filing System, and much of the 68000 series Tripos environment. I would like to record my gratitude for his work and help.

5.2 Overview of the Office

5.2.1 Introduction

As with the CR office, the CFR office comprises a collection of multimedia components which collectively provide a variety of services to the user. These services include dis-

Figure 5.1: The Cambridge Fast Ring Multimedia Office
5.2 Overview of the Office

Distributed computing facilities, traditional connection-oriented data services, and new real-time multimedia services. A schematic diagram of the CFR office is given in Figure 5.1.

The design of the office is based on the distributed office architecture discussed in chapter 2. Each component has an independent network interface, and is designed to handle one medium well, rather than many media badly. The intention was to allow all office components to be remotely controlled from a workstation. This removes complexity from the components themselves and provides maximum flexibility for the design of the control mechanisms and the user interface. The policy of workstation control does not preclude the possibility of a component offering an alternative mechanism for its use; for example, the wideband speech system supports a keypad which can also be used for call set-up. The desire for workstation control is in line with the concept of distributed control discussed in chapter 2.

Another design goal was to keep the functionality of each component to a minimum, but to provide the 'hooks' so that higher level applications software could add value to the basic service. For example, individual telephones are not designed to handle such things as call queueing or call diversion, but a higher level call management service could do just this.

5.2.2 Office Components

5.2.2.1 The Workstation

As in the CR office, the Acorn A500 serves as the office workstation. It includes a CFR interface module, and runs the ARM Tripos operating system. [Although Unix has now been ported to Acorn RISC machines, this was not completed until after the project.] The CFR workstation provides a more sophisticated working environment than that available with the CR workstation in terms of the performance of the network interface and the multi-tasking capabilities of the operating system. A feature of the ARM Tripos system is that it relies on a remote fileserver; the intention is that any diskless ARM can support the system.

Tripos is a real-time multi-tasking operating system primarily used for experimental purposes. Although Tripos supports several programming languages including C and Modula-2, the native language is BCPL [Richards 80]. Much of Tripos itself is written in BCPL (except for the kernel and most device drivers), and this was an influence in choosing BCPL as the implementation language for Tripos office applications. However, the most important reason for choosing BCPL was that an implementation of Unity RPC was available in that language [Wilson 87a] as well as a BCPL RPC stub generator [Wilson 87b] from very early on in the project. As mentioned above, Unity RPC is an essential part of the CFR protocol architecture, and it is convenient to base applications
development on the same communication mechanism. In addition, the RPC package provides a useful multi-threaded programming environment which isolates the application from the complications of handling the network.

5.2.2.2 The Video Framestore

The video framestore was designed to take advantage of the higher bandwidth available from the CFR to enable the transmission of high resolution colour images over the network. The performance of the network interface and improvements in host processor bandwidth offered the promise of higher data rates than those available from the CR framestore.

Unlike the CR framestore, the transmitter and receiver framestores are implemented as separate units. Each unit comprises a number of custom designed boards based on the Inmos Transputer as the processing engine. The receiver features a commercially available display processor that supports windowing. The framestore handles images up to a resolution of 512 * 512 and with a maximum depth of 16 bits. The analogue circuitry is switchable between YUV and RGB signals, allowing experimentation with monochrome pictures and image coding techniques. The transmitter has been designed to accept an optional hardware compression board to support a block-conditional transmission scheme. Refer to [Lu 89] for a full description of the CFR video framestore.

The CFR framestore is a more 'lightweight' system than the CR framestore in terms of functionality and complexity. A fundamental design goal was to build a relatively simple system which would still meet the requirements of colour image transmission but which would be easier to design and maintain, and which would be less expensive.

As with the CR framestore, the CFR framestore is fully programmable in terms of the services provided to the user. The software is written in Occam, and was developed and debugged using the Transputer Development System. Production code is blown into EPROM. The framestore offers a control and a data interface to the network. The control interface is exercised by a control entity (usually a workstation) for call setup. The data interface is for access by peer framestores for data transmission.

5.2.2.3 The Telephone

The toll-quality telephone system handles standard 64 kbps PCM speech and was designed to emulate a conventional telephone system. In fact, much of its complexity is due to the support for dialling and ringing. The system is based on a CFR station, a standard host Transputer card, and a custom-built codec board.
A familiar telephone handset is used for control. No work has been done to provide an alternative mechanism for call set-up by using the workstation. However, this work has been done for the wideband speech system (see below), and there are no technical barriers to prevent this from being incorporated into the telephone.

5.2.2.4 The Wideband Speech System

The wideband speech system uses the same SB-ADPCM codecs described in chapter 4. The system comprises a CFR station, a standard host Transputer card, and a Transputer-based X.21 codec interface board. A desktop microphone and a pair of loudspeakers replace the conventional handset used by the toll-quality system. The intention is that this system will be the basis for teleconferencing experiments, and a conventional telephone handset is seen as a hindrance to such activities.

Control is either from the workstation via the network control interface, or by using an (optional) keypad. Support for remote control enables the design of a more sophisticated user interface than that available from a handset or keypad. For example, workstation control software can automatically interact with directory services to remove the need for a user to remember an individual's telephone number. Refer to [Hardman 89] for a full description of the wideband speech system.

5.2.2.5 Tripos Servers

A Tripos server is a general purpose VME-based 68020 machine running the Tripos operating system. Tripos servers are used in the office to host some of the network services such as the secretary, directory, bootserver, etc. They also host some application services such as a printer spooler and the scanner system described later in this chapter.

In practice, a single Tripos server is usually devoted to the role of fileserver for the ARM Tripos systems and for other diskless Tripos systems. This is usually a 68020-based machine with a large SCSI disk. The performance of the filing system is very good when the fileserver and host are on the same CFR. For example, the disk access time from a 68020 client to a remote SCSI disk is shorter than to a local SASI disk.

On the question of booting, although a CFR boot protocol has been defined and a bootserver implemented (by ORL), only the ARM and 68000 series machines can boot across the network. Production office component software (written in Occam for the Transputer) is contained in on-board EPROM. This has advantages in terms of simplicity and restart speed, but makes the distribution of new code much more difficult. The problem is that the CFR boot protocol was never implemented in Occam to provide a bootstrap EPROM for Transputer-based components.
5.2 Overview of the Office

5.2.3 Office Services

5.2.3.1 General Workstation Services

In principle, the workstation would offer the full range of standalone office services such as word processing, spreadsheets, etc. In practice, such business software is not available for the experimental workstation environment based on Tripos. However, good support is provided for network services such as remote printing, remote file access, remote job execution, remote login, etc. These facilities (developed at ORL) are similar to those available in a networked Unix environment.

The availability of Unix for the workstation operating system would have provided the same sort of network services as available on Tripos. In addition, Unix provides a standard environment which supports many third-party software products. As such, Unix appears to be an attractive candidate for the role of workstation operating system. [Note, however, that an argument in favour of Tripos is its support for real-time applications such as the video system described in section 5.6. Unix does not offer real-time support]. However, it is interesting to note that the traditional view of Unix as being an operating system for scientists and engineers has meant that there has been little interest in developing business applications based on this platform. This situation may change if Unix continues to make inroads into the office systems market.

5.2.3.2 The Video Service

The video service allows high resolution colour images to be transmitted across the network. The data rate is limited to about 1.5 Mbps, giving a full resolution frame rate of approximately one third a frame per second. A framestore may have multiple connections to other stores, and a receiver framestore can display data in separate windows. The operation of the framestores is under the full control of the workstation. Refer to [Lu 89] for further information on the video service.

5.2.3.3 Voice Services

The toll-quality speech system provides a simple telephone service based on the use of dial-up connections. The wideband speech system provides a high quality voice service suitable for teleconferencing activities. The wideband system can be controlled from the workstation. Refer to [Hardman 89].

5.2.3.4 The Scanner Service

The scanner service allows high resolution paper-based images to be transmitted across the network, filed on disk, and printed on a network printer. This service is fully described in section 5.5.
5.2.3.5 The Workstation Video Service

The workstation video system allows low resolution monochrome images to be transmitted across the network and displayed in windows on the workstation display. This service is fully described in section 5.6.

5.3 Component Control

5.3.1 Basic Scheme

The component control mechanism is based on the use of the Unity RPC mechanism referred to in chapter 3. Every component is expected to offer a well defined RPC control interface to the outside world. The component advertises a control service with the secretary, and a controller must first bind to this service in order to use the interface. The term 'bind' is used here to refer to the process during which a client creates an association with a named service; this is achieved by making an association.create RPC to the secretary in the manner described in chapter 3. Once a successful binding has been established, the workstation is able to control the component over this control association.

As well as offering a control interface, each component is required to provide a data interface for access from client components. Peer components exchange data over a data association. Communication across the data interface is preceded by a suitable exchange across the control interface. The general scheme is shown in Figure 5.2.

![Diagram of the Basic Control Scheme]

Figure 5.2: The Basic Control Scheme

The data association is set up after a suitable RPC exchange over the control association.
The problem of locating the appropriate control service and the management of connections between components that support multiple streams are complicated topics, and are fully considered in chapter 6. This section describes the prototype control mechanisms implemented in the CFR office.

5.3.2 Framestore Control

A simple interactive program has been written to allow a Tripos machine (such as the ARM workstation) to control the video framestore system. This allows connections to be set up between framestores giving the user full control over parameters such as image size and position. It also allows video communication to be terminated.

The program works by first creating a control association with the control service video-c of the named framestore. This then becomes the initiator framestore and must be configured as a transmitter. Once the user has specified the connection parameters, the workstation sends a start RPC down this (control) association, supplying the location of the receiver framestore and the connection parameters. On receiving the start RPC, the initiator creates a data association with the data service video-d of the specified peer, passing on the connection parameters. If this succeeds, the initiator framestore completes the start RPC with a success notification. Note that the association.create RPC used to establish the data association is nested within the start RPC from the initiator workstation.

For example, to set up a connection between framestore fs00 at domain lut-blue and framestore fs03 at domain ral-blue, a user would first type "select lut-blue*fs00". This would create a control association with the video-c service on fs00. He would then type "start peer ral-blue*fs03", and be prompted for connection parameters such as speed, resolution, size and position. Once these are supplied, the workstation sends a start RPC down the control association with the location of the peer and the connection parameters as arguments. The framestore fs00 at lut-blue then attempts to create a data association with fs03 at ral-blue, and reports the outcome to the workstation by completing the start RPC with a suitable return code.

To close down all the connections to or from a framestore, the workstation sends a stop RPC down the control association to that framestore. Peer framestores are not explicitly informed of closures - a ‘dead man’s handle’ will time out inactive data associations. [Transmitter and receiver framestores periodically exchange alive packets on every data association to confirm that each association remains valid. This is necessary since UDL gives no guarantee of end-to-end delivery, and it is used to detect redundant connections due to peer closures.]

There should really be some mechanism for selectively closing individual connections, but no such mechanism has been implemented. The problem is that this requires worksta-
tion knowledge of the identifier allocated by a framestore for every data association, and it is not obvious how the workstation should acquire such information. This is a general problem for multi-stream components, and it is discussed further in chapter 6.

5.3.3 Wideband Speech System Control

A similar interactive program has been written to allow a Tripos machine to control the wideband speech system. The control of this system is comparatively simple, since there are no parameters governing a speech connection, and a speech system is a single-stream device.

The control association is established between the workstation and the \textit{wbs-c} service of the initiator speech system. To set up a call, the workstation sends a \textit{start} RPC down this control association, quoting the location of the peer wideband system. The wideband system then attempts to create a data association with the \textit{wbs-d} service on the specified peer. The \textit{start} RPC succeeds if the connection is successfully established, but fails otherwise.

To terminate the call, the workstation sends a \textit{stop} RPC down the control association. An in-band \textit{stop} packet is then passed down the data association to terminate the peer speech system.

5.3.4 Traffic Generator Control

A similar program was also written to provide remote control of the traffic generators used for network performance analysis [Siddiqui 89].

The workstation first makes an association with the \textit{gen-c} service of the named traffic generator. It then sends a \textit{start} RPC down this control association, quoting the name of the peer traffic generator. On receiving the \textit{start} RPC, the initiator traffic generator attempts to set up a data association with the \textit{gen-d} service of the peer traffic generator. If the connection is established, the \textit{start} RPC completes with a success notification. There is no concept of stopping a traffic generator, since a test only runs for a finite time.

5.4 The Workstation Window Environment

5.4.1 Introduction

There were various incentives for providing some sort of window environment for the ARM Tripos system. The Arthur WIMP module had previously been used to provide a window-based user interface to the CR video system (chapter 4), and it was considered
desirable to have such a user interface for the control of CFR applications. Also, early experiments with workstation video had suggested that a video picture within a workstation window would be a worthwhile objective.

Tripos access to the underlying Arthur WIMP module seemed to offer the simplest solution. An alternative strategy would have been to implement a native window package for Tripos, but this was quickly ruled out on the basis of complexity and effort. Another option would have been to port a third-party window system to the ARM Tripos platform, but once again this was ruled out on the basis of effort. [Since this decision was made, there has been a successful port of the X Windows environment to ARM Tripos (at ORL), but this came too late to influence the development work described in this chapter. However, if such a standard environment had been available, it would have been adopted in preference to the solution described here. The Tripos X system was successfully installed on the A500 and tested over the Unison network (by the author) after the end of the project.]

5.4.2 Access to the Arthur WIMP Module

Use of the Arthur WIMP module is possible because the ARM Tripos environment exists alongside Arthur, rather than replacing it (Figure 5.3). Indeed, several Tripos device drivers (eg screen and console) rely on Arthur input/output primitives rather than having to talk directly to the hardware. To avoid language dependency, access to Tripos kernel primitives is provided by software interrupts (SWIs). Likewise, access to Arthur primitives and relocatable module routines (such as those for the window manager) is via the same mechanism. Therefore, since the Tripos kernel can be configured to pass unknown software interrupts to the resident Arthur software interrupt handler, simultaneous access can be provided to both Arthur and Tripos.

An ARM assembler library has been written to provide BCPL access to the Arthur window manager primitives. It actually maps BCPL global vector locations to window manager SWIs. The BCPL compiler conforms to the ARM Procedure Call Standard [Acorn 87b] in mapping BCPL procedure arguments and the stack pointer to ARM registers. The assembler library then maps these incoming BCPL parameters to registers appropriate...
5.4 The Workstation Window Environment

to the particular SWI call, performs the SWI, and then maps the results to outgoing
BCPL parameters. Note that use of the Arthur window manager in this way re-
quires that its relocatable module be loaded before the ARM Tripos system is started.

5.4.3 Detecting WIMP Events

The Arthur WIMP module has been designed to work in a single-threaded environ-
ment in which the client has full control of the machine. Having defined and
opened one or more windows, the client is expected to sit in a tight polling loop wait-
ing for WIMP events. WIMP events can be generated by such things as mouse clicks
or key presses. Incoming WIMP events must be dispatched to suitable service routines,
and control should be quickly returned to the central polling loop. When there is no
WIMP activity, the client is allowed to perform background tasks provided that, once
again, control is quickly returned to the polling loop. Protracted service routines or back-
ground tasks result in a sluggish response to WIMP events which can be most distract-
ing to the user.

Having solved the problem of access to the Arthur WIMP module, the next problem was
how to drive the module in the multi-tasking environment of Tripos. It is not permis-
sible for a Tripos client to sit in a tight loop polling the WIMP module in the manner
described above since such a client would hog the processor to the exclusion of
other Tripos tasks.

The solution adopted was to make use of a customised mouse device driver to detect
potential WIMP events. The mouse driver is programmed to detect mouse events, which
are defined here as mouse button clicks or mouse movements. A mouse event
may also cause a WIMP event; for example, a mouse click over a window title bar will
also generate a WIMP event, whereas a mouse movement over the background area
will not cause a WIMP event. A dedicated Tripos WIMP task has been designed
to perform the WIMP module polling function described above, but only when a mouse
event is detected by the mouse driver. If the mouse event does correspond to a WIMP
event, this is detected by the polling loop. Otherwise, the WIMP task suspends it-
self waiting for the next mouse event, thus freeing the processor for other tasks.

This is achieved in the following way. The WIMP task starts up by queueing a Tripos
packet on the mouse device driver. The task is then automatically suspended by
the Tripos scheduler until a mouse event causes the packet to be returned. When this
happens, the WIMP task enters its polling loop checking for WIMP events. Any detected
WIMP events are dispatched to appropriate service routines (see next section) before the
packet is requeued on the mouse driver and the WIMP task is again suspended.

There are circumstances when a WIMP event is generated without a corresponding mouse
event. This happens when a client makes a direct call to the WIMP module which
results in the synchronous generation of a WIMP event. Most WIMP events are asynchronous, and are the result of an external event such as a mouse click. In order for such a synchronous event to be detected and dispatched to an event handler, a simple mechanism is provided allowing a client to explicitly force the WIMP task to enter its polling loop. This is by means of the WIMP task packet interface referred to below.

5.4.4 Dispatching WIMP Events

The next problem was how to dispatch an event to the appropriate client event handler. The solution adopted was to implement a callback routine interface to the WIMP task. A client is required to register the address of a callback routine for every window and WIMP event in which it is interested. This event handler is then run automatically by the WIMP task whenever the corresponding WIMP event is detected.

This is achieved in the following way. When a client starts up, it registers a window directly with the WIMP module. That is, the client calls the WIMP module supply-
5.4 The Workstation Window Environment

ing various parameters defining the window attributes, and on success the WIMP module
returns a handle identifying the window. The client then registers event handlers for
this window with the WIMP task, which are then called asynchronously. The WIMP task
maintains a matrix of client event handlers for each window, and an appropriate hand-
ler is called when the corresponding WIMP event is detected.

Client access to the WIMP task is via a Tripos packet interface. This interface sup-
ports the registering and withdrawal of event handlers, and miscellaneous operations such
as forcing the WIMP task to enter its polling loop. A block diagram of the system
is given in Figure 5.4.

The procedure to open a window is as follows. A client registers the new window
directly with the WIMP module, and is returned a handle for the window. It then registers
a set of event handlers for this window with the WIMP task. The client then
makes an open window call to the WIMP module requesting that the window be dis-
played on the screen. This causes the WIMP module to draw the window outline, and
to generate a (synchronous) redraw window event which must be trapped by the WIMP
task. Since this WIMP event is not the result of a mouse event, the client must make
a further call to the WIMP task explicitly requesting that it enters its polling loop. The
WIMP task then calls the redraw window event handler, which causes the contents
of the window to appear on the screen. Additional redraw window events are generated
for each visible rectangle in the new window (the window may be partially obscured);
the client redraw window event handler must respond to such events by filling in the
specified rectangles.

In general, moving or enlarging a window generates an open window and subsequent
redraw window events. A client responds to an open window event by calling a WIMP
module routine to re-display the window outline in the new position. A client responds
to each redraw window event by redrawing the contents of the window (in an applica-
tion-specific way) for each visible rectangle returned by the WIMP module.

5.4.5 Event Handler Run-time Environment

In Tripos, each task has a corresponding Task Control Block (TCB) which is used
by the kernel for scheduling and for access to various task-specific data structures. One
of these data structures is the BCPL global vector which supports the linking of separate-
ly compiled BCPL modules and the sharing of variables between procedures within a
task.

One problem with the callback mechanism described above is that the event handlers
are run in the environment of the WIMP task rather than that of the client task. That
is, the active Task Control Block is that for the WIMP task, and not for the client task.
This means that it is not possible for an event handler to access directly the global
variables used by the client task, and this prevents the simple sharing of data between a client and its event handlers.

The solution adopted is for the client to register with the WIMP task the address of global variables which require to be accessed by any of the event handlers. This set of pointers is copied to the same location in the WIMP task global vector as in the client task global vector, thus allowing an event handler to access client variables by means of indirection. This is illustrated in Figure 5.5. This mechanism requires that

![Figure 5.5: Event Handler Run-time Environment](image)

Client global variables may be accessed by dereferencing pointers that are copied across at start of day.

there is no clash between pointer locations and private variable locations in the client global vector; in practice, well defined locations above those normally used by clients are reserved for the pointers.

5.4.6 Window Support for the Console Handler

The first client task to be implemented was that to provide window support for the Tripos console handler. This task presents console output in a moveable and scaleable window. Automatic horizontal wrap-around (rather than truncation) is supported, but not the scaling of text.

The Arthur operating system supports the concept of a text window, of which there may be only one at a time. A text window confines text output to the specified area
and automatically handles wrap-around on a per-character basis. The basic function of the console window task is therefore to map a suitable text window to the size and position of the WIMP console window. Resizing the WIMP console window requires a redefinition of the corresponding text window.

One difficulty is that it cannot be permitted for the console window to be partially obscured by another window, since there is no means of clipping textual output when using the text window technique described above. The result would be that the foreground window would get overwritten by text from the obscured console window. The only solution is to prevent other client windows from obscuring the console window. This is achieved by granting the console window the special status of priority window. The console window task registers its window as the priority window at start of day. Every other client is required to request the priority window handle (if any) from the WIMP task before redrawing a window. This enables a client to ensure that its windows are always displayed behind the priority window.

Another problem is that the definition of a text window is lost after every redraw window or update window request to the Arthur WIMP module. This means that the text window must be redefined after every redraw operation. Redefining the text window must be the responsibility of the console window task, since only this task has knowledge of the text window definition. The adopted solution is for the console window task to register a null event handler for the priority (console) window which simply redefines the text window position. The WIMP task is programmed to automatically run the priority window null event handler after the completion of its dispatch loop (and therefore after any redraw operations).

Another implication of this is that because the text window is undefined during a redraw operation, keyboard input or screen output must be prevented during this time. This is automatically achieved by making the priority of the WIMP task higher than that of the console handler. The console handler is usually the highest priority task in the system (to give maximum interactive response), so the WIMP task must become the highest priority task in the system. This has the added benefit that the servicing of WIMP events has precedence over all other processor activities, giving the optimum response to mouse events and preserving the integrity of screen display.

5.4.7 Operation

A utility program wm has been written to start up the window environment. It loads the mouse device driver, the WIMP task itself, and the console window task. By default, the WIMP task selects the highest available screen resolution and selects a monochrome palette of sixteen grey levels. However, different screen modes can be selected if required.
All standard Tripos programs run as normal - the only difference is that console output appears in a moveable window which, in general, will not be the same size as the screen. As with all window systems, applications that make assumptions about the characteristics of the output device (such as line length) can run into trouble when displaying data in an arbitrary-sized window.

Some applications (specifically the screen editor) can corrupt or destroy the window display. For example, the Worcestar editor completely reformats the screen in order to display the file under edit. The wm program can restart the window environment in order to recover from such an occurrence. This is done by wm sending a Tripos packet to the WIMP task requesting a complete screen refresh. The WIMP task duly calls the WIMP module requesting a full screen redraw, thus invoking every client redraw handler and restoring the display.

Although not normally required, the wm program can be used to explicitly stop the WIMP task and return to full screen display. Once again, wm sends a packet to the WIMP task requesting termination. Before closing down, the WIMP task runs any death handlers that have been previously registered by clients. This mechanism allows clients to be informed that the WIMP task is finishing, and may well cause prior termination of the client.

5.4.8 Performance

The performance of the system is surprisingly good. There is no noticeable difference in response between running the WIMP module under Arthur, and running the windowing system under Tripos.

It had been expected that rapid mouse movements would clog up the processor and noticeably slow down the system. This is because every mouse movement causes the WIMP task to be scheduled, even though typically there will be no WIMP event to service. However, there appears to be no significant performance penalty to pay. This is partly due to the speed of the RISC processor, and partly due to the fact that Tripos task switches have been purposefully designed to be cheap in terms of processor overhead.

Furthermore, a fast response to mouse clicks and other WIMP events is ensured by giving the WIMP task the highest priority in the system. This ensures that mouse events are serviced and windows are redrawn in preference to any other processor activity. It has the added benefit of blocking other tasks during critical activities such as redrawing the screen.
5.4.9 Evaluation

The Tripos WIMP system provides a simple window environment which has been implemented with relatively little effort. It achieves the aim of providing an environment for a window-based and mouse-driven user interface to applications. It runs alongside all standard Tripos software, allowing unrestricted access to the RPC package and other system utilities. Its performance is quite satisfactory for the simple role it was envisaged to play.

However, there are severe limitations in its use. The WIMP module has been designed to provide a simple window environment, and it offers none of the sophistication of more ambitious window managers. For example, there is very little control over the appearance or attributes of a window, and there is no way of defining a hierarchy of windows or of specifying relationships between windows. This is clearly not the environment for developing complicated window-based applications.

Another problem with the WIMP module is that it has been designed to run in the single-threaded environment of the Arthur operating system, meaning that it must be used with care in the multi-tasking environment of Tripos. In particular, the WIMP module is non re-entrant. This means that it is not permissible for one client to explicitly call an update window routine while the WIMP task is part way through running another client's event handler. For example, if a mouse click causes a WIMP event (such as a window exposure event) while another client is part way through making a series of synchronous calls to the WIMP module in order to update a window, the machine will crash.

The solution adopted requires that all clients make redraw requests through the WIMP task, rather than directly to the WIMP module. This has the result that requests are automatically queued and dealt with serially. Unfortunately, this restriction imposes limitations on the type of redraw operation that can be used. In particular, windows are cleared to their background colour before redrawing, causing a distracting flicker when displaying video data, for example.

Other criticisms include the fact that the window environment is non-standard. Although the applications themselves are running in a fairly unique network environment, there are advantages in using a standard window environment for reasons of external uniformity and public acceptance.

A final point is that the Tripos support provided in order for applications to be able to use the WIMP module has required some cunning but hardly ideal solutions to overcome the otherwise insurmountable problems described above. For example, the runtime environment for a client event handler requires indirection through previously registered
5.5 The Scanner System

5.5.1 Background

Recent advances in CCD (Charge Coupled Device) technology have heralded the arrival of the first generation of digital document scanners. These are capable of digitising paper-based images at resolutions matching currently available photocopying machinery and Group IV facsimile equipment. The available resolution is far greater than that achievable by video framestore systems, and is often adequate for the sampling of textual data.

A typical office document scanner will accept A4 paper, digitise to a resolution of 300 dots per inch, and offer some support for halftone images. The usual configuration is for the scanner to be connected to a PC or workstation via a high speed link into the processor data bus. Desktop publishing packages are popular applications that can make use of a scanner; photographs, line drawings and other paper-based information can then be integrated into the publication. Optical Character Recognition (OCR) software packages also rely on scanners for data input; these applications can interpret printed text in a variety of sizes and typefaces and convert it into ASCII or equivalent for possible manipulation using word processors, etc.

This section describes the work done to provide access to such a document scanner over the Unison network. It describes a simple application that makes use of the scanner, and considers more sophisticated ways that a document scanner could contribute to the working of a multimedia office.
5.5 The Scanner System

5.5.2 System Design

5.5.2.1 Introduction

The scanner system is based on the client/server paradigm. The scanner server offers its services to the network, and any client workstation may then bind to that service. This scenario enables the scanner to be shared between a number of workstations, and is consistent with the desire for a distributed office architecture which facilitates resource sharing. The alternative strategy of simply plugging the scanner into a particular workstation clearly lacks this flexibility.

5.5.2.2 Server Functionality

The scanner server is required to accept network requests from client workstations. Most requests invoke corresponding scanner operations; other network requests relate to locking, and are described in section 5.5.2.4 below. There are three basic types of scanner operation:

- status inquiries
- configuration requests
- read requests

The status inquiries return information relating to the hardware capabilities of the scanner, as well as to the current value of scanning parameters such as resolution and contrast settings.

The configuration requests provide the means for setting various scanning parameters, and for initialising the scanner in preparation for a scan. The scanning parameters relate to the size and resolution of the image to be read. The current value of these parameters can be discovered by making a status inquiry.

The read requests allows data to be scanned once all the scanning parameters have been set, and once the scanner has been initialised.

5.5.2.3 Reading

Two types of read operation are supported by the server; synchronous and asynchronous. A synchronous read is one in which a read request to the server causes a block of data to be read from the scanner and then returned to the client. An asynchronous read is one in which 'chunks' of data are read asynchronously from the scanner; a read request to the server causes the next block of data from the chunk to be returned to the client.
The advantage of an asynchronous read operation is that it is faster. One reason for this is that it introduces parallelism into the system - the server can be reading the next chunk while the client is processing a previous block. The other reason is that scanner read operations are more efficient when reading large quantities of data; there is a fixed overhead for every read operation (due to the mechanics of the scanner), and this becomes less significant for larger quantities of data.

5.5.2.4 Locking

An interesting problem is that of scanner locking. Although several clients may have simultaneous sessions with a single server, it is not permissible for more than one client to have instantaneous control of the scanner. For example, it should not be possible for a client to alter scanning parameters while a second client is in the middle of a scanning operation. There is clearly a requirement for some form of locking mechanism to control client access.

A simple locking mechanism has been implemented in the server based on the `getlock` and `freelock` primitives. A client requesting control of the scanner should call the `getlock` procedure until successful. The call will succeed if the lock is free, but fail if the lock has been granted to another client. When the call succeeds, the server blocks access to the scanner from all other clients until the lock is explicitly freed by the owner using a call to `freelock`. The only exception to this is when the server detects that the owner has crashed (the corresponding session times out), in which case the lock is automatically freed by the server to prevent deadlock.

A scan transaction should therefore be preceded by a successful call to `getlock`. The client may then set the scanning parameters and read the image in the knowledge that access is blocked to all other clients. It is the responsibility of the client to explicitly call `freelock` at the end of a transaction to permit access from other clients. There is no timeout on owning the lock, so an antisocial client could hog the scanner. However, it is assumed that the network environment is friendly (a common assumption in prototype distributed systems) and that clients will be well behaved.

5.5.2.5 Server Interface

The server interface is divided into a control interface and a data interface. The control interface provides access to the scanner configuration and status inquiry operations mentioned in section 5.5.2.2, as well as to the locking primitives described in section 5.5.2.4.

The data interface provides access to the synchronous and asynchronous read operations described in section 5.5.2.3. The server interface is based on Unity RPC and is defined in Appendix B. Owing to the fact that the Tripos implementation of Unity RPC
does not support 'multiple call packets', the data block size is limited by the maximum size of the UDL protocol data unit (7 kbytes) to be no more than 5000 bytes. In this implementation of the server, a value of 4 kbytes is used.

5.5.2.6 Client Access

Another design decision was the sort of interface that client applications should have to the scanner server. One possibility was to provide a Tripos stream interface [Knight 82]. A Tripos stream gives clients a uniform interface to a device or pseudo-device using a standard set of library routines. This is similar to the way Unix provides a standard i/o interface to system peripherals such as the disk and network interface. A stream is implemented by a stream handler which provides the set of i/o routines required by the standard interface library. These i/o routines may talk directly to a device or to another stream handler. A stream may map onto a network connection. A scanner stream handler would provide a Tripos stream interface to the server RPC interface discussed above. The problem is that although this would provide a standard data interface, there needs to be an additional control interface for scanner set-up. That is, the stream handler would allow data to be read from the scanner in a standard way, but there is still the need for a control interface to allow applications to set scanning parameters such as size and resolution. However, this can be solved because it is possible to pass parameters to a handler when a stream is first opened, although this implies that the stream would need to be closed and then re-opened after every scan.

To use the stream handler, a client application would open the stream in the standard way, passing parameters to the handler. The handler would use these parameters to set up the scanner ready for scanning. The application would then read the data (in units of bytes or larger blocks) using the standard library routine until the scan is complete. The stream would need to be closed and re-opened for a subsequent scan.

The provision of a scanner stream handler would simplify the implementation of scanner applications, and this would be a worthwhile exercise if further scanner development work was being considered. However, the prototype application mentioned below uses the server RPC interface directly.

5.5.3 Implementation

5.5.3.1 The Scanner Hardware

A Microtek MSF-300C scanner was chosen with a SCSI interface to the host computer. This is a flatbed scanner accepting paper sizes up to A4. It operates at resolutions up to 300 dots per inch (dpi) in either line-art mode (for text and other bilevel images)
or halftone mode (for photographs and other multilevel images). Other parameters that may be varied include contrast, brightness and the area of the page to be scanned. The grain size and dithering pattern may be specified for halftone scans. See [Microtek 87a] for a full specification.

The scanner is interfaced to a Tripos VME-based 68000 server via a commercially available SCSI to VME interface board. The server can host other SCSI peripherals (such as disk drives and tape streamers) since up to seven devices can be daisy-chained to the SCSI bus. This configuration is shown in Figure 5.6.

![Figure 5.6: Hardware Configuration of the Tripos Scanner Server](image)

5.5.3.2 Device Driver

A Tripos device driver was required to drive the scanner from the Tripos server. A generic Tripos driver accepts inter-process messages (known as packets) usually from a device handling process (or task) and performs a corresponding device action. It must also handle asynchronous device interrupts. The scanner driver provides a Tripos packet interface to the scanner SCSI interface defined in [Microtek 87b].

5.5.3.3 Server

The scanner server itself is implemented in BCPL and runs as a Tripos task on the Tripos server. The Tripos RPC package provides the multi-threaded environment required for the code.
5.5.4 A Prototype Scanner Application

The prototype scanner application is an interactive menu-based program giving the user full control over the operation of the scanner. It allows variable-sized images to be scanned, saved on disk, and printed on the network printer. The application communicates with the scanner via the scanner server RPC interface, thus allowing remote operation over the network. When running on an ARM Tripos system, scanned images can be displayed in the window environment described in section 5.4. These images can then be scaled by allowing the user to zoom in on an area of the image.

This client application was used to investigate the performance of the scanner system, and to evaluate the role of a scanner in a multimedia office.

5.5.5 Performance

The following performance measurements were made with the scanner server running on a 68020 machine, and the prototype client application running on an ARM Tripos machine. Both machines were lightly loaded, and there was minimal network traffic.

By default, reading operations are asynchronous (as described above), using a server cache size of 512 kbytes.

Figure 5.7 shows the variation of scanning speed with resolution for a full A4 image using default reading parameters. These figures represent the time taken to read the data form the scanner together with the transmission delay across the network. Note that all readings include a constant initial start-up delay at the scanner of about four seconds during which the scanner hardware is initialised. As might be expected, the scanning time decreases as the resolution decreases since there is less data to transmit across the network. [Note that a full A4 image represents 1 Mbyte of data.]
Figure 5.8 shows the variation of scanning speed with server cache size. A rapid improvement in scanning speed can be achieved as the server cache size is increased from zero to about 256 kbytes, after which the improvement is less dramatic. The improvement is achieved because a suitably chosen cache size reduces inefficiency at the scanner caused by reading large numbers of very small blocks; using an excessively large cache gives little extra benefit since performance is then limited by network transmission time rather than scanner access time. The measurements suggest that a cache size of about 256 kbytes would be a good compromise between scanning performance and server memory requirements. There is clearly a practical limit imposed by the server hardware on the amount of memory available for use as cache, and it is therefore prudent to restrict cache size to a reasonable value.

Figure 5.9 shows the variation of scanning speed with ISDN bandwidth when communicating with the server in 'loopback' mode through the ramp. The scanning time is influenced by the (constant) delay imposed by the portal and ramp, and the (variable) delay imposed by the ISDN U-channel width. It can be seen that for a sufficiently large U-channel, the scanning speed approaches...
that achievable over a single CFR. This is a general result which holds true for all applications data - if there is adequate ISDN bandwidth, then delays are usually negligible.

5.5.6 Evaluation

Experience with the system suggests that the concept of a network scanner shared by several client workstations is indeed realistic. The network transmission overhead was found not to be significant for a single CFR; in fact, running the client on a remote machine was found to improve performance since the client and server are not fighting over the same processor. Transmission speeds of about 450 kbits of user data per second were recorded between client and server (using the Tripos implementation of Unity RPC).

The biggest bottleneck in overall performance was due to client manipulation of the data, rather than due to the overhead in reading the data from the server. This reflects the large amount of data that makes up a scanned image (up to 1 Mbyte). For example, the time taken to display a full resolution A4 page on the workstation screen is about 25 seconds (for a quarter screen image), and this is greater than the time required to read the data from the server. Furthermore, the time taken to print a full resolution A4 page is about 30 minutes (2 Mbytes of ASCII data down a 9600 baud RS-232 link).

Sharing the scanner (and many other network resources) introduces the problem of access control and locking. However, the simple locking mechanism employed here seems to work quite well.

5.5.7 Providing a Fax Service

A network scanner can be seen either as a shared resource for standalone workstation applications such as desktop publishing packages, or as a communications tool for the transmission of scanned images. In the latter role, the scanner is behaving very much like a fax machine, and it is interesting to consider the paper design of such a system for the Unison network.

The action of sending a fax would involve scanning an image on the local scanner and transmitting this to the remote user. The transmitted image could then be filed on disk, printed on paper, or displayed on the peer workstation screen (or any combination of the three). The conventional fax service only supports the remote printing of faxed documents; the fax system proposed here offers additional support for image display and filing. The default action might be to file the image on disk, and then to notify the recipient (either by writing a message on his workstation display, or by leaving an electronic mail message) to inform him that a fax had arrived. The recipient may then
view the fax at his leisure, and print it as necessary. Furthermore, the print utility allows a selected area of the image to be positioned and scaled as required; this facility is not available using a standard fax service.

Ideally, the fax system would be supported by underlying network services providing location-transparent access to services by user name (this is fully discussed in chapter 6). This would obviate the need for a user to know the name or location of his own or the target scanner service.

The fax program could support simple and interactive modes of operation. In simple mode, default values would be used for all scanning parameters (eg full A4 page, 300 dpi resolution, etc). In interactive mode, the user would have full control over all parameters. The simple mode of operation could be invoked by typing "fax to tint" (or by using a suitable icon); this provides the functionality of the conventional fax service. The interactive mode of operation could be invoked by simply typing "fax" and then using a window (to specify the size and position of the image to be transmitted) and menus (to specify other parameters such as resolution). Transmission could then be initiated by typing "send to tint", or again by using a suitable icon.

5.6 The Workstation Video System

5.6.1 Background

All previous video work described in this thesis is based on custom-built video equipment designed to handle high resolution colour images. These video systems are made up of discrete network components controlled over the network by the workstation. Video data is displayed on a separate screen from the workstation.

Part way through the project, a video grabber module became available for the ARM workstation. This module allows a relatively low resolution monochrome picture to be captured from a standard video camera and displayed on the screen of an ARM workstation. The video module is described below.

Michael Ellis (from Acorn Computers) developed an Arthur application using a be-spoke windowing system based on the WIMP module to display video pictures captured by the grabber module. This was later extended to allow the transmission of video data between ARM machines on a single CFR using a simple minipacket protocol. Michael's work renewed interest in the idea of being able to display and manipulate video images on the screen of a workstation operating in the Unison environment. Although these pictures are low quality compared with those handled by the custom video equipment, there are advantages in having the video data displayed on the same screen as other workstation data such as text and graphics.
This section describes the work done to provide a workstation video system based on the ARM Tripos environment and compatible with the Unison network architecture. It is based on the video podule mentioned above, and on the Tripos WIMP environment described earlier in this chapter. The system was demonstrated working over a single CFR at the 1989 Satellite "Olympus" utilisation conference in Vienna.

5.6.2 The ARM Video Podule

The video podule has 128 kbytes of on-board video memory which is used for the capture of images from an attached camera, and for the display of images on the workstation screen. An image is stored in video memory as a 512 * 256 array of pixels, each pixel being 6 bits deep. [Owing to the design of the ARM video circuitry, it is not possible to display all 64 grey levels without using a monochrome monitor. Using a colour monitor, the maximum number of grey levels available is 16.] A full specification of the video podule is given in [Watford 88].

An on-board PROM provides SWI access to routines for transferring data between video memory and main memory. An image may be transferred as a series of blocks containing an arbitrary number of lines, each line being 512 bytes long. The PROM also provides a collection of image processing primitives (for scaling, inversion, rotation, etc) for the manipulation of images before display.

The procedure to grab an image into main memory is as follows. A SWI call is made to capture the image into podule video memory. A single or sequence of SWI calls can then be made to copy the image into main memory.

The reverse procedure is used to display an image on the screen. A single or sequence of SWI calls is/are made to copy the image from main memory into video memory. Another SWI call can then be made to display the image on the screen, either by clipping or scaling the image to the specified area.

5.6.3 Design

The aim was to provide a video system which allows a workstation to have a number of outgoing and incoming video streams to and from peer workstations on the network. Full control must be provided to the user at each end of the connection allowing him to terminate a video connection. The system must run in the Tripos WIMP environment, thereby giving the user control over the size and position of images displayed on his workstation.

The design adopted allows a user to set up a video transmission to a peer workstation. That is, a video connection is a simplex video channel. If the recipient wishes to reply to an incoming video stream, he must explicitly set up a transmis-
sion to his peer. This contrasts with a telephone connection which provides a full-duplex voice channel. Although such an approach could have been taken, the simplex design was chosen for simplicity.

5.6.4 Implementation

The video system has been implemented as a single grabber program which can run as a Tripos task in any ARM workstation fitted with the video podule. The grabber task is a client of the WIMP task described in section 5.4. That is, event handlers are registered with the WIMP task for every window that is managed by the program. The RPC interface to the grabber task is defined in Appendix C.

5.6.4.1 Access to the Video Podule

An ARM assembler library has been written to provide BCPL access to the video grabber SWI interface. This works in exactly the same way as the library described above for providing BCPL access to the WIMP module. It provides full control of the video podule to BCPL applications allowing an image to be grabbed into memory and displayed in an arbitrary position on the screen.

Since the video system is required to handle multiple video streams, there must be a mechanism to provide concurrent (but not simultaneous) access to the video podule memory. The video memory represents a single resource that must be shared between a number of consumers. Clearly, some form of locking mechanism is required to control access.

The solution adopted is as follows. Associated with each video channel is a frame buffer large enough to store a full image. The process of grabbing an image involves copying the image from video memory into the appropriate frame buffer. The process of displaying an image involves copying the image from the frame buffer into video memory. The BCPL coroutine mechanism [Moody 80] ensures that these procedures (which will be part of separate coroutines) are mutually exclusive since one coroutine cannot pre-empt another. The locking mechanism is thereby implemented by the use of frame buffers and by the coroutine mechanism.

5.6.4.2 Transmission of Video Data

Once in main memory, the RPC package is used to transmit the image to a peer workstation. The decision to use the RPC mechanism was based on the desire to keep the program as simple as possible by using the RPC package to handle the network. The grabber program includes the RPC client procedures to transmit video data, and the RPC server procedures to handle video data coming in off the network.
An image is transmitted over the network as a sequence of RPC data blocks followed by a sync block. As with the scanner system described in the section 5.5, the maximum size of an RPC block is limited by the maximum size of the UDL protocol data unit to be 5000 bytes of user data. In practice, a block size of 4 kbytes is used.

The RPC server thread at the recipient workstation simply copies incoming data block into the appropriate frame buffer. On receiving the sync pulse, the receiver displays the contents of the buffer on the workstation screen. In principle, transmitting a sync block is unnecessary since the receiver can calculate when to display an image by knowing the size of the image and counting the number of incoming blocks. However, using a sync pulse ensures that in the unlikely event of an RPC transmission failure due to congestion the transmitter and receiver remain in step.

5.6.4.3 Handling Multiple Streams

In order for the program to be able to handle multiple incoming video streams, the RPC server procedures must have some means of knowing to which video channel an incoming RPC is being directed. For example, the RPC server procedure that accepts incoming data must be able to find out which frame buffer it ought to be using. The solution adopted is for the transmitter and receiver to exchange handles at call setup time. Any RPC from transmitter to receiver must quote the receiver handle, and vice-versa. This allows RPC server procedures to index the appropriate video channel.

In order for the program to be able to handle multiple transmissions, there must be multiple threads available for this purpose. In practice, one client thread is devoted to handling keyboard input and interpreting user commands, and a number of other client threads are available for transmission. The number of transmission threads determines the number of simultaneous transmissions that are available (three in this implementation). Communication between the 'user thread' and the transmission threads is via the semaphore mechanism implemented as part of the RPC package.

5.6.4.4 Compression

A very simple compression technique has been implemented to reduce bandwidth requirements by fifty percent. This is achieved by coding two pixels using one byte. There is no loss of information for the following reason. The video module codes a 6-bit pixel using one byte, giving a potential of 64 grey levels. However, since only a maximum of 16 shades of grey can be displayed using a colour monitor, each pixel can be completely encoded using just 4 bits.

A compression routine has been implemented in ARM assembler to perform a linear mapping between the raw 6-bit deep image and the stored 4-bit deep image. The processing overhead in performing this mapping is made worthwhile by the fifty percent
reduction in image data and subsequent saving in transmission time. A transmitted image is then 512 * 256 * 4 bits = 64 kbytes per frame.

5.6.4.5 Termination

Either end may stop a video transmission by simply deleting the appropriate video window, or by the transmitter explicitly requesting termination. An attempt is made to close the connection gracefully using a terminate RPC. The workstation initiating the termination notifies its peer by sending a series of terminate RPCs until successful. The terminate RPC will fail if the peer workstation is a transmitter which is part way through transmitting a frame; a subsequent call will succeed when transmission of that frame is complete. The terminate RPC causes transmission to be suspended, and the association to be deleted. The frame buffers are left intact at each end, so that the last transmitted image can still be viewed in its window.

5.6.5 Operation

The grabber task starts up by registering the specified service name (if supplied) or a default service name with the secretary. A window is then opened and used to display a local image grabbed from the camera. The user thread provides a command line prompt which responds to a number of keywords. For example, typing "snap" causes a further window to be opened and used to display another local image grabbed from the camera. Typing "help" gives information on the available commands and their usage.

To start transmission (to the default service name) type "transmit to <location>". This causes an association to be set up, and a window opened at each end. The user is returned a handle identifying the thread handling the transmission. The transmitter grabs an image into a buffer, transmits it block by block, and sends a sync block to mark the end of a frame. The receiver copies successive blocks into a buffer, and on receiving a sync block it displays the contents of the buffer on the screen. The whole process then repeats itself until terminated by either party.

Since the initiating user is returned a handle identifying the transmitting thread, the transmission may be explicitly stopped by this user typing "stop <handle>". Alternatively, the transmission may be implicitly stopped by the user at either end deleting the corresponding video window.

Images are displayed in a collection of windows which may be manipulated using the mouse. Resizing a window causes the image to be automatically scaled to the new window dimensions.

Typing "quit" deletes all associations, closes all windows and terminates the program.
5.6.6 Performance

The performance of the video system in terms of frame update rate is dependent on a number of variables. For example, a larger display window or a higher screen resolution causes redraw times to be longer and the frame rate therefore decreases. However, potentially the most important factor is the load imposed on the processor by concurrent processes. For example, compiling a program or accessing the filing system may compromise grabber performance. The effect of other tasks on video performance can be minimised by making the priority of the grabber task higher than other ‘user’ tasks in the system. In practice, this means running the grabber task in the highest priority shell. However, the inevitable effect of concurrent transmission processes is that the individual performance of each of these process is reduced.

For a one-way transmission over a local CFR using the highest resolution screen mode and a window size of 500 * 500 (approximately a screen quadrant), the frame rate is about 0.4 frames per sec (ie a new frame is transmitted every 2.5 seconds). This represents a useful bit rate of approximately 205 kbps. If the window size is increased to cover the entire screen, the frame rate is reduced by about 30% to 140 kbps.

For a two-way transmission over a local CFR, again using the highest resolution screen mode and the default window size of 500 * 500, the frame rate in both directions is about 0.25 frames per second (ie a new frame is transmitted every 4 seconds). This represents a one-way bit rate of approximately 130 kbps, and a total bit rate of 260 kbps.

Note that the use of an intermediary frame buffer at both ends (in order to meet the requirement for multiple streams) means that much processor time is expended in copying data between frame buffer and video memory. Higher data rates were achieved by eliminating the frame buffer at the expense of loss of support for multiple streams.

It is interesting to record the effect of the compression technique used by the video system. Without compression, for a one-way transmission over a local CFR using the highest screen resolution and default window size, the frame rate is approximately 0.25 frames/sec, which this time represents a bit rate of about 260 kbps. This compares with the rate of 0.4 frames/sec recorded above using compression, representing a 60% speed improvement. Note, however, that the achievable bit rate is higher (260 kbps compared with 205 kbps) when not using compression, since removing the compression routine reduces the load on the processor.

To summarise, maximum performance can be achieved by using small windows in relatively low resolution screen modes, and by running the task at a higher priority than
other user tasks in the system. In addition, the number of parallel video streams should be kept to a minimum.

5.6.7 Evaluation

The video system described here has proved to be an interesting experiment in the incorporation of video pictures into a workstation window environment. The performance of the system is fairly modest, but quite adequate for single snapshot transfers.

The video system highlights the point that a single processor workstation is hard pressed to deal with multiple high speed data streams while still providing acceptable performance to other applications. One problem with the video module used in this system is that the processor is responsible for the transfer of what turns out to be a large quantity of image data between main memory and video memory, and this is a processor intensive activity. A better design would be to use DMA hardware to remove this overhead. Another potential bottleneck is that the video data shares the same network interface as other workstation traffic. A possible solution here would be to devote a second network interface entirely to video traffic. Some of these ideas are currently being implemented as part of another project [Hopper 88].

It is important to realise that although this system allows video pictures to be displayed on the workstation screen, there is no provision for the integration of video pictures with other applications. It would be a relatively simple job to provide support for the filing of video pictures on disk, and this would at least allow access from other applications (such as desktop publishing packages, picture databases, printing utilities, etc). It is only by proper integration that the video system would become a useful tool in the office rather than simply an experimental application.

5.7 Multicast Video

5.7.1 Introduction

A general multicast service offers clients a data replication facility useful for such purposes as bulk file distribution and multipoint conferencing. The main benefit to applications is that a multicast system removes the need for source replication of data destined for multiple peers. That is, the multicast service will handle the replication on behalf of the client. Another benefit to network designers is that bandwidth requirements can be reduced through critical sections of the network such as switches and wide area links. For example, a client at one site wishing to send data to two peers at a remote site need only send a single stream to a multicast entity at the remote site; the multicast entity will then handle the local replication of the stream.
It is important to distinguish between the functions of broadcasting and multicasting. A broadcast entails transmission to all machines at all locations on the network. By contrast, a multicast limits transmission to a selected group of receivers. Multicast systems therefore make more efficient use of network bandwidth, typically at the expense of increased complexity.

This section describes some experiments conducted using a prototype multicast system to support the three-way transmission of video data. These experiments highlighted a number of issues that relate to the general use of a multicast service, and these are also considered here.

5.7.2 The Unison Multicast Service

The Unison multicast system comprises a collection of data splitters and control entities that interact to provide a data replication service across the network. The control functions are handled by the Multicast Control Service (MCS), while the actual data replication is performed by the Multicast Data Service (MDS). A service that is dynamically created to provide a particular instance of a multicast mapping is termed a multicast channel.

In the prototype multicast system described here, the MCS and MDS are implemented on different machines. The MCS is a Tripos task using the BCPL RPC package. For performance reasons, the MDS is a Transputer system programmed in Occam. A full description of the multicast system is beyond the scope of this thesis - refer to [Shrimpton 87] for details.

The multicast service offers data replication at the UDL level. No support is provided for higher level protocols such as RPC. To set up a multicast, a host contacts its MCS requesting the creation of a multicast channel. The MCS then contacts an appropriate MDS on behalf of the client, causing a new channel service to be advertised with the secretary. Any client may then join the channel in order to send or receive data by simply creating an association with the channel service. A reserved location in the userdata vector of the association.create RPC is used to specify whether the client wishes to transmit to or receive from the particular channel.

5.7.3 Multicast Extensions to the Video System

5.7.3.1 Background

The workstation video system was identified as a promising client application for early evaluation of the multicast system. Video systems in general have much to benefit from the use of a multicast service due to the high volumes of traffic involved. In addition, the workstation video system offered the benefit of a Tripos/BCPL software environ-
ment to facilitate the modifications required to make use of the multicast service. For both of these reasons, the video system was chosen as a model multicast client.

5.7.3.2 UDL Transmission

The major extension necessary to the video system was the adoption of the raw UDL protocol for data transmission. As explained above, all MDS replication is performed at the UDL level, freeing the multicast service from the difficult requirement of supporting higher level protocols such as RPC. This means that multicast clients are not able to use higher level two-way communication protocols such as RPC. Furthermore, there are interesting philosophical problems relating to the multicasting of higher level protocols to do with the semantics of a single protocol failure in relation to the rest of the multicast (see section 5.7.6).

In practice, the adoption of raw UDL requires that the grabber program described in the previous section must take responsibility for accessing the Tripos UDL driver directly rather than via the RPC package. Having established an association in the normal way, the image is transmitted as a series of UDL blocks instead of by using a series of RPCs. A reception thread is started at association set-up time whenever the initiator has requested raw UDL as the transmission protocol. This thread 'double buffers' reception requests and copies incoming UDL data blocks into a suitable frame buffer.

A single CFR packet sync pulse is transmitted at the end of each frame. This serves the purpose of instructing the receiver when to display a new frame, and also ensures that the transmitter and receiver remain in step in the event of block loss. The receiver also keeps a count of incoming blocks for each frame. If the sync block arrives before the required number of data blocks, the receiver assumes that there has been a transmission failure. Incomplete frames that are detected in this way are simply discarded.

Data blocks consist entirely of user data (4 kbytes per block); that is, there is no higher level protocol structure. This is to allow incoming data blocks to be copied directly from the network interface to the receiver frame buffer without using intermediate memory. This reduces the load on the processor, thus improving overall throughput. However, the absence of any block structure prevents the use of a block typing scheme; data blocks are differentiated from sync blocks by comparing the size of the incoming UDL block (4 kbytes, compared with 28 bytes respectively). Furthermore, the lack of a block numbering scheme for data blocks requires the use of the block loss detection technique described above.

There is no flow control mechanism above that provided by the medium access control level imposed by the CFR slot structure. This reflects the desire for a lightweight protocol, but it does have its disadvantages. In particular, UDL blocks that arrive at the receiver too quickly (ie faster than the sink process can handle them) end up being dis-
carded without the transmitter being informed. That is, no back pressure is applied to the transmitter at the UDL level, meaning that an unacceptable number of frames may be discarded.

5.7.3.3 UDL Termination

An interesting problem is that of terminating a UDL transmission. A transmitter can simply stop transmitting and then delete the underlying association. However, a receiver needs to somehow inform the transmitter of its intention to close the connection. Unfortunately, simply deleting the receiver end of an association does not cause UDL transmission failures. Incoming UDL blocks with an invalid port number are simply discarded at the receiver without the transmitter being notified. There needs to be some other mechanism for a receiver to terminate a connection.

One solution is to arrange for the receiver to send a message to the transmitter instructing it to stop transmission. This has the practical problem of requiring that the connection be full-duplex and that the transmitter be listening out for termination request packets. A more fundamental problem is the semantics of this termination packet during a multicast. This is a problem discussed later in this section.

Another solution is to make use of a hardware feature on the CFR station chip allowing a receiver to reject all CFR packets from a specified source thus generating packet transmission failures at the transmitter. This is done by making a suitable entry in the receiver source selection map (also known as the 'hate list') for the source station address specifying that all packets from that address be rejected. If this is done for a short time, the transmitter can be persuaded to back off (this is sometimes called 'blipping' the transmitter). The problem with this technique is that all packets from the specified source are rejected, and not just those on the relevant association. However, since higher level protocols can cope with such packet losses, parallel connections using high level protocols will remain intact.

The blipping technique was the solution adopted by the extended video system. The blip duration must be chosen to be long enough to convince the transmitter to give up trying, but not so long as to disrupt parallel communications unnecessarily. A period of one second is used in the grabber program, but the critical blip duration is dependent on the transmitter retry strategy. The situation is made more complicated when the video stream passes through an intermediary network component; in general, the retry strategy of the intermediary component will be different from that of the originating source.

5.7.3.4 UDL Operation

To start a raw UDL transmission (to the default service name), the user starts the grabber program in the usual way and then types "transmit to <location> udl". This is the
same syntax as for the RPC transmission described above, except that the \textit{udl} switch has been added. [RPC and UDL transmissions may run concurrently.]

Initiating a transmission in this way causes an association to be made with the default service at the specified location. An entry in the \textit{userdata} vector of the \textit{association.create} RPC informs the peer workstation that the UDL protocol is to be used instead of RPC. The receiver duly attempts to start a reception thread and returns a success notification. Having made the connection, the transmitter starts a transmission thread and the process is complete. Once again, a handle is returned to identify the transmitter thread. A user at the transmitter may stop the transmission by typing "\texttt{stop <handles>}"; alternatively, a user at either end may simply delete the appropriate video window.

5.7.3.5 UDL Performance

For a one-way transmission over a local CFR using the highest resolution screen mode and a window size of 500 $\times$ 500 (approximately a screen quadrant), the frame rate is about 0.5 frames/sec (ie a new frame is transmitted every 2 seconds). This represents a useful bit rate of approximately 270 kbps, which is a 30\% improvement over using the RPC package. This increase in performance is largely due to the fact that when using raw UDL there is no processor overhead in marshalling and unmarshalling video data into and out of RPC buffers.

By careful choice of transmitter retry counts, UDL transmission can be made almost as reliable as using RPC. The transmitter retry strategy must be able to cope with occasional block transmission failures and CFR reframing events. This has been achieved by using a combination of hardware and software retry loops with a suitable delay between attempts. The station chip makes 7 hardware retries per packet with an inter-packet delay of two CFR gaps; the transmission routine makes 10 software retries per UDL block with an inter-block delay of 0.5 seconds in the event of a CFR reframe event.

Lack of UDL flow control means that if the receiver is very busy (for example, if it is handling more than two or three concurrent video streams), blocks are discarded without the transmitter being informed. This contrasts with the RPC protocol in which every block is acknowledged and therefore guaranteed to get through.

The \textit{sync} mechanism was successful in preserving image integrity despite block losses.

The blipping mechanism was successful in terminating UDL connections without breaking parallel RPC streams from the same source.
5.7.4 Video Multicast Experiments

For the convenience of these experiments, a third-party system was used to actually create the video multicast channel. That is, a separate Tripos task was used to interact with the Multicast Control Service and create a multicast channel service of the specified name. The channel service is offered by an appropriate multicast data server.

Receiver video systems connect to the channel by creating an association with the particular channel service. This is achieved by typing "join <multicast channel name>". They register themselves as receivers with the multicast server by using the appropriate location in the userdata vector in the association.create RPC.

![Figure 5.10: The Three-way Multicast Experiment](image)

To start transmission, the user types "transmit service <multicast channel name> multicast". This causes the transmitter to make an association with the specified channel service (rather than the default video service) and start transmission. The multicast switch automatically invokes the raw UDL protocol, and causes the initiator to register as a transmitter with the multicast server by use of the userdata vector. Thereafter, the multicast system handles the replication of data to the registered receivers.

A number of multicast mappings were tested, including one transmitter to two receivers and one transmitter to three receivers. Since there were only three video modules available, the third receiver was on the same machine as the transmitter. This configuration is shown in Figure 5.10. Unfortunately, a temporary bug in the grabber program prevented reception of more than one video stream. This meant that it was not possible to try out the full three-party multicast in which each machine transmits to the other two.
5.7 Video Multicast Performance

The multicast mapping between one transmitter and two receivers worked very well. The transmitter data rate was comparable to that between a transmitter and a single receiver, and there was no noticeable degradation in performance.

The multicast mapping between one transmitter and three receivers was less successful. The problem was one of contention between the transmitter and receiver processes running in the same machine. Although the multicast worked well for the single receiver systems, there were large numbers of dropped blocks at the 'transmitter and receiver' machine. However, the multicast system itself seemed well able to handle the data rates involved (in fact, the MDS supports speeds of about 1 Mbps).

It is interesting to note that when a receiver joins a multicast dynamically, the synchronisation mechanism described above is successful in synchronising the receiver with the transmitter. In general, the receiver will start receiving part way through a frame, and this first (incomplete) frame must be discarded. However, receipt of the sync pulse then allows the receiver to latch on to the transmitter. Without a sync pulse, it would not be possible for the receiver to synchronise with the transmitter.

5.7.6 General Multicast Issues

There are many interesting problems relating to the design of multicast systems that are beyond the scope of this thesis. However, it is interesting to consider some of the issues facing an applications writer wishing to make use of such a multicast system.

The first is that an application needs some means of creating a multicast mapping and advertising its existence to the outside world. In Unison, this is achieved by interaction with an MCS which causes a multicast channel service to be advertised with the secretary. In the experimental configuration described above, this interaction was performed by a third-party system. However, in a production application, this capability must be included in the application itself.

The means of name distribution to allow a client to join an existing multicast channel is an interesting management problem. Furthermore, a multicast client application must be able to interact with such a management function in order to locate existing channels. The way that this is achieved depends on the nature of the management function, but this is another interface that must be built into the application.

Another issue is that of multicast protocols. Most conventional two-party protocols fall down in a multipoint environment. For example, consider a multicast RPC from an initiator to two recipients. The multicast system could be programmed to effectively duplicate the original call by making two new calls to the destination machines. The problem
is that if one of these new calls fails, the initiator needs to be informed of a 'partial success'. This could be achieved by the multicast system mapping all the return codes from the destination machines onto a single return code to be passed back to the initiator. However, this requires an extension to the basic two-party protocol. The upshot of this is that the multicast of high level protocols (in which replies or acknowledgements are used) is possible, but only in a protocol-specific way and by making extensions to the basic protocol. The implication for application writers is that support may need to be provided for a lower level unacknowledged block protocol with a corresponding reduction in reliability.

The management of a multicast channel is another interesting issue. Normally, a transmitter will not (and should not need to) know to whom it is speaking and how many clients are listening. However, a user may need a guarantee that at least one person is 'out there somewhere'. This knowledge can only be gained by the application interacting with a multicast control entity. This management interface could be an extension to the control interface described above.

A final point is that if a transmitter knows that it is likely to be involved in a multicast, although initially it may only be required to transmit to a single receiver it must use the multicast service to allow possible future access by other potential receivers. For example, if a transmitter wishes to transmit to a chosen receiver but is happy for that video stream to be copied to other receivers some time later, it must go through the multicast service at the outset rather than go directly to the chosen receiver.

5.8 Experiments over the Wide Area Network

An early version of the CFR office was demonstrated at the 1988 Alvey ITEX exhibition in London. The demonstration office was shown interworking with a similar office at Loughborough over the Alvey High Speed Network. The colour slow-scan video system, the wideband speech system, and the toll-quality telephone were all shown to operate over the wide area network. In addition, a number of full resolution scanned images were read from the server at Loughborough, although this was not part of the mainline demonstration. [Although the colour video system was under the control of the workstation at this stage, the user interface to this video system was rather simple and was not based on any windowing system. As a consequence, the demonstration may have been less impressive to a casual observer, although the underlying infrastructure was in place.]

Experience with the system on this occasion revealed a number of shortcomings in the network infrastructure particularly relating to the performance of the CFR portal. This prompted a redesign of the network handling processes used in the portal, and this modification led to a significant improvement in inter-site performance [Siddiqui 89].
On subsequent occasions, the workstation video system and the Olivetti remote filing system have been demonstrated working between Loughborough and the Rutherford Appleton Laboratory. Furthermore, extensive testing of applications has been conducted at Loughborough by using 'loopback' connections over the ISDN and by using the two client rings at that site.

The performance of most applications over the wide area network is comparable to that over a local CFR assuming that sufficient ISDN bandwidth has been allocated for the connection. In practice, this requires that the allocated U-channel is sufficiently wide. [Since no single application generates traffic at more than the primary rate bandwidth (2 Mbps), the ISDN is not a limitation to any one client. However, multiple applications may well have a joint bandwidth requirement in excess of that available from the ISDN.] If there is insufficient ISDN bandwidth, traffic is significantly delayed due to back pressure being applied by the wide area network (via the portal). The delay imposed by the network in 'non-overload' conditions (about 7 ms [Siddiqui 89]) is only significant for real-time voice services. For a detailed analysis of the characteristics of the wide area network, refer to [Siddiqui 89].

Note that the complete Unison exchange management services were not in place during application tests over the wide area network. In particular, U-channels had to be configured manually at the ramp. In a fully working exchange, U-channels would be set up and adjusted dynamically according to user connectivity requirements. This will clearly impose some overhead on clients accessing services over the ISDN, particularly due to the initial delay associated with synchronising a U-channel to a new site. The single slot synchronisation delay is approximately 150 ms [Burren 89b], but the total call set-up delay (including exchange management overhead) awaits measurement.

5.9 Summary

This chapter describes an experimental multimedia office based on the Cambridge Fast Ring local area network. As with the Cambridge Ring office, it is based on a loosely coupled collection of components interlinked via the network. There has been a greater emphasis on workstation control in the CFR office, and many components offer a control interface to permit their control over the network.

The CFR office infrastructure provides a more sophisticated working environment than that available from the CR office. The multi-tasking workstation operating system provides a better basis for applications development. In particular, the Tripos implementation of Unity RPC has proved to be of great benefit to applications. The RPC package provides a convenient multi-threaded programming environment that hides details of the network from the application. It incorporates a reliable transport service that overcomes the reliability and timeout problems encountered in the CR office.
A distributed scanner system has been described in some detail, and experience with this system seems to validate the server-based design. The network performance of the CFR ensures that the penalty paid for remote access is kept to a minimum.

A workstation video system has also been described. This has the advantage over previous video systems of having the video picture displayed in a workstation window alongside other media such as text and graphics. However, the performance of the system in terms of frame update rate is rather poor. Indeed, this experience emphasises the problem of a single processor handling an intensive network activity while attempting to run other user processes. The workstation video system was demonstrated at the 1989 "Olympus" utilisation conference in Vienna.

A prototype multicasting system has been tested using workstation video data. Although the system appears to offer performance benefits to certain applications, there are problems in integrating the system into the office environment.

A photograph of the workstation window environment is given in Photograph 5.1. It shows the console task window, a scanner window and a number of video windows.
Photograph 5.1: The Workstation Window Environment

This photograph shows the workstation display while running the ARM Tripos window system. The console window is shown at the bottom of the screen; the console handler is currently connected to the scanner task. A scanner window is shown (partially obscured) on the top right of the screen; it is displaying a previously scanned image of the Apple LaserWriter title sheet. Several video grabber windows are also shown. All these windows may be moved and scaled as required. Further video and scanner windows can be added as necessary, although an eventual limit is imposed by the amount of available memory.
Chapter 6
Control and Management of Office Components

6.1 Introduction

The integrated office proposed in this thesis is one in which a collection of multi-
media components will offer a variety of services to the user. A generic office com-
ponent will offer a control interface and a data interface. The control interface is for
the benefit of a host wishing to operate the component. The data interface is for client
access to the resource being provided. The distributed nature of the office reflects the
requirements for resource sharing, efficiency of implementation, reliability and extensibility.

In order to provide integrated services at the user level, these office components
must be controlled and co-ordinated in a coherent manner. Such action is required to
hide the distributed nature of the office and give the user the illusion that he is work-
ing at a single multimedia terminal. Although the individual components are physically
discrete, they must be logically bound together in software.

Previous experience with the Cambridge Ring and Cambridge Fast Ring offices revealed
the general need to manage the flow of control within a network. This is the first re-
quirement for co-ordinating office components in the manner described above. One ap-
proach is to divide the network into groups of components on the basis of user require-
ments and to confine the flow of control to remain within those groups. The require-
ment is to provide the user belonging to each group complete control over every com-
ponent within the group, but to deny such access from users outside the group.

This chapter describes a model for the management of control based on such a scheme.
The model also turns out to have interesting properties relating to accessing peer ser-
tices on the basis of user name. The chapter describes the application of the
model both to the problem of control in the Cambridge Fast Ring office and to the
problem of accessing peer services. The implementation of the model is described
in some detail. The chapter goes on to identify several deficiencies in the basic model,
and to suggest possible enhancements. The chapter concludes by considering the im-
plications that a proposed modification to the Cambridge Fast Ring network architec-
ture might have on some of the ideas discussed in this chapter.
6.2.1 Domains of Control

The model described here is based on the allocation of 'domains of control' to office users. A domain of control (DoC) defines the group of office services over which a user has complete control. There is a one-to-one correspondence between a user and a DoC. The user associated with a DoC can be thought of as the owner of that DoC. Every DoC must contain at least one control entity to allow control of the components within that DoC. A domain of control is only defined for the duration that a user is registered with (or logged on to) a control entity responsible for that DoC. Domains of control may overlap arbitrarily, meaning that components from one DoC may belong to any number of other DoCs.

The model specifies that the flow of control from a user be confined to his domain of control. That is, office control software conforming to the model must only provide access to those services within the user's DoC. The model merely maintains the definition of a user's domain of control, and it is the responsibility of host control

Figure 6.1: Two overlapping Domains of Control for Users A and B.
A domain of control defines a set of services (shown in italics), and these are provided by corresponding office components.
software to make use of this information. No mechanism is provided to prevent host software from violating the model.

In practice, a DoC will encompass a collection of personal components that might be considered private (like a telephone) and a number of shared components that might be considered public (such as a printer). Typically, a workstation will be the control entity. The workstation user then has free access to all the services within his DoC and can perform all office functions supported by the collection of components under his control.

A schematic representation of the model is shown in Figure 6.1. Here there are two users sharing a couple of components (a printer and scanner) but each with a set of personal components including a workstation. For example, user A on workstation W1 has within his DoC the toll quality speech services tqs-c and tqs-d (one for control, one for data) on telephone T1, the video services video-c and video-d on the frames-tore F1, the printer service lsw on the shared printer P1, and the scanner service scanner on the shared scanner S1. The DoGs overlap where components are being shared by more than one user.

6.2.1.2 User Mobility

A domain of control comprises a set of static services that can be associated with a user irrespective of his whereabouts, and a set of dynamic services that depends on location. This means that a user's DoC changes as he moves around the network. Static services have the additional property that they are accessible (using this model) by other network users even when the owner is not registered with the system. By contrast, dynamic services are only made accessible (using this model) when the user is logged on to a workstation.

An example of a static service is an electronic mailbox which does not normally move around the network and which is accessible regardless of the availability of the owner. That is, a user's mailbox will be at a fixed location and will be available regardless of whether the user is logged on to his workstation.

An example of a dynamic service is a telephone service whose location depends on the whereabouts of the owner and whose accessibility is dependent on the availability of the owner. That is, a user's telephone will depend on his location on the network and it is only possible to call him (using this model) when he is logged on to his workstation.

The dynamic quality of a DoC reflects the requirement to preserve a convenient office environment regardless of location. The components belonging to a DoC are expected to be physically close to the workstation that is controlling them. If a user moves
to a different workstation, it is reasonable for him to acquire a new set of components that is more conveniently situated, and this requires a redefinition of his DoC. For this reason, a user's domain of control tracks his movement over the network providing the user with the freedom of choice over where he works on the network. This concept is termed user mobility.

A diagram illustrating this feature is given in Figure 6.2. At location 1, the domain of control associated with user A comprises the services belonging to workstation W1 and the static services belonging to user A (a mailbox and a filestore, for example). At location 2, the domain of control comprises the services belonging to the new workstation W2 and the same set of static services.

6.2.2 Realisation

This section describes how the abstract concept of a 'domain of control' is realised within the model.

Associated with each office user is a user profile. A user profile is simply a set of static services that can always be associated with that user irrespective of his location on the network.
Associated with each control entity (workstation) is a desktop profile. A desktop profile defines the set of services that are under the control of that control entity. This set of services reflects the set of physical components that provide them. The services defined in the desktop profile provide the dynamic component of a user's domain of control.

A user's domain of control is therefore defined by his personal user profile and by the desktop profile belonging to the control entity at which he is working. This combination is dynamic, and is referred to here as a user's office profile. That is, an office profile is the combination of a user profile and a desktop profile that defines the domain of control for a particular user at a particular location. The office profile for user A at location 1 is shown in Figure 6.3.

Desktop profiles will vary with time as offices are re-organised and as components are added and removed from the network. User profiles will also vary as static services such as a mail system or telephone answering machine are re-organised. Office profiles vary rather more quickly with time as users come and go, and as users move around the network.

In practice, a network management function is required to maintain the definitions of a user profile for each user and of a desktop profile for each workstation. Since these profiles have a slow variation with time, some form of directory service is adequate for this purpose.

An additional management function is required to maintain the domains of control (defined by the office profile) for each user currently using the network. This entity is christened an office manager.

Both of these network functions must exist within every Unison domain (chapter 3) that conforms to the model described. That is, an office manager is responsible for maintaining office profiles for users within a single Unison domain.
Associated with every user is a home domain which can be considered as the home base of that user. The office manager in a user’s home domain is known as the home office manager for that user. User names within a given domain must be unique; a fully qualified user name consists of the combination of domain name and user name.

6.2.3 Access to Services

The office manager can be thought of as a simple directory that maintains the dynamic bindings between services and people. A user can discover the services within his domain of control by looking up the services associated with his own user name. In addition, the office manager provides a useful means of locating peer services on the basis of the user to which they belong.

More precisely, the office manager maintains the dynamic mappings between the triplets:

\[(<\text{generic service}>, <\text{home domain}>, <\text{user}>) \rightarrow (<\text{service}>, <\text{domain}>, <\text{host}>).\]

In the first triplet, the generic service name is the general name of a service such as telephone or mailbox, and the home domain and user names specify a fully qualified user name. The second triplet is the particular instance of the generic service for the specified user, and this is termed the service instance. The service instance is the currency required by the secretary in a binding request.

In locating a service within a local user’s domain of control, the user name specifies the local user name, the home domain name is null, and the generic service name specifies the required service. This is the functionality required by control software in order to implement the ‘domain of control’ concept described above.

To locate a service within a peer user’s domain of control, the user name specifies the peer user name, the home domain specifies the home domain of the peer user (or can be null if his home domain is the local domain), and the generic service name specifies the required service. This functionality provides a convenient means of locating peer services on the basis of user name.

Note that a control entity need only communicate with the local office manager; requests to remote domains are passed on transparently to a peer office manager.

Access to a service now becomes a two-stage process. The first stage involves an interaction with the office manager to derive the location-dependent physical name of the service (based on network topology) from the location-independent logical name (based on the user name). The second stage involves presenting the physical name to the secretary as part of the binding request in the usual way.
For example, suppose that user Bill wants to send a facsimile to user Ben within the local (home) domain of blue. Bill specifies the triplet (facsimile, <null string>, Ben) to the office manager, and will be supplied with the physical name of that service (the service instance), say (fax, blue, hermes). This physical name can then be presented to the secretary in the normal way as part of the binding request. Note that Bill requires no knowledge of the location of Ben within this domain. Furthermore, if Ben changes location, a new physical name is returned as a result of the first stage, but overall access is unaffected.

The fact that a client does not need to know the location of a peer service is a property known as location transparency. Furthermore, the fact that changing the location of the service does not affect client access is a property known as migration transparency. Use of the office manager can be seen to provide intra-domain location-transparent and migration-transparent access to services on the basis of user name.

In order to provide inter-domain migration-transparent access to peer services, there has to be interactions between office managers across the network. These interactions are not defined in the basic model described here; an extension to the model in order to provide such support is proposed in section 6.5. Note that inter-domain location-transparent access to peer services is already supported by means of a suitable entry in the user profile.

It is worth mentioning that a scheme for providing transparent access has been implemented in Unison by enhancing the secretary service [Chapman 89]. It is based on the use of the Unison directory service to maintain the mappings between an arbitrary logical name space and the physical name space used by the secretary. However, this scheme only supports a restricted form of inter-domain migration transparency based on the use of a set of ‘hints’ to suggest where a service might be found. Furthermore, there is no support for the concept of user mobility.

6.2.4 Operation of the Model

This section describes how the above features of the model are used in managing a connection between two office components.

A component will typically advertise two services with the secretary; one will be a data service for access from a peer component; the other will be a control service for access from a controlling workstation. For example, a framestore offers the data service video-d and the control service video-c. This section describes how the model is used to implement the concept of a ‘domain of control’, and how it is used to provide access to peer services on the basis of user name.
6.2.4.1 Setting up a Connection

Suppose user Bill working on workstation Egor wishes to set up a video framestore connection to the peer user Ben on workstation Gollam in the same (home) domain. Suppose Bill's framestore is called Darius, and Ben's framestore is called Venus. See Figure 6.4. The sequence of events is as follows:

1) The video control application running on Egor goes to the office manager requesting all the instances of the generic video control service associated with user Bill. In this example, a single instance is provided by the single video framestore within Bill's domain of control, and this is defined in the desktop profile for workstation Egor. In this case, it is video-c at location Darius.

2) The video control application running on Egor goes to the office manager requesting all the instances of the generic video data service associated with user Ben. In this example, a single instance is provided by the single video framestore within Ben's domain of control, and this is defined in the desktop profile for workstation Gollam. In this case, it is video-d at location Venus.

3) The video control application running on Egor creates an association with the video control service returned during step 1 (video-c at Darius). It then makes a start RPC down this control association quoting the name of the the peer video data
service returned during step 2 (video-d at Venus), and supplying any other parameters relating to the intended connection.

- 4) On receiving the start RPC, Bill's framestore attempts to create a data association with the named peer data service and to start transmission. This fails if the peer location is not contactable (ie the initiator was unable to bind to the specified peer service because either the network is down or the service does not exist at that location), or if the specified peer refuses the call (because it is busy or malfunctioning). Having set up a connection, the workstation plays no further part until the user wishes to modify the connection (if applicable) or close down the connection.

The sequence of events is illustrated in Figure 6.5.

There are a number of points to notice:

- Each step described above depends on the success of the previous step.
- The start RPC made by Bill's video control application succeeds only if transmission is started successfully. That is, the subsequent association.create RPC made by Bill's framestore is nested into the start RPC made by Bill's workstation.
- In practice, the generic service name used to specify a service to the office manager is usually the same as the actual service name offered by the component. Therefore, the process of instantiating a service usually only requires knowledge of the
service location. For example, the video control application knows that the control service name is video-a, and so only needs to discover the location from the office manager to instantiate the service.

6.2.4.2 Setting up a Connection with Prior Notification

A more realistic scheme for setting up a connection involves first notifying the peer user of the initiator's intention to start a call. That is, there should be some prior interaction between caller and callee to confirm that the call is acceptable to both parties. This might be by means of a flashing icon on the screen of the callee's workstation notifying him of an incoming call; he is then offered the choice of either accepting or rejecting it. This is analogous to the bell on a conventional telephone set - the callee has the choice over whether or not to answer.

Referring once again to the above example, the sequence of events is as follows:

- 1) The video control application running on Egor goes to the office manager requesting all the instances of the generic notification service associated with user Ben. In this example, a single instance is provided by the peer workstation (and might involve flashing an icon and displaying a message). The service is defined in the desktop profile for Gollam.

- 2) The video control application running on Egor goes to the office manager requesting all the instances of the generic video control service associated with user Bill. In this example, a single instance is provided by the video framestore within Bill's domain of control, and this is defined in the desktop profile for workstation Egor.

- 3) The video control application running on Egor goes to the office manager requesting all the instances of the generic video data service associated with user Ben. In this example, a single instance is provided by the single video framestore within Ben's domain of control, and this is defined in the desktop profile for workstation Gollam.

- 4) The video control application running on Egor creates a (data) association with the notification service returned during step 1. It then makes a notify RPC to this service and waits for a reply. If this succeeds, the call request has been accepted and the procedure can continue as before.

- 5) The video control application running on Egor creates a control association with the video control service returned during step 2. It then makes a start RPC down this association quoting the name of the peer video data service returned during step 3, and supplying any other parameters relating to the intended connection.
6.2 A Model for the Management of Control

- 6) On receiving the start RPC, Bill's frameset attempts to create a data association with the named peer data service and to start transmission.

The sequence of events is illustrated in Figure 6.6.

![Diagram showing setup of a video connection with prior notification.]

**Figure 6.6: Setting up a Video Connection with Prior Notification**

6.2.4.3 Closing Down a Connection

To close down a connection, the workstation must make an appropriate RPC to a control service within its domain of control requesting that the specified connection be closed down. This may require the workstation to bind (or rebind) to the control service depending on the current state of the control association to the relevant component.

To terminate the data association, the component may explicitly inform the peer by means of an out-of-band RPC or by means of an in-band control packet. Alternatively, it may simply stop transmission and/or reception and leave the peer to time out the connection (on the assumption that some sort of 'dead man's handle' is being used over the connection). The problem of terminating a connection is a difficult one, especially when data is being multicast; this aspect of the problem is discussed more fully in section 5.7 of the last chapter.

6.2.4.4 Management of Multiple Streams

A difficulty arises when trying to manage connections between components that support multiple streams (eg the frameset - they can handle several data associations at
There needs to be some means of identifying one stream from another within a multiple stream component. The general scheme is as follows.

When the workstation establishes a connection between two multiple stream components, it is returned two tokens (or handles) identifying each end of the connection. That is, both the caller's component and the callee's component allocate a private (locally unique) token to identify the connection. These tokens are exchanged during the call set-up procedure so that each component has a copy of both tokens. The two tokens are then returned to the callee's workstation as the result of a successful call set-up operation. The workstation must quote one of these tokens if it wishes to refer to the connection at a later date; if it is talking to the caller's component it must quote the caller component's token, if it is talking to the callee's component it must quote the callee component's token.

The problem arises when the callee's workstation wishes to gain access to the same connection. Suppose, for example, that the callee wants to terminate the connection. Since his workstation had no part in the original call set-up procedure, it will have no knowledge of the tokens that have been allocated.

One solution is for the callee's workstation to request a copy of the tokens from the caller's workstation whenever it requires access to the connection. However, this requires knowledge of the location of the caller's workstation - information that is not necessarily available using this model (unless it has been cached during the notification phase described above).

Another possibility is for the caller's workstation to notify the callee's workstation of the allocated tokens immediately after the call set-up procedure described above. This would require the addition of a special notification service for this purpose. Although this solution fits the model, it involves excessive complexity in managing the exchange of tokens, and requires extra interactions in the call set-up procedure.

The preferred solution is for all multistream components to support a status inquiry RPC. This allows any control entity to retrieve all tokens currently allocated by that component. This approach has the advantage that all status information is maintained by the components themselves and does not need to be cached by remote control entities (which are subject to failure). Therefore, for the price of a little added complexity in the component itself, the management of connections is made more simple and reliable.

To illustrate this scheme, consider once more the configuration illustrated in Figure 6.4. When Bill first established the connection, his workstation would have been returned two tokens identifying the video stream. One token will correspond to the transmitter (Darius), the other will correspond to the receiver (Venus). If Bill wishes to terminate transmission, he can send a stop RPC down his control association to
Darius, quoting the transmitter token as an argument. However, Ben does not have this option, since his workstation has no knowledge of the tokens that have been allocated. The procedure for Ben to close down the channel is as follows:

- 1) The video control application running on Gollam goes to the office manager requesting all the instances of the video control service associated with user Ben. In this example, there is a single instance provided by the single video framestore within Ben's domain of control (Venus), and this is defined in the desktop profile for workstation Gollam.

- 2) The video control application running on Gollam creates a control association with the control service returned during step 1. It then makes a status RPC down this association requesting information on the current status of framestore Venus, in particular on the tokens that have been allocated.

- 3) The video control application running on Gollam then makes a stop RPC down the control association quoting the token corresponding to the video connection to be terminated. The choice of token is down to the user; typically, the list of connections will be displayed on the workstation (either graphically as windows, or textually) and the user must select the connection that is to be closed.

The sequence of events is shown in Figure 6.7.
The above scheme is exactly that adopted for the management of connections between CR framestores. Refer to chapter 4 for more information.

6.2.5 General Comments

Using the above model, the flow of control is confined to remain within the domain of control for each user. Note, however, that there is no mechanism provided to enforce this discipline. The model only works by the conventions adopted by well behaved applications, and there is nothing to stop a rogue application gaining control of any component on the network. Such an environment is clearly unsuitable for secure transactions, but the mechanisms are provided for orderly conduct.

Another important feature of this model is that the office components themselves need have no knowledge of the network environment. In particular, all control information flows from the workstation to the component, and not in reverse. If a user requires status information, an explicit request must be made to the component. A component has no knowledge of who should be controlling it. The reasons for this design decision are to keep component complexity to a minimum, and to keep the model as simple as possible.

A ramification of this is that there is no support for reports from component to workstation. This means that it is not possible to report back asynchronous events such as communication failures. Support for such a capability would require an extension of the model (for example, by registering a reporting service with each component at association set-up time).

Returning to the question of security, any authentication technique would require the component to interact with some sort of authentication server, thus increasing the complexity of the component. Support for a secure office environment would therefore require an additional extension to the model.

A final point to make is that use of this model is intended to supplement existing call set-up techniques, rather than to replace them. Where other mechanisms already exist (such as by using a telephone handset), these may still be used but they are quite independent of the model.

6.3 Implementation

6.3.1 Introduction

This section describes the software that has been written to realise the above model. A sketch of the system is shown in Figure 6.8.
The office manager has been implemented as a Tripos office manager task with an RPC client interface (defined in Appendix D). The office manager relies on the Unison directory service [Chapman 89] for the storage of user and desktop profiles. The modification of these profiles is the responsibility of third-party directory management tools.

Every control entity (workstation) requires an office manager 'stub' which must be resident on every client of the office manager and which handles communication with the office manager on behalf of the client. This stub has been implemented for Tripos machines as a logon task. The office manager stub registers a user with the office manager when the user first logs on to his workstation, and then maintains an active session with the office manager until he logs off.

At the start of day, the office manager reads the static services for all the users registered in that domain. When a user logs on to a workstation, the process of registering with the office manager (automatically performed by the stub) causes the appropriate desktop profile to be read for the workstation and then associated with the particular user. The user's domain of control then encompasses the static services read at start of day, and the dynamic services read when the user becomes active. A client must maintain an active session (ie periodically shake a 'dead man's handle') with the office manager; if this session times out, the office manager deletes the dynamic services on the assumption that the user has logged off or that the client workstation has crashed.
Note that a client of the office manager can be any system that supports the Unity RPC mechanism, and need not be a Tripos machine. In principle, a Unix workstation would be a perfectly acceptable control entity provided it had an office manager stub.

6.3.2 The Office Manager

6.3.2.1 Overview

The purpose of the office manager is to maintain the dynamic bindings between users and office services. To meet the requirement of user mobility, these bindings vary as a user moves about the network.

Although the directory service could be used to maintain such bindings, it is not well suited to this purpose. The directory service has been designed to maintain relatively static information, whereas the bindings in question are dynamic. Furthermore, directory accesses are fairly slow meaning that clients would pay a significant performance penalty during call set-up when several office manager interactions are required (see section 6.2.4.1).

Therefore, a separate office manager task has been implemented to maintain these dynamic bindings in main memory. The directory service is used to maintain the relatively static definitions of user profiles and desktop profiles, a function for which it is ideally suited. The office manager maintains the dynamic office profile for each user.

The office manager reads the static services from the user profiles at start of day. When a user registers with the office manager, his dynamic services are read from the desktop profile corresponding to the control entity which he is using. This binding between user and dynamic services is maintained while the client sustains an active session with the office manager.

6.3.2.2 Binding to the Directory Service

At start of day, the office manager attempts to bind to the directory service in the local Unison domain. The office manager loops until this binding is successful; all logon requests (from client stubs at client initialisation time) are rejected until the office manager has established a session with the directory service. When the binding has been established, the static services are read for each user in the domain and stored as linked lists in main memory.

This directory service session is continuously refreshed to detect directory server crashes. In the event of a directory session timeout, the office manager enters a loop to re-establish the binding. During this phase, all new logon requests are rejected, although
service name lookup requests on existing office manager sessions are serviced as normal.

6.3.2.3 Client Access to the Office Manager

At the start of day, the office manager attempts to advertise a unique service with the secretary. This will fail if another task is already providing this service - the new office manager runs but is not contactable.

A user registers with the office manager by simply creating an association with the office manager service. This is performed by the office manager stub when the user logs on to his workstation. The user name is supplied as an argument to the association.create RPC, allowing the office manager to link the desktop profile for that workstation to the office profile for that user. That is, the office manager reads the desktop profile for the particular workstation from the directory and adds these dynamic services to the office profile for that user. A user must continuously refresh his session with the office manager to prevent his dynamic services from being deleted.

A single user may be logged on to more than one workstation. His domain of control therefore includes the dynamic services offered by all his current workstations.

6.3.2.4 Representation of Profiles within the Directory

Within a user profile directory, users are represented as directory objects. Each user has a set of (group) attributes which defines the static services associated with that user. The structure of a user profile directory is shown in Figure 6.9.

Within a desktop profile directory, control entities (workstations) are represented as directory objects. Each workstation has a set of (group) attributes which defines the services associated with the components controlled by that workstation. The structure of a desktop profile directory is shown in Figure 6.10.

Each attribute corresponds to a generic service. Due to the implementation of the directory, an attribute is identified by an integer rather than by a string. An external mapping has been defined between a generic service name (such as 'telephone') and the corresponding integer (needed to index the particular attribute).

The value of each attribute is a list of service instances of the form <service>@<location>. Note that there may be several instances of a generic service.

[A mechanism that allows a service to be located on the basis of specified properties rather than simply by generic name requires that additional information be associated with a service instance. This could be achieved by making the value of an attribute
point to a 'service object' which would define the particular service instance and the properties relating to that instance. A constraint language might then be defined to allow a client to locate a service based on logical constructs involving these properties [ANS89]. However, the simple scheme described here was considered adequate for the prototype implementation.

6.3.2.5 Structure of Directory Names

Within the directory service, the user profiles for a given domain are stored in a directory named /profiles/<domain>/users (see Figure 6.9). Likewise, the desktop profiles for a given domain are in a directory named /profiles/<domain>/desktops (see Figure 6.9). An example of a user profile directory for domain 'lut-blue' is shown below:

```
Directory=/profiles/lut-blue/users

Object=bjm
  Attribute101  mailbox@lut-blue*sirius
  Attribute113  fileserver@lut-blue*zeus
  Attribute126  fileserver@lut-blue*orion
  Attribute129  fileserver@lut-navy*xerxes
  etc

Object=vjh
  Attribute101  mailbox@lut-blue*sirius
  Attribute113  fileserver@lut-blue*orion
  Attribute126  recorded-voice@lut-blue*sirius
  etc

Object=glu
  Attribute101  mailbox@lut-blue*sirius
  Attribute113  fileserver@lut-blue*orion
  etc

etc
```

Figure 6.9: An Example of a User Profile Directory for Domain 'lut-blue'

Each object refers to a user name. The attributes of an object refer to the static services that can be associated with that user. The attributes are identified by unique integers which are mapped (externally to the directory service) to generic service names. For example, Attribute101 maps to the generic service 'electronic-mailbox', Attribute113 maps to the generic service 'general-filestore'. The value of each attribute is a list of instances of the corresponding generic service.
6.3 Implementation

6.10). This naming strategy was adopted to provide a consistent name structure for profiles based on domain. The 'domain' component ensures that profile names are global, and may therefore co-exist in the same directory if required. This provides the flexibility of allowing a directory service to handle an office manager from a remote domain.

This naming structure requires that the office manager is able to discover the name of the domain in which it is running. This is information held only by the secretary for each domain. In order for the office manager to be able to acquire this information, the secretary stub interface [Chapman 89] has been extended to include a mydomain call to return the domain name. This name is used by the office manager when it first starts up in order for it to be able to construct the names of the user and desktop directories to which it requires access.

```
Directory=/profiles/lut-blue/desktops

Object=cadmus
  Attribute1       tqs-c@lut-blue*tel34
  Attribute1       tqs-d@lut-blue*tel46
  Attribute2       tqs-d@lut-blue*tel34
  Attribute2       tqs-d@lut-blue*tel46
  Attribute6       notify@lut-blue*cadmus
  Attribute10      talk@lut-blue*cadmus
  Attribute27      lsw@lut-blue*faraday
  etc

Object=egor
  Attribute1       tqs-c@lut-blue*tel27
  Attribute2       tqs-d@lut-blue*tel27
  Attribute6       notify@lut-blue*egor
  Attribute10      talk@lut-blue*egor
  Attribute19      scanner@lut-blue*remus
  Attribute27      lsw@lut-blue*faraday
  etc

etc
```

Figure 6.10: An Example of a Desktop Profile Directory for Domain 'lut-blue'

Each object refers to a control entity (workstation). The attributes of an object refer to the services that can be associated with that control entity. The attributes are identified by unique integers which are mapped (externally to the directory service) to generic service names. For example, Attribute1 maps to the generic service 'telephone-control', Attribute27 maps to the generic service 'laser-printer'.

The value of each attribute is a list of instances of the corresponding generic service.
6.3 Implementation

The addition of the `mydomain` RPC has required a modification to the secretary and secretary stub code. For Tripos machines, the secretary stub has been programmed to make a `mydomain` call at start of day. This name is then stored in the `rootnode information vector`, thus allowing client applications to read the domain name without having to interact with the secretary or secretary stub.

6.3.2.6 Looking up a Service

Having logged on to the office manager, a user can make a lookup request by sending a `lookup` RPC down the association. The user specifies the generic service name and the fully qualified name of the peer user (user name and home domain name). The home domain name is only mandatory if it specifies a foreign domain. The office manager searches for all the instances of the generic service for the specified user. If the specified domain is local, then the office manager can deal with the request itself. If the domain is foreign, the office manager passes the request onto a peer office manager in the specified domain.

If the lookup is successful, the user is returned a `list identifier` and a count of the number of elements (service instances) on the list. The user must then quote this identifier in the subsequent `nextstring` calls to the office manager that are necessary to retrieve each of the values on the list. The office manager uses the list identifier to index the appropriate list internally. The state of the list is maintained in terms of the position of the pointer to the current element, and this is updated after every successful read operation. The list is automatically deleted when all the values have been read.

This mechanism is used to get around the problem of returning a variable length array of strings. The RPC implementation does not support an 'array of strings' type. Although there is no problem in principle for supporting such a type, there are practical considerations relating to memory allocation that make such support unattractive. Furthermore, some may have philosophical objections to supporting such a composite type.

This list identifier technique is the same as that used in the implementation of the Unison directory service [Chapman 89]. A directory client wishing to read the value of a composite type (eg an array of strings) must first obtain a list identifier and then make successive calls to the server to retrieve the component values.

If the specified home domain is other than the local domain, the lookup request is passed on to the peer office manager for that domain. An office manager maintains a set of `casual associations` with peer office managers. That is, when an association is first created between peer office managers, the connection is left open in case further lookup requests require use of the same association. A 'dead man's handle' is started on such connections, and if there is no traffic on them within a given period
they are deleted. Office managers that are required to interact frequently to satisfy lookup requests are saved the overhead of setting up a fresh association for each interaction. This is merely a performance optimisation since if there is no current association with a peer office manager (or if the existing one is no longer valid) a new one will be created. It is for this reason that peer associations are termed ‘casual’.

For example, suppose user ptc wishes to look up all the instances of the generic telephone control service telephone-control for user mdc whose home domain is the local domain lut-blue. User ptc makes a lookup request to the office manager specifying the generic service telephone-control and the user name mdc. The home domain parameter is null since it refers to the local domain. Suppose that the office manager has knowledge of two instances of this service for user mdc (that is, there are two telephones in mdc’s current domain of control). In this case, ptc is returned a list identifier and an element count of two. User ptc must then make two subsequent nextstring calls to the office manager to retrieve the two instances; they might be tqs-c@lut-blue*tel31 and tqs-c@lut-blue*tel56. These are in the form <service>@<location> and can be made into a suitable form to be presented to the secretary as part of an association.create RPC.

Suppose now that user ptc wishes to look up all the instances of the same generic telephone control service for user dlt whose home domain is camb-blue. User ptc makes a lookup request to his local (lut-blue) office manager specifying the generic service telephone-control, the user name dlt, and the domain name camb-blue. The office manager realises that this is a foreign domain, and searches its table of casual peer associations to find an existing association to the peer office manager in that domain. If an existing association is not found, a new association is created. In either case, a peer lookup request is made down the association quoting the generic service name, the user name, and the domain name. The peer office manager then handles the request in the same way as before. Suppose the camb-blue office manager has knowledge of one instance of this service for user dlt. The lut-blue office manager is returned a list identifier and an element count of one, and must make a single subsequent peer.nextstring call to retrieve the value from the camb-blue office manager. This value is in the same form as above and might be tqs-c@camb-red*tel62. User ptc is then returned a list identifier and an element count of one, and he must make a subsequent nextstring call to retrieve the value from his local office manager.

Note that there is no guarantee that a service returned to a client as the result of an office manager lookup request actually exists and is registered with the secretary. For example, although ptc has located dlt’s telephone service, there is no guarantee that dlt’s telephone is actually switched on and that the service is registered with the secretary. Given the current network architecture, the only way that such a guarantee could be made would be by combining the functions of the secretary and the office manager. By modifying the current network architecture, one solution would be to remove nameserver
functionality from the secretary and have it provided by an office-manager-like entity. Such a proposal is considered later in this chapter.

Note also that the requirement to specify a user's home domain name might seem to violate the principle of inter-domain location transparency. However, this is purely a consequence of the particular user naming scheme adopted by the model. User names are global, but for organisational convenience they are partitioned on the basis of the Unison domain name. This gives complete freedom to naming authorities within each domain for choosing (locally unique) user names. There is no implication that a service instance will be located within a user's home domain.

6.3.2.7 The Who Facility

An additional service provided by the office manager is a who service. This allows a client of the office manager to discover the names of every user currently working in the specified domain (or, by default, the local domain), the names of the workstation on which they are working, and the times at which they logged on to the system.

6.3.3 The Logon Task

6.3.3.1 Overview

The logon task is a Tripos implementation of an office manager stub. It provides the mechanism for a workstation to bind to its office manager. It also provides a Tripos packet interface to the office manager, allowing a client task to interact with the office manager using the standard Tripos inter-process communication mechanism.

6.3.3.2 Binding to the Office Manager

At start of day, the logon task attempts to bind to the local office manager. The logon task loops until the binding is successful; all packet requests from client Tripos tasks are rejected until a valid office manager session has been started. When a successful binding has been established, the workstation user is said to be registered with (or 'logged on' to) the office manager.

The session is then continuously refreshed partly to enable the office manager to confirm that the workstation is still functioning, and partly to enable the workstation to detect office manager crashes. In the event of a session timeout, the logon task enters a loop to re-establish the binding. During this phase, all packet requests from client Tripos tasks are rejected.
6.3 Implementation

6.3.3.3 Choice of User Name

An essential parameter in the binding request is the name of the user as recognised by the office manager. This allows the office manager to form its internal binding between user name and dynamic services.

In this implementation, the user name recognised by the office manager is that used when logging on to the workstation. For example, if a user logs on to his workstation as bjm, there should be a user profile for user bjm to be read by the office manager when it starts up. In this example, bjm is the user name that is included in the binding request from the workstation to the office manager.

The facility for a client program to discover the name of the workstation user has required a slight modification to the Tripos system. When a Tripos machine starts up, one of the functions of the initialisation program (cli-init [Richards 79]) is to prompt for the workstation user name; various personal initialisation sequences are then run and assignments are made on the basis of the user name supplied. This initialisation program has been modified to store the user name in the rootnode information vector. The logon task then uses this name as the parameter supplied during the binding request to the office manager.

6.3.3.4 The Tripos Packet Interface

The logon task packet interface allows a client task to perform name lookups or make who queries. In general, the result of such an operation is an array of strings. The logon task generates such an array on behalf of the client by making successive calls to the office manager to fetch the appropriate number of elements. The result of an operation is therefore a list of strings in the same format as that used internally by the office manager. A BCPL list handling library has been written to enable client programs to access and manage such lists.

6.3.4 Evaluation and Limitations

The performance of the above implementation has proved to be highly satisfactory. Although a service lookup operation may involve many network transactions between peer office managers and between the local office manager and office manager stub, there is no noticeable overhead. This is because all office manager data is stored in main memory, removing the inevitable overheads imposed by disk accesses (such as those experienced when using the directory service). The good performance also reflects the efficiency of the network software in client machines and bridging components.

Another cause for satisfaction was the performance of the binding strategy used by the office manager to bind to the directory service, and by the logon task to bind to the
office manager. This strategy was found to cope well with server crashes - the binding was simply re-established without the client being dragged down by the server.

On a similar subject, the binding strategy between peer office managers also proved to work well. This involves the use of casual associations that remain open while traffic is flowing, but are timed out if inactive for a certain period. An office manager which interacts frequently with a particular peer is thus saved the overhead of establishing a new association for each transaction. This proved to be a useful way of managing peer associations to maximise performance.

One limitation in the implementation of the office manager was the lack of support for an aliasing scheme for user names. This would have been particularly useful for referring to foreign users without needing to specify their home domain name. For example, it might be convenient to associate an alias such as vicky with the user 'vhj from home domain lut-blue', especially if frequent contact is required with that user. The provision of such an aliasing scheme is not a problem in principle, but merely a question of implementation effort.

A more fundamental limitation in the model itself is the lack of support for inter-domain migration-transparent access. This is considered in section 6.5 below.

6.4 An Example Application using the Office Manager

6.4.1 Introduction

In order to give the reader a better understanding of how the above system is used, a simple application using the office manager is described below. This is a talk facility, along the lines of the Unix facility of the same name. By typing "talk to <user> [from <domain>]", a byte stream is opened to the appropriate talk service allowing the initiator to type messages onto the screen of the peer workstation.

6.4.2 Description

The facility comprises two programs: the client half and the server half. The client program is used to initiate a talk connection to a peer user. It is explicitly invoked from the command line, and once the connection has been established it provides a command line prompt for entering messages. Subsequent characters typed into this shell are copied to the remote screen. The server program is a client of the Tripos BSP (bytes stream protocol) handler. The talk service is registered with this handler at start of day, and the program is automatically invoked on the server machine when a BSP connection is made to that service.
The client program works in the following way. The command line is parsed to obtain the peer user's name and, optionally, his home domain name (the local domain is assumed by default). The program then sends a Tripos packet to the logon task requesting a list of all the instances of the generic service 'talk' associated with the specified user and domain. This causes the logon task to issue a lookup RPC down its association with the office manager; if this association does not exist, the packet request is failed outright.

The office manager performs the lookup request on the specified parameters, if necessary interacting with a peer office manager in the manner described above (if the peer user's home domain is other than the local domain). Assuming that the lookup operation is successful, the logon task is returned a list identifier to reference the resulting list within the office manager. The logon task then retrieves each element on the list, and copies it to a newly created private list within client memory. The packet is then returned to the client task with a success notification and a pointer to the newly created list of services.

If there is more than one instance of the service (for example, if the peer user is logged on to more than one workstation), the client program chooses the first instance on the list but prints a warning message to the effect that other instances exist. This is analogous to the Unix talk program which takes similar action in the event of the specified user being logged on to more than one terminal in the system.

The client then supplies the service instance to a BSP library routine which opens a BSP stream to the specified service. This routine causes the BSP handler on the peer machine to invoke the talk service program that has been previously registered. The server program prints a message on the screen of the peer workstation to the effect that there is an incoming talk connection from the initiator user at the initiator location. Messages typed at the client program prompt then appear on the screen of the peer workstation.

The peer user may reply to the initiator by invoking his client talk program supplying the user name and domain name of the initiator. This causes a separate BSP connection to be opened in the opposite direction in the same manner as described above.

6.4.3 Discussion

The first point to notice is that this is an improvement over the Unix program for the following reason. To talk to a remote user on a Unix machine, a user must supply the name of the host on which the user is working. However, using the facility described here, only the (fully qualified) name of the user is required. This is because the office manager handles the mapping between user name and host and thereby provides location transparency.
A second point to notice is that communication with the office manager is handled by the logon task, and peer communication is handled by the BSP handler. This means that neither the client or server programs need to deal with the network directly; in particular, neither program has to work with the RPC package. This means that the complexity of the implementation is kept to a minimum.

A final point worth mentioning is that this simple facility could easily be extended to provide a multipoint chat facility. There is nothing at the moment to prevent a user from having multiple talk connections, but an improvement would be to tag incoming messages with the name of the initiating user if multipoint working is being anticipated. Furthermore, the command line prompt should also be tagged with the name of the peer user in order to support multiple outgoing connections from separate talk tasks.

6.5 Inter-domain Migration-transparent Access to Office Services

6.5.1 Overview

The basic model described above does not support inter-domain migration-transparent access to office services. That is, if a user moves outside his home domain, although his static services are available as before, his dynamic services are not generally accessible except to those users within his new domain.

This section describes an extension to the above model in order to support inter-domain migration-transparent access. It has not yet been implemented.

6.5.2 The Proposed Scheme

The scheme requires that a user's home office manager be informed of all the domains in which the user is currently working. This is done by arranging that an office manager registers the location of a foreign user with the home office manager whenever such a user logs on to the system. Furthermore, this name must be withdrawn from the home office manager when the user logs off from the system. This scheme means that access to the home office manager will always provide access to the user regardless of his location.

This is achieved in the following way. The office profile maintained by the home office manager (or master office manager) is termed the master profile. The office profile maintained by another office manager in a domain in which the user is working (a shadow office manager) is termed a shadow profile.
Associated with either sort of profile is a *home domain* parameter, and a *locations* parameter. The value of the *home domain* parameter is the name of the domain which can be considered the home base of the user. This parameter is only of use to a shadow office manager. The value of the *locations* parameter is a list of domains in which the user is currently working. This parameter is only of use to the master office manager.

When a foreign user registers with a (shadow) office manager, a shadow profile is created for him by the office manager. In addition, the user's presence is registered with his home office manager - the new domain name is added to his master profile. The home domain parameter in his shadow profile takes the value of his home domain name. The dynamic services are then registered with this shadow profile. This is illustrated in Figure 6.11.

A user within the same domain as a foreign user may access the dynamic services by simply quoting the foreign user's name and the generic service name (i.e., a fully qualified peer user name is not required). A user within the home domain may access the...
dynamic services in the same way; this time, however, the home office manager inter-
acts with the shadow office manager specified in the locations list in order to dis-
cover this information. A user in any domain can access the dynamic services by quot-
ing the fully qualified peer user name in the usual way.

Static services are accessed in a similar way. A user within the same domain may
access the static services by simply quoting the foreign user's name and the generic
service name. This time, however, the shadow office manager must interact with the
home office manager specified by the home domain parameter. A user within the home
domain may access the static services directly by simply quoting the user's name
and generic service name. Once again, a user in any domain can access the
static services by quoting the fully qualified peer user name in the usual way.

6.5.3 The Lookup Algorithm

The general algorithm used by an office manager to look up a generic service is as
follows. If this is a master office manager, the master office profile is searched
first. Peer lookup requests are then made to each of the shadow office managers
(if any) listed in the locations list of the master profile. In this way a list is built
up of all the instances of the generic service.

If this is a shadow office manager, a single peer lookup request is made to the master
office manager specified by the home domain parameter. This causes the master of-
vice manager to make recursive peer lookup requests to all the shadow office managers
(including the one from which the request has originated) listed in the locations list
of the master profile. Once again, a list is built up of all the instances of the generic
service.

An office manager knows that it is the master for a given user if the domain name
specified in the user's home domain parameter is the same as the domain in which the
office manager knows that it is running.

6.5.4 Maintaining Dynamic Services

It is the responsibility of every office manager to maintain the dynamic services
within a domain. In particular, when an office manager session times out, the office
manager must delete the dynamic services registered for that workstation.

Office managers that support shadow profiles have the additional responsibility of inform-
ing the home office manager whenever dynamic services are added or deleted. This is
to enable the home office manager to maintain the list of shadow locations in the master
profile. In practice, an office manager will support a combination of master and shadow
profiles.
6.6 Proposal for a Modified CFR Architecture

6.6.1 Background

There has been much recent thought on a modification to the Unison Cambridge Fast Ring architecture that has relevance to some of the ideas presented here concerning access to services. In particular, there are proposals to do away with the Unison secretary service for client CFRs. This has grown out of the desire to support call set-up mechanisms other than the existing method based on the S-association and association.create RPC described in chapter 3. There is also the feeling that the current secretary is an excessively complex object, needing to provide routing, call set-up and nameserver functionality. Refer to [Cooper 89] and [McAuley 89] for a full discussion on this subject.

The aspect of this development that is relevant in this context is the desire to entrust the function of name resolution currently provided by the secretary to a separate network entity. This provides the opportunity for combining the functionality of the current office manager with that of a conventional nameserver. This would solve the problem encountered above to do with the fact that a service returned to a client as a result of an office manager lookup request may not actually exist in that it has not been offered by a server. Combining the two functions would ensure that the result of a successful office manager lookup request would be some currency that would actually be valid (at least for a certain period) in a subsequent binding request from a client.

6.6.2 A New Mechanism for the Management of Control

The new scheme requires that office components wishing to conform to the model should create a special association with the office manager at the start of day. This requires the use of a very low level protocol to exchange port numbers; the office manager must be at a well known station address and must be listening on a well known port. This special association is christened an OM-association, and is analogous to the S-association currently used by the secretary.
Components that wish to advertise their services to users of the model must register these service names with the office manager in a similar fashion to the way components advertise their services with the current secretary. The intention is to use a very simple protocol to keep this and all such interactions with the office manager as simple as possible. This protocol would involve the exchange of packets on the OM-association described above. This approach contrasts with the current design of the secretary which requires all network components to support the RPC protocol in order to be able to communicate with the secretary (a significant implementation burden for a simple component such as a telephone). The purpose of registering a service is to allow the office manager to maintain a list of all the services currently available within a domain.

Control entities have the additional requirement of supplying the user name as a parameter to the OM-association set-up request. This enables the office manager to maintain the dynamic binding between user and control entity (workstation).

User profiles and desktop profiles are defined in exactly the same way as before. That is, there is a directory entry for all the static services that may potentially be associated with each user in a domain. Likewise, there is a directory entry for all the dynamic services that may potentially be associated with each control entity in a domain.

However, a user's domain of control is now defined by the combination of his static services and dynamic services that are currently available. The office manager can check that the service defined in a user's office profile actually exists since it will have been previously registered with the office manager by the server.

The result of a successful service lookup request will be whatever currency is required by the call set-up mechanism to be used to bind to the service. That is, the value of a generic service name can be in whatever format that is desired by the client for accessing the service.

Note that this model exists over and above any call set-up mechanism that may be employed. Indeed, it can exist alongside other models for accessing service names. If a host is able to discover the network address of a service by some other means, then it is perfectly permissible for it to opt out of the above scheme. The intention is simply to provide the user with a high level of abstraction by allowing him to access services based on user name.

6.7 Summary

This chapter discusses the subject of the management of control in a distributed office environment. A model is presented that provides a framework for the division of network space into domains of control. By conforming to the conventions of the
model, office users are granted complete control over the components within their domain of control, but denied control over other components on the network.

An implementation of the basic model is described involving the design of a prototype office manager and an office manager stub for Tripos. The implementation of the model as a dynamic directory service provides intra-domain location- and migration-transparent access to office services on the basis of user name. The basic model also supports inter-domain location transparency, and an extension is proposed to support inter-domain migration transparency.

The performance of the model has been evaluated, and a simple application has been described that makes use of its features. This application was seen to have an advantage over conventional programs due to the transparent access capability provided by the office manager.

A modification to the Unison CFR architecture is then proposed involving the roles of the secretary and office manager. The suggestion is that the nameserver functionality provided by the current secretary be combined with the directory functionality of the office manager to provide an improved office manager capability. This new office manager would still be central to the mechanism for the management of control, and it would also provide a powerful means of locating services based on user name.
Chapter 7
Conclusions

7.1 Introduction

This thesis has been concerned with the provision of multimedia services in a distributed office. It describes the practical experience gained in designing, implementing and using such services in an experimental office environment. This environment comprises a number of geographically dispersed offices each based on a local area network and interlinked by an ISDN. A collection of custom-built components has enabled experimentation with a variety of media.

The underlying theme of this work has been the integration of media at the user level. The forthcoming generation of digital networks offers the promise of integration at the network level, and this provides great benefits in terms of uniformity and efficiency of communication. However, a higher level of integration is necessary in order to be able to present these media to the user in a coherent fashion. The integrated office is one in which this level of integration is achieved. Although the realisation of such an office is still some way off, it is hoped that the experience gained during Unison will contribute towards that goal.

Much effort is currently being directed towards the development of multimedia ISDN terminals [Yamaguchi 86] particularly aimed at the basic rate ISDN subscriber. These systems can combine the transmission of voice, video (at about 10 frames per second) and data over a '2B + D' basic rate interface. Although these systems are of great technical interest (in terms of the coding schemes that are used), they have not been conceived with the aim of integration in mind. These new types of media will only be of real benefit when they can be properly integrated into the information processing environment of the office. Such integration will allow voice and video to be manipulated along with other forms of business information.

There are a great many human factors relating to the problem of user level integration. For example, how should a voice editor be structured? How should video data be presented - in a workstation window, or on a separate display? More importantly, how should the user interact with his workstation? What should be the appearance of the user interface? Much research is being done in this area ([Hoffner 88], [Hodges 89]), and it is obviously of critical importance. The work described in this thesis has been primarily directed towards system and network level issues, although some consideration has been given to user interface design.
This chapter presents some ideas on the design of an integrated office in the light of the experience gained from the experimental multimedia offices. The following section discusses the subject of network support for an integrated office, particularly with respect to the diversity of traffic types that must be expected. The next section suggests a suitable architecture for the integrated office based on a distributed computing approach. The penultimate section considers the sorts of multimedia services that might be available in such an office. The final section indicates the likely direction for future work in this area.

7.2 Network Support

7.2.1 Local Area Support

The experimental offices described in this thesis were based on local area networks. LANs have a number of general properties that make them useful for interconnecting components within an office. For example, they offer high transmission speeds, low delays, and low error rates. As such, they provide an ideal basis for intra-office communications and resource sharing (in terms of information and physical peripherals).

The current generation of LAN runs at about 10 Mbps and has already proved useful in the office for communications and distributed computing activities such as file sharing and remote printing. The next generation of LAN (running at nearer 100 Mbps) has the potential of providing simultaneous support for the full spectrum of traffic types including packet voice and video. A LAN that can simultaneously meet the conflicting requirements of all types of traffic can be classified as a fully integrated network.

A difficult requirement for an integrated LAN to meet is that of the provision of minimum (or sometimes fixed) bandwidth guarantees for real-time traffic. Furthermore, the high bandwidth required for real-time video places great demands on the network. Integrated LANs under development in the research laboratories often take a hybrid approach to the problem of support for real-time media and computer data traffic by using a combination of circuit-switching and packet-switching techniques. These solutions usually require specialist hardware and are typically expensive to implement.

The slotted rings used in the experimental offices go some way towards supporting integrated services by providing an inherent bandwidth sharing mechanism (by preventing a transmitter from using successive slots) at a reasonable cost. Experience with the Cambridge Fast Ring (running at about 50 Mbps) confirmed its suitability for the simultaneous support of a modest amount of real-time traffic (voice and slow-scan video) as well as distributed computing traffic (generated from a network fileserver). While the CFR may not be classified as a fully integrated LAN, it does represent a first approximation to such a network which has been implemented at a sensible price.
It is interesting to note that the performance of the experimental network services was invariably limited by the performance of the network interface and interface processor rather than by any bandwidth constraints imposed by the network. For distributed computing applications, such as using a remote disk or remote document scanner, the network overhead was not generally significant. However, the speed of the network interface and the host processor bandwidth were certainly the limiting factors on the performance of the various video systems.

7.2.2 Wide Area Support

Each of the experimental office LANs was interconnected by the Unison wide area network based on a prototype ISDN. The Unison network was conceived with the aim of extending the functionality of a LAN to a wider geographical area. The Unison exchange provides client LANs with a very basic inter-site packet relaying service over primary rate ISDN. The 32-byte CFR slot is the basic unit of transmission, and there is no network support for flow control or error recovery. This minimal level of network support is exactly that required by real-time services; reliable data transfer services are expected to exercise higher level protocols on an end-to-end basis.

The Unison philosophy on multi-service network design has much in common with current thinking on ATM techniques [Vorstermans 88]. In particular, the use of a small, fixed-sized packet (or cell) and the minimal level of support provided by the network are both fundamental characteristics of an ATM network. ATM aims to provide the bandwidth utilisation and connectivity benefits of packet-switching with the low delay and jitter characteristics of circuit-switching. Proponents of ATM believe that such techniques will provide the basis for fully integrated services without recourse to circuit-switching or to the sorts of hybrid solutions mentioned above.

Experience with the Unison network confirmed its suitability for the transmission of a variety of traffic types including voice, slow-scan video, and conventional computer data [Murphy 88]. The performance of most office services over the wide area network was comparable to that over a local area network assuming that sufficient ISDN bandwidth was available (is assuming that the U-channel size was sufficiently large). In practice, no single application generates data at more than the maximum primary rate bandwidth, although multiple clients may have a combined bandwidth requirement in excess of that available from the ISDN. If there is insufficient ISDN bandwidth, traffic is significantly delayed due to back pressure being applied by the exchange. The delay imposed by the network in 'non-overload' conditions (about 7 ms) is only significant for real-time voice services.

The Unison network provides an ATM overlay on top of the synchronous transfer mode service provided by the circuit-switched ISDN. Although native wide area ATM networks may emerge in years to come, ATM overlays on top of synchronous services are like-
ly to be around for some time. The realisation of such overlays requires a mechanism for the management of the underlying synchronous network resources for the provision of the ATM service. For example, inter-site bandwidth allocations need to vary dynamically to reflect instantaneous ATM connectivity requirements. In practice, the configuration of the underlying circuits to new sites will involve a start-of-day delay, and the impact of such delays on the ATM service has yet to be evaluated.

Other problems facing the ATM designer include those of resource management and quality of service. In order to meet the delay and jitter requirements of real-time traffic, a mechanism must be provided to safeguard the network against overload. This may involve call blocking for real-time connections, since this approach is the only means of flow control for delay-sensitive traffic. Such a mechanism requires that in-band switching components continuously monitor network utilisation and adjust their behaviour accordingly.

The efficient implementation of ATM switching components is fundamental to the overall performance of the network. Experience with the Unison network confirmed that applications performance over the wide area was generally limited by the bridging components (especially the portal) rather than by bandwidth constraints imposed by the ISDN. As wide area bandwidth continues to increase, further improvements in switching technology will be required in order to realise the potential benefits.

7.3 Office Architecture

The experimental offices described in this thesis are based on a loosely coupled collection of multimedia components with the local area network as the only means of communication. Each component is designed to handle a single medium well, rather than several media badly. This contrasts with the design being adopted by a number of multimedia terminals in which the various media are handled by a single system. The arguments for such a distributed architecture are summarised below.

Firstly, there are often significant performance advantages in implementing a complicated system as a collection of subsystems which have each been designed with one function in mind. In the context of a multimedia workstation, these performance issues relate to host processor bandwidth, network interface performance and network driver design. A single host processor is unlikely to be able to simultaneously handle a variety of streams such as voice, video and graphics. Likewise, a single network interface would be stretched to meet the delay and throughput requirements of the full range of traffic types. Furthermore, the transmission requirements of different types of media vary so much (in terms of quality of service, etc) that common network driving software is likely to represent a poor compromise between a number of conflicting requirements.
Secondly, distributed systems can offer the benefit of improved reliability. The individual subsystems are, by definition, more simple than the entire system and therefore tend to be less prone to failure. Furthermore, there are opportunities for replicating components to provide redundancy and fault tolerance. In the context of an office, a number of network printers might be provided so that a single failure would not disrupt all printing activity.

Thirdly, the modular design of distributed systems lends itself to ease of upgrades. For example, an extra fileserver can be added to improve disk availability. Furthermore, components can be moved around in response to organisational changes.

Finally, distributed systems can offer economic benefits by encouraging competition between manufacturers for the supply of subsystems. This is particularly true when subsystems conform to some sort of standard (such as ‘Ethernet’ or ‘Postscript’). Clearly, the greater the degree of standardisation, the better the opportunities for competition.

However, there is an apparent contradiction between the desire for user level integration and the proposal for a distributed office architecture. The ultimate aim is to present a variety of media to the user in a coherent fashion, and this is made more difficult if the media are handled by separate office components. To avoid this problem, many research groups use complicated multimedia workstations that include hardware for handling graphics, voice and video all within the same system, but this approach contravenes the distributed design principles outlined above.

An alternative strategy for providing user level integration in a distributed office environment is by use of control and co-ordination (or ‘orchestration’) services to control and manage logically related streams (such as voice and video streams in a video-telephone call) emanating from physically discrete components. Some of the work described in this thesis has been to do with the control of such components, but no consideration has been given to the problem of synchronising logically related streams.

A multimedia workstation is a logical concept in this environment, comprising a collection of multimedia components. Although these components are physically discrete, they are logically bound together in software. A mechanism has been developed for maintaining this logical binding between workstation components, and for maintaining the dynamic bindings between ‘workstations’ and users. This mechanism also provides a means of locating peer services on the basis of the office user to whom they belong.
7.4 Multimedia Services

7.4.1 Introduction

This section considers the sorts of services that might be useful in the office of the future. It is based on experience gained with the experimental services described in this thesis. While the list will certainly not be exhaustive, it gives a personal impression of the types of activities that should be possible in an integrated office.

7.4.2 Distributed Computing Services

Distributed computing services are seen as the most fundamental class of services in the integrated office. These include familiar facilities such as file sharing, remote terminal access, remote printing, etc. The new generation of distributed operating systems will provide better performance, resource utilisation and load balancing capabilities. Distributed computing activities are now the subject of standardisation efforts under the banner of 'Open Distributed Processing'.

As well as providing these traditional computing facilities, distributed computing techniques are seen as essential for the control and co-ordination of the multimedia components that make up the office.

7.4.3 Data Communications

There remains the need for data communication services such as electronic mail, teletext and facsimile. Although the distinction between distributed computing services and data communications services is becoming increasingly blurred, this class of services is mentioned explicitly to emphasise its importance. Much of the applications level standardisation work (eg X400, FTAM) is being conducted in this area; refer to [McConnell 89] for a comprehensive summary of such work.

7.4.4 Voice Services

Speech communication is seen as an essential tool in the office. There are a number of advantages in using packet voice over the office LAN rather than using a conventional PABX. Firstly, there are potential cost benefits in using the same network for voice and computer data traffic. Secondly, the integration of voice equipment with computing equipment permits a new class of application that can incorporate voice as well as text and graphics. Thirdly, since voice can be represented as a 'string of bits', it may be manipulated like any other computer data. For example, it may be edited, stored on disk, played back, etc. Finally, there is great scope for improving the user interface to the telephone system using the power and flexibility of the workstation (for example, using windows and icons) instead of the conventional handset. Further-
more, on-line directory services may be used to obviate the need to remember telephone numbers.

The current range of speech codecs typically generate constant bit rate PCM speech at 64 kbps, and are geared towards fixed bandwidth channels of the sort provided by the circuit-switched ISDN. The move towards ATM networks should encourage the adoption of more efficient variable bit rate coding schemes (such as ADPCM) which can take advantage of the packet nature of ATM to improve bandwidth utilisation.

Conventional telephone systems are likely to be supplemented by high quality wideband speech systems. Experience with the experimental Unison wideband speech systems has confirmed their superior quality and their desirability for certain applications. These systems are primarily intended for teleconferencing activities, and are normally used with a microphone and loudspeaker rather than a hand-held telephone set.

7.4.5 Still Image Services

Support for the capture, manipulation and display of still images is also considered important. Paper-based images may usefully be digitised using a document scanner, whereas more general images may be captured using a framestore. The resolution available from a scanner is usually superior, and text that is scanned in this way can often be converted from the bit-map image into ASCII text using appropriate OCR software. Digitised still images may be edited, stored on disk (for example, in an image library), and incorporated into desktop publishing documents.

Practical experience gained of these images has highlighted a number of points that should be taken into consideration by the office designer. Each image represents a substantial amount of data, and this has repercussions on storage requirements, host memory requirements, etc. Also, processing these images (say for printing or for display within a workstation window) can take a significant time. Furthermore, transmitting such images (especially scanned documents) requires sufficient network bandwidth to ensure adequate performance.

7.4.6 Video Services

The role of video in the office of the future is still not clear, but applications are likely to develop as suitable technology becomes available. Current networks and host hardware are not able to support full frame rate video, and this is seen by many as being critical for the widespread take-up of this medium. Videoconferencing has been identified as a possible applications area, although some are doubtful of its benefits. Video-annotated documents may prove to be more popular.
Practical experience with several Unison video systems has revealed a number of practical problems relating to this medium. The most obvious problem is that the amount of information associated with even a slow-scan video stream is quite significant, and this places great demands on the network and host processors. Practical limitations demand a trade-off between quality (in terms of colour and resolution) and frame rate. Slower, colour video systems are suitable for some application areas (for example, surveillance), whereas faster, monochrome systems may be better suited to others (say, conferencing).

The same comments about the integration of voice apply to the integration of video. Once digitised, video data can be edited, stored on disk, and transmitted over a common communications network. The adoption of ATM techniques is likely to see a move away from the current generation of variable quality, constant bit-rate coding schemes towards a new generation of constant quality, variable bit-rate systems [Chin 89]. The implementation of these algorithms on silicon may herald the introduction of video into the office.

7.4.7 Distributed Blackboard Services

Although only limited practical experience was gained in using a distributed blackboard, it was sufficient to confirm the usefulness of this type of service. The basic concept is that a group of users can have graphical access to a common workspace which is displayed on the screen of each workstation. The act of creating a graphic on one workstation causes it to be written on the blackboard and copied to other blackboard users. This was found to be a useful means of interactively communicating with one or more people, particularly when used in conjunction with voice. The blackboard can also be used to annotate other objects of interest such as a still image.

The design of a multipoint blackboard system (ie involving three or more people) raises a number of issues that are relevant to multipoint conferencing activities in general. For example, there may need to be some form of floor control to preserve order. Various mechanisms have been proposed, including those using speech (he who shouts first wins), but there are clearly important human factors to be considered.

Another issue is that of replication of data. One solution is to provide a central entity which is responsible for the distribution of data and to which all hosts must transmit (this is the design adopted in [Waters 87]). Indeed, a design was considered by Michael Ellis and the author using the X Windows environment in which the blackboard would be implemented as a single X client talking to multiple X servers. The problem with this approach is that it does not scale well, especially over a wide geographical area. A more distributed approach using some form of multicast service may be more appropriate, but there is clearly a need for some form of locking mechanism in this environment to maintain data consistency between participants.
7.5 Future Work

This thesis has described a variety of multimedia services implemented in an experimental office environment. A great deal of additional work is required in order to be able to provide fully integrated multimedia services in a 'real-world' environment. It is only when comprehensive integration can be achieved that these new types of media will have more than merely demonstration value and will be of some real benefit in the office.

A fundamental area of research is in the provision of multi-service networks that can meet the communications requirements of the integrated office. Although the widespread installation of optical fibre is paving the way for the implementation of such networks, challenges remain in such areas as the design of suitable network access mechanisms and in the provision of high performance switching components.

In the local and metropolitan area contexts, the Fibre Distributed Data Interface Phase 2 (FDDI-II) and the Distributed Queue, Dual Bus (DQDB) network standards currently under consideration by US standards bodies address the problem of multimedia support. Both these networks employ a hybrid access mechanism which can offer isochronous circuit-mode access for real-time media and traditional packet-mode access for data. Their suitability as multi-service networks awaits assessment, and such an investigation would appear to be a worthy topic of research.

In the wide area context, the possibility that Asynchronous Transfer Mode (ATM) might be the chosen technology for the implementation of broadband ISDN is currently receiving much attention. Proponents of ATM (including the author) believe that such techniques could provide the basis for multi-service support in any geographical context, although it is yet to be proven that such networks can be engineered to meet the stringent delay and jitter requirements imposed by real-time media. A fruitful area of research could involve a prototype implementation of such a network and an experimental assessment of its performance.

Another important subject is the design of the multimedia workstation. This thesis has presented the arguments for a distributed architecture in which the workstation comprises a collection of multimedia components which are physically discrete but are logically bound together in software. An alternative approach is to adopt a more integrated architecture in which the required functionality is provided by a single system. Both approaches have their advantages and disadvantages (which are partly influenced by the state of technology), and a combination of the two designs may prove to represent the optimal solution for the current 'state of the art'.

Given the availability of suitable multimedia workstations, a distributed systems architecture is necessary in order to be able to construct the desired multimedia environ-
Future Work

The Advanced Network Systems Architecture (ANSA) [ANSA 89] is an architecture for building distributed systems in a heterogeneous, multi-vendor, multi-domain environment. An interesting area of research could be to consider whether the orchestration services (synchronisation, floor control, etc) [Adams 87] mentioned in Chapter 2 could be accommodated within the ANSA architectural framework. Orchestration is used here to describe the services that add value to the underlying network of ad hoc multimedia components in order to provide an integrated multimedia environment, and this would seem to be a worthy addition to the basic ANSA architecture.

ANSA defines a mechanism for the binding of different parts of a distributed application based on the use of a ‘trader’. A trader performs the typed matching of interface exports to import requests based on interface name and property values. Indirection via the trader in this way not only supports late binding between application modules but also offers opportunities for the functional matching of interfaces which is the basis for coping with heterogeneity. A possible area of research would be to extend these ideas to allow communication between functionally equivalent multimedia components, perhaps by employing the services of a translating component.

It is becoming increasingly popular to take an object-oriented approach to programming in a distributed environment (for example, [Shrivastava 88] in the design of a reliable distributed computing system, [Aguilar 85] in the design of a military command and control system). The main reason for this is that the object model partitions and encapsulates state, and it is therefore well suited to representing a network of components. ANSA defines an object-oriented computational model based on an interface definition language (IDL) and a distributed processing language (DPL). A set of operations supported by an object is called an interface; an object might offer several interfaces each intended for a different class of user. DPL uses objects as the unit of distribution and interfaces as the unit of structuring. A possible area of research is to investigate extensions to IDL/DPL that might be required in order to be able to handle multimedia objects. This includes the requirement to provide synchronisation between logically related isochronous streams such as voice and video. Furthermore, there might be opportunities for developing the ‘domain of control’ model described in Chapter 6 by specifying a suitable ‘UserName’ property variable during a binding request in order to select the appropriate interface to a component.

A final important area for research is the design of the human interface to the multimedia workstation. For example, how should video be presented and manipulated by the user? How should conference voice be handled - can selective mixing of sources combined with stereo/quadrephonic techniques give the user the illusion that his conference colleagues are in different parts of the room, and, if so, is this desirable? The human interface is a topic that is often overlooked by systems designers, and yet it is crucial to the successful uptake of the system. Collaboration between computer scientists, engineers and human factors specialists may well prove to be fruitful.
Appendix A

The Cambridge Ring Framestore Network
Interface Specification

A.1 Introduction

This appendix defines the network interface to the Cambridge Ring framestores described in chapter 4. The interface can be divided into two parts: the control interface to allow the framestore to be controlled by a separate control entity (usually a workstation), and the data interface to provide client access (from peer framestores or from electronic blackboard software running on the workstation).

The control interface is based on the single shot protocol discussed in chapter 3. The workstation sends an SSPREQ requesting an action, and the framestore attempts to perform this action and reports the outcome (along with any other results) by returning an SSPRPLY. A control entity should access the control interface by indirection through the nameserver in the usual way.

The data interface is based on the lightweight virtual circuit discussed in chapter 3. Call set-up is by means of the usual OPEN/OPENACK exchange, but thereafter the data protocol depends on the channel type (video, still image, or electronic blackboard). Once again, the data interface is accessed by indirection through the nameserver.

A.2 Structure of Service Names

The service names that define both the interfaces have the general structure '<service>-<framestore>'. The service component is the generic service name such as 'make.channel' (control interface) or 'connect' (data interface). The framestore component is the name of the framestore that is offering the service (for example 'darius'), and it is necessary to define the particular instance of the generic service. Remember that a fully qualified Universe service name is of the form '<site>*<localname>' (chapter 3), so that a fully qualified framestore name is of the form '<site>*<localname>-<framestore>' (eg ral*make.channel-darius).
A.3 The Control Interface

A.3.1 Introduction

Fifteen services define the control interface, and their generic service names are given below:

- activate.channel
- start.channel
- stop.channel
- mod.channel
- kill.channel
- configure
- status
- reset
- grab.frame
- paint
- colour.bars
- toggle.frame
- codeid
- print.image

Each of these services is described below, together with the corresponding access protocol.

A.3.2 Interface Definition

A.3.2.1 make.channel

This is the control service that is used to establish channels between framestores. The framestore receiving a call to this service is destined to become the transmitter. The SSPREQ is decoded to construct the channel descriptor for this new connection. The transmitter then sends an OPEN block to the connect service (see section A.4) on the machine that is destined to become the receiver. The receiver sets up an identical copy of the channel descriptor, and returns an OPENACK containing a newly allocated channel ID. The transmitter then returns an SSPRPLY containing the channel ID supplied by the receiver and the additional channel ID allocated by the transmitter. The channel is now established (although not yet necessarily active) and ready for data transmission. Note that the OPEN/OPENACK exchange is nested within the SSP exchange.

The structure of the SSPREQ blocks for the three types of data channel that are supported is shown in Figure A.1 (a, b & c). Most of the parameters relate to the size and position of the data that is to be transmitted and the data that is to be displayed. As such, they define a linear mapping between transmitter and receiver. Other
Figure A.1a: SSPREQ Format for the *make.channel* Service for a Video Channel
Parameters prefixed by "tx" are transmitter parameters; those prefixed by "rx" are receiver parameters.
Numbers represent byte offsets.

Figure A.1b: SSPREQ Format for the *make.channel* Service for a Still Image Channel
Parameters prefixed by "tx" are transmitter parameters; those prefixed by "rx" are receiver parameters.
Numbers represent byte offsets.
parameters include the names of both the transmitter and receiver (of the form `<site>`*<machine>*) to enable a control entity to discover the name of the peer machine, and the initial channel status at call set-up time.

The SSPRPLY will indicate the outcome of the entire operation, including the success or otherwise of the virtual circuit set-up request.

A.3.2.2 activate.channel

When a channel is first set up, its status may be defined to be either active or dormant. A transmitter will only transmit data on an active channel. This mechanism is useful for temporarily suspending transmission while a channel is being modified. A channel may explicitly be made active by making a call to the `activate.channel` service.

The structure of the two types of SSPREQ is shown in Figure A.2. The SSPRPLY will indicate a command error if the specified channel does not exist.
A.3 The Control Interface

Figure A.2: Structure of the SSPREQs to the `activate.channel` service
The simplest block structure is shown on the left in which the channel ID specifies the channel to be activated. A more complex structure is shown on the right in which a class of channels specified by the 'channel type' parameter may be activated simultaneously. Numbers represent byte offsets.

A.3.2.3 `start.channel`

A call to this service starts transmission beginning with the specified channel. A side effect is that transmission starts (or resumes) on all other active channels of the same type (eg all video channels).

The SSPREQ simply contains the channel ID of the channel to be started. The SSPRPLOY will indicate a command error if the specified channel does not exist.

A.3.2.4 `stop.channel`

A call to this service changes the state of the specified channel to `dormant`. This prevents further transmission on that channel.

The structure of the two types of SSPREQ to this service is shown in Figure A.3. As before, the SSPRPLOY will indicate a command error for an invalid channel ID.

Figure A.3: Structure of the SSPREQs to the `stop.channel` service
The simplest block structure is shown on the left in which the channel ID specifies the channel to be stopped. A more complex structure is shown on the right in which a class of channels specified by the 'channel type' parameter may be stopped simultaneously. Numbers represent byte offsets.
### The Control Interface

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SSPREQ Header</td>
<td>58</td>
<td>rx pixels per line</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>58</td>
<td>rx lines per frame</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>channel ID</td>
<td>60</td>
<td>rx pixel pitch</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>channel type &quot;video&quot;</td>
<td>62</td>
<td>rx line pitch</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>tx pixels per line</td>
<td>64</td>
<td>rx first pixel</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>tx lines per frame</td>
<td>66</td>
<td>rx first line</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>tx pixel pitch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>tx line pitch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>tx first pixel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>tx first line</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>tx delaytime</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure A.4a:** SSPREQ Format for the mod.channel Service for a Video Channel

Parameters prefixed by "tx" are transmitter parameters; those prefixed by "rx" are receiver parameters.
Numbers represent byte offsets.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SSPREQ Header</td>
<td>58</td>
<td>rx pixels per line</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>58</td>
<td>rx lines per frame</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>channel ID</td>
<td>60</td>
<td>rx pixel pitch</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>channel type &quot;still&quot;</td>
<td>62</td>
<td>rx line pitch</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>tx pixels per line</td>
<td>64</td>
<td>rx first pixel</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>tx lines per frame</td>
<td>66</td>
<td>rx first line</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>tx pixel pitch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>tx line pitch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>tx first pixel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>tx first line</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure A.4b:** SSPREQ Format for the mod.channel Service for a Still Image Channel

Parameters prefixed by "tx" are transmitter parameters; those prefixed by "rx" are receiver parameters.
Numbers represent byte offsets.
A.3 The Control Interface

A.3.2.5 mod.channel

A call to this service allows an existing channel to be modified. The new channel descriptor parameters are supplied as arguments to the SSPREQ. Note that the control entity must call this service on both the machines handling the channel to ensure that the channel descriptors remain consistent. In practice, a call to this service must be preceded by a call to the stop.channel service on the transmitter to (temporarily) suspend transmission until after the modification. A call to activate.channel will then resume transmission with the new channel parameters.

The format of the mod.channel SSPREQs is shown in Figure A.4 (a & b). The SSPRP-LY will indicate a command error if the specified channel ID is not valid.

A.3.2.6 kill.channel

A channel descriptor may be deleted by calling this service. The channel ID is supplied as the only argument to the SSPREQ. As before, the SSPRP-LY will indicate a command error for an invalid ID.

A.3.2.7 configure

A call to this service allows certain default framestore parameters to be altered. The parameters supplied in the SSPREQ are:

| 0 | SSPREQ Header |
| 2 | verbose switch |
| 4 | reverse lookup switch |
| 6 | transmitter timeout keyword |
| 8 | transmitter timeout value |
| 10 | transmitter retry keyword |
| 12 | transmitter retry value |
| 14 | end of block (EOB) |

Figure A.5: Structure of the SSPREQ to the configure service

Not all of these parameters need to be supplied, but the parameter list must be terminated by an EOB marker. Numbers represent byte offsets.

- Verbose switch. By default, a framestore will print debugging information on the screen of an attached terminal (the terminal is connected to the RS-232 port on the 86/30 processor board). This information can be suppressed (thus improving performance) by turning off the verbose switch.

- Reverse lookup switch. By default, a framestore will print the station address of every incoming block as part of the debugging information supplied in verbose mode. However, by setting this switch, the store will perform reverse lookups on incoming blocks and print the originating station name instead of the address.
A.3 The Control Interface

- **Transmitter timeout.** This parameter is the time period for which the VMI-1 will retry block header minipacket transmissions. The default value is one second, although other values may be specified by supplying the 'transmitter timeout' keyword followed by the new value.

- **Block retries.** This parameter is the number of block retries that the framestore will make in the event of a block transmission failure. The default value is 50, although other values may be specified by supplying the 'transmitter retry' keyword followed by the new value.

The format of the SSPREQ to this service is shown in Figure A.5.

**A.3.2.8 status**

This call provides the mechanism for a control entity to recover the state of a framestore. In particular, it returns the list of channel descriptors containing all the channel IDs that have been allocated by that store, together with the various parameters relating to each channel. This information can be displayed on the debugging terminal, returned to the caller as part of the SSPRPLY, or both. It is by decoding this list that workstation software is able to generate a window corresponding to each channel.

The SSPREQ contains a single code specifying a local dump (to an attached terminal only), a remote dump (to the caller as part of the SSPRPLY), or a complete dump (to both).

The format of the SSPRPLY is given in Figure A.6. It contains the list of channel descriptors maintained by the framestore. The structure of the three types of channel descriptors is shown in Figure A.7 (a, b & c).

**A.3.2.9 reset**

A call to this service resets the framestore to its initial state. All memory is cleared and hardware is re-initialised. There are no arguments to the SSPREQ.
The Control Interface

<table>
<thead>
<tr>
<th>Channel ID</th>
<th>Receiver Station Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Receiver Port Number</td>
</tr>
<tr>
<td>4</td>
<td>Pointer to Reception Buffer 1</td>
</tr>
<tr>
<td>6</td>
<td>Pointer to Reception Buffer 2</td>
</tr>
<tr>
<td>8</td>
<td>Current Buffer Pointer</td>
</tr>
<tr>
<td>10</td>
<td>Receiver Station Address</td>
</tr>
<tr>
<td>12</td>
<td>Receiver Port Number</td>
</tr>
<tr>
<td>14</td>
<td>Pointer to Reception Buffer 3</td>
</tr>
<tr>
<td>16</td>
<td>Pointer to Reception Buffer 4</td>
</tr>
<tr>
<td>18</td>
<td>Current Buffer Pointer</td>
</tr>
<tr>
<td>20</td>
<td>Receiver Station Address</td>
</tr>
<tr>
<td>22</td>
<td>Receiver Port Number</td>
</tr>
<tr>
<td>24</td>
<td>Pointer to Reception Buffer 5</td>
</tr>
<tr>
<td>26</td>
<td>Pointer to Reception Buffer 6</td>
</tr>
<tr>
<td>28</td>
<td>Current Buffer Pointer</td>
</tr>
<tr>
<td>30</td>
<td>Receiver Station Address</td>
</tr>
<tr>
<td>32</td>
<td>Receiver Port Number</td>
</tr>
<tr>
<td>34</td>
<td>Pointer to Reception Buffer 7</td>
</tr>
<tr>
<td>36</td>
<td>Pointer to Reception Buffer 8</td>
</tr>
<tr>
<td>38</td>
<td>Current Buffer Pointer</td>
</tr>
<tr>
<td>40</td>
<td>Receiver Station Address</td>
</tr>
<tr>
<td>42</td>
<td>Receiver Port Number</td>
</tr>
<tr>
<td>44</td>
<td>Pointer to Reception Buffer 9</td>
</tr>
<tr>
<td>46</td>
<td>Pointer to Reception Buffer 10</td>
</tr>
<tr>
<td>48</td>
<td>Current Buffer Pointer</td>
</tr>
</tbody>
</table>

Transmitter Name

<table>
<thead>
<tr>
<th>Transmitter Name</th>
<th>Receiver Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmitter Port Number</td>
<td>Receiver Port Number</td>
</tr>
<tr>
<td>Transmitter Station Address</td>
<td>Receiver Station Address</td>
</tr>
<tr>
<td>Transmitter Channel ID</td>
<td>Receiver Channel ID</td>
</tr>
<tr>
<td>Transmitter Line Pitch</td>
<td>Receiver Line Pitch</td>
</tr>
<tr>
<td>Transmitter First Pixel</td>
<td>Receiver First Pixel</td>
</tr>
<tr>
<td>Transmitter First Line</td>
<td>Receiver First Line</td>
</tr>
<tr>
<td>Transmitter Pixel Pitch</td>
<td>Receiver Pixel Pitch</td>
</tr>
<tr>
<td>Transmitter Pixels Per Line</td>
<td>Receiver Pixels Per Line</td>
</tr>
<tr>
<td>Transmitter Lines Per Frame</td>
<td>Receiver Lines Per Frame</td>
</tr>
</tbody>
</table>

Figure A.7a: The Video Channel Descriptor

Parameters prefixed by "tx" are transmitter parameters; those prefixed by "rx" are receiver parameters. Numbers represent byte offsets.

Appendix A - 172 - CR Framestore Interface
<table>
<thead>
<tr>
<th>Channel ID</th>
<th>Channel ID</th>
<th>Channel ID</th>
<th>Channel ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>16</td>
<td>18</td>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td>24</td>
<td>26</td>
<td>28</td>
<td>30</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transmitter Station Address</th>
<th>Transmitter Port Number</th>
<th>Channel Status</th>
<th>Channel Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>52</td>
<td>54</td>
<td>56</td>
</tr>
<tr>
<td>58</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Receiver Station Address</th>
<th>Receiver Port Number</th>
<th>Pointer to Reception Buffer 1</th>
<th>Pointer to Reception Buffer 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>74</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>76</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>78</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>82</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Receiver Name</th>
<th>Receiver Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>98</td>
<td></td>
</tr>
</tbody>
</table>

**Figure A.7b: The Still Image Channel Descriptor**

Parameters prefixed by "tx" are transmitter parameters; those prefixed by "rx" are receiver parameters. Numbers represent byte offsets.
<table>
<thead>
<tr>
<th>Channel ID</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>18</th>
<th>20</th>
<th>22</th>
<th>24</th>
<th>26</th>
<th>28</th>
<th>30</th>
<th>32</th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmitter channel ID</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>13</td>
<td>15</td>
<td>17</td>
<td>19</td>
<td>21</td>
<td>23</td>
<td>25</td>
<td>27</td>
<td>29</td>
<td>31</td>
<td>33</td>
<td>49</td>
</tr>
<tr>
<td>Receiver station address</td>
<td>50</td>
<td>52</td>
<td>54</td>
<td>56</td>
<td>58</td>
<td>60</td>
<td>62</td>
<td>64</td>
<td>66</td>
<td>68</td>
<td>70</td>
<td>72</td>
<td>74</td>
<td>76</td>
<td>78</td>
<td>80</td>
<td>82</td>
<td>98</td>
</tr>
<tr>
<td>Receiver port number</td>
<td>51</td>
<td>53</td>
<td>55</td>
<td>57</td>
<td>59</td>
<td>61</td>
<td>63</td>
<td>65</td>
<td>67</td>
<td>69</td>
<td>71</td>
<td>73</td>
<td>75</td>
<td>77</td>
<td>79</td>
<td>81</td>
<td>83</td>
<td>99</td>
</tr>
<tr>
<td>Pointer to reception buffer 1</td>
<td>52</td>
<td>54</td>
<td>56</td>
<td>58</td>
<td>60</td>
<td>62</td>
<td>64</td>
<td>66</td>
<td>68</td>
<td>70</td>
<td>72</td>
<td>74</td>
<td>76</td>
<td>78</td>
<td>80</td>
<td>82</td>
<td>84</td>
<td>99</td>
</tr>
<tr>
<td>Pointer to reception buffer 2</td>
<td>53</td>
<td>55</td>
<td>57</td>
<td>59</td>
<td>61</td>
<td>63</td>
<td>65</td>
<td>67</td>
<td>69</td>
<td>71</td>
<td>73</td>
<td>75</td>
<td>77</td>
<td>79</td>
<td>81</td>
<td>83</td>
<td>85</td>
<td>99</td>
</tr>
<tr>
<td>Current buffer pointer</td>
<td>54</td>
<td>56</td>
<td>58</td>
<td>60</td>
<td>62</td>
<td>64</td>
<td>66</td>
<td>68</td>
<td>70</td>
<td>72</td>
<td>74</td>
<td>76</td>
<td>78</td>
<td>80</td>
<td>82</td>
<td>84</td>
<td>86</td>
<td>99</td>
</tr>
</tbody>
</table>

**Figure A.7c: The Blackboard Channel Descriptor**

Parameters prefixed by "tx" are transmitter parameters; those prefixed by "rx" are receiver parameters.

Numbers represent byte offsets.
A.3.2.10 grab

A framestore can be instructed to grab a frame by calling this service. This is intended for testing purposes. There are no arguments to the SSPREQ.

A.3.2.11 paint

A call to this service allows the specified rectangle to be painted (flood filled) with the specified colour. Painting an area with the background colour (typically white) clears that area of the screen. The format of the SSPREQ is given in Figure A.8.

A.3.2.12 colour.bars

A framestore may be instructed to display internally generated colour bars by calling this service. This is useful for testing the video circuitry hardware. There are no arguments to the SSPREQ.

A.3.2.13 toggle.frame

A call to this service causes the display memory planes to be interchanged. Two complete images may be saved in memory at any one time, and this call allows each to be displayed on the screen alternately. There are no arguments to the SSPREQ.

A.3.2.14 codeid

This is a standard Universe service that returns a version number and version string to identify the type of the server and the version of the software that it is running. There are no arguments to the SSPREQ. The structure of the SSPRPLY is defined in [Adams 84].

A.3.2.15 image.print

A call to this service instructs the framestore to transmit a specified area of the screen to a named (PostScript) print service. The SSPREQ defines the name of the print service to which the framestore must connect (via the usual OPEN/OPENACK exchange), and various parameters that define the area to be printed. The SSPREQ also
The Data Interface

The data interface uses the OPEN/OPENACK protocol for virtual connection set-up between machines for the exchange of data. The framestore offers a single connect service for client access. The OPEN block that is sent to this service defines the type of channel that is to be established (video, still image, or electronic blackboard).

The structure of the three OPEN blocks that can be sent to the connect service are defined in Figure A.10 (a, b & c). In each case, the channel ID is returned to the caller as the only result in the OPENACK.

Figure A.9: Structure of the SSPREQ to the image.print service

Includes a PostScript prologue destined for the printer; this is not decoded by the framestore but is simply passed on to the printer in the OPEN request. The structure of the SSPREQ is shown in Figure A.9.

A.4 The Data Interface
Appendix A - 177 -

The Data Interface

Figure A.10a: OPEN Format for the connect Service for a Video Channel

Parameters prefixed by "tx" are transmitter parameters; those prefixed by "rx" are receiver parameters.
Numbers represent byte offsets.

Figure A.10b: OPEN Format for the connect Service for a Still Image Channel

Parameters prefixed by "tx" are transmitter parameters; those prefixed by "rx" are receiver parameters.
Numbers represent byte offsets.
The Data Interface

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OPEN Header</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>rx x origin</td>
</tr>
<tr>
<td>102</td>
<td>rx y origin</td>
</tr>
<tr>
<td>104</td>
<td>rx x length</td>
</tr>
<tr>
<td>106</td>
<td>rx y length</td>
</tr>
<tr>
<td>200</td>
<td>channel type “blackboard”</td>
</tr>
<tr>
<td>202</td>
<td>tx x origin</td>
</tr>
<tr>
<td>204</td>
<td>tx y origin</td>
</tr>
<tr>
<td>206</td>
<td>tx x length</td>
</tr>
<tr>
<td>208</td>
<td>tx y length</td>
</tr>
<tr>
<td>216</td>
<td>Transmitter Name</td>
</tr>
<tr>
<td>232</td>
<td></td>
</tr>
<tr>
<td>290</td>
<td>tx channel ID</td>
</tr>
<tr>
<td>292</td>
<td>channel status</td>
</tr>
</tbody>
</table>

Figure A.10c: OPEN Format for the connect Service for a Blackboard Channel
Parameters prefixed by "tx" are transmitter parameters; those prefixed by "rx" are receiver parameters.
Numbers represent byte offsets.
Appendix B
Scanner RPC Stub Definition

B.1 Introduction

This appendix defines the RPC interface to the CFR scanner system (chapter 5) in terms of the RPC stub definition language described in [Wilson 87b]. In all calls, \(rc\) is the return code passed back to the client, and \(session.id\) is the session identifier allocated by the server at association set-up time.

B.2 Definition

PROCEDURE getlock \((rc, session.id)\) PROTOCOL exactlyonce

\[
\begin{align*}
\text{INT} & \quad rc & \text{OUT} \\
\text{INT} & \quad session.id & \text{IN} \\
\end{align*}
\]

(*
Get a lock on the scanner for this session. This prevents other clients from accessing the scanner until the lock is explicitly freed.
*)

END getlock

PROCEDURE freelock \((rc, session.id)\) PROTOCOL exactlyonce

\[
\begin{align*}
\text{INT} & \quad rc & \text{OUT} \\
\text{INT} & \quad session.id & \text{IN} \\
\end{align*}
\]

(*
Free the lock on the scanner for this session. This allows other clients to access the scanner.
*)

END freelock

PROCEDURE test \((rc, session.id)\) PROTOCOL exactlyonce

\[
\begin{align*}
\text{INT} & \quad rc & \text{OUT} \\
\text{INT} & \quad session.id & \text{IN} \\
\end{align*}
\]

(*
Request a scanner self test. This checks out the scanner hardware.
*)

END test
PROCEDURE inquire (rc, session.id, inquireblk) PROTOCOL exactlyonce
  INT    rc                     OUT
  INT    session.id            IN
  CHAR   inquireblk ARRAY LENGTH = len.inquireblk   OUT
(*)
Request information on the scanner type and hardware capability. This includes
such information as the resolution settings available, the grain sizes available (for
halftone operation), the number of contrast settings, etc.
)
END inquire

PROCEDURE setframe (rc, session.id, frameblk, len.frameblk) PROTOCOL exactlyonce
  INT    rc                     OUT
  INT    session.id            IN
  CHAR   frameblk ARRAY LENGTH = maxlen.frameblk   IN
  INT    len.frameblk          IN
(*)
Set up frame size. This call defines the area of the image that is to be
scanned, together with the scanning mode (lineart or halftone).
)
END setframe

PROCEDURE setmode (rc, session.id, modeblk) PROTOCOL exactlyonce
  INT    rc                     OUT
  INT    session.id            IN
  CHAR   modeblk ARRAY LENGTH = len.modeblk   IN
(*)
Set up scanner mode parameters. These include resolution, brightness, contrast
setting, etc.
)
END setmode
PROCEDURE sensemode (rc, session.id, modeblk) PROTOCOL exactlyonce
  INT rc OUT
  INT session.id IN
  CHAR modeblk ARRAY LENGTH = len.modeblk OUT
  (*
  Read scanner mode parameters. These include resolution, brightness, contrast setting, etc.
  *)
END sensemode

PROCEDURE command (rc, session.id, action) PROTOCOL exactlyonce
  INT rc OUT
  INT session.id IN
  INT action IN
  (*
  Start or stop the scanner depending on the value of action.
  *)
END command

PROCEDURE readinit (rc, session.id, bytesperline, linecount, preread, linesperchunk) PROTOCOL exactlyonce
  INT rc OUT
  INT session.id IN
  INT bytesperline IN
  INT linecount IN
  BOOL preread IN
  INT linesperchunk IN
  (*
  Initialise the scanner server in preparation for a read operation. If the preread switch is set, then a synchronous read is being requested. In this situation, the combination of linesperchunk, linecount and bytesperline specify the size of the cache (chunk) that is being requested. If sufficient memory is not available, the call is failed, otherwise the first chunk of data is read immediately. If this a synchronous read, only the linecount parameter is relevant since the bytesperline parameter in included in every read request anyway.
  *)
END readinit
PROCEDURE read (rc, session.id, readblk, linesthiscall, bytesperline, size)

PROTOCOL exactlyonce

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>rc</td>
</tr>
<tr>
<td>INT</td>
<td>session.id</td>
</tr>
<tr>
<td>CHAR</td>
<td>readblk ARRAY LENGTH = size MAXLENGTH = maxlen.readblk</td>
</tr>
<tr>
<td>INT</td>
<td>linesthiscall</td>
</tr>
<tr>
<td>INT</td>
<td>bytesperline</td>
</tr>
<tr>
<td>INT</td>
<td>size</td>
</tr>
</tbody>
</table>

(*
Read a block of data from the scanner server into readblk. If this a synchronous read operation, the block has to be read from the scanner itself. If this an asynchronous read operation, the block can be read directly from the cache. The combination of linesthiscall and bytesperline specifies the amount of data to be read.
*)

END read

PROCEDURE readuninit (rc, session.id)

PROTOCOL exactlyonce

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>rc</td>
</tr>
<tr>
<td>INT</td>
<td>session.id</td>
</tr>
</tbody>
</table>

(*
De-allocate the cache after an asynchronous read operation.
*)

END readuninit

PROCEDURE alive (rc, session.id, message)

PROTOCOL exactlyonce

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>rc</td>
</tr>
<tr>
<td>INT</td>
<td>session.id</td>
</tr>
<tr>
<td>STRING</td>
<td>message ARRAY LENGTH = scanner.maxmessagelen</td>
</tr>
</tbody>
</table>

(*
Refresh the scanner server session. The server may pass back a message to the caller using the message string (for example, to notify clients that it is about to terminate.
*)

END alive
Appendix C
Grabber RPC Stub Definition

C.1 Introduction

This appendix defines the RPC interface to the workstation video grabber system (chapter 5) in terms of the RPC stub definition language described in [Wilson 87b]. In all calls, rc is the return code passed back to the client, and peervcb is the handle on the connection that is allocated by the peer at association set-up time.

C.2 Definition

PROCEDURE transmit (rc, peervcb, linecount, linesperblock, videoblock)

PROTOCOL exactlyonce

INT rc OUT
INT peervcb IN
INT linecount IN
INT linesperblock IN
CHAR videoblock ARRAY LENGTH = bytes.videoblock IN

(*
Transmit a block of video data contained in videoblock. The number of lines is specified by linesperblock, and the start line is specified by linecount.
*)

END transmit

PROCEDURE showpic (rc, peervcb) PROTOCOL exactlyonce

INT rc OUT
INT peervcb IN

(*
Force the receiver to display the contents of its frame buffer. This call is made after a sufficient number of transmit calls have been made to transmit an entire frame.
*)

END showpic
PROCEDURE terminate (rc, peervcb) PROTOCOL exactlyonce

INT rc OUT
INT peervcb IN

(*
Attempt to terminate the peer grabber program. This call will fail if the peer is a transmitter that is part way through transmitting a frame. In this situation, the receiver should retry until successful.
*)

END terminate
Appendix D
Office Manager RPC Stub Definition

D.1 Introduction

This appendix defines the RPC interface to the office manager (chapter 6) in terms of the RPC stub definition language described in [Wilson 87b]. In all calls, \( rc \) is the return code passed back to the caller, \( sid \) is the session identifier allocated to a client by the office manager at association set-up time, and \( pid \) is the peer session identifier allocated to a peer office manager by the office manager at association set-up time.

D.2 Definition

PROCEDURE alive (rc, sid) PROTOCOL exactlyonce

\[
\begin{align*}
\text{INT} & \quad \text{rc} & \quad \text{OUT} \\
\text{INT} & \quad \text{sid} & \quad \text{IN} \\
( & \quad \text{\text{"\text{Refresh the client office manager session. This is essential to prevent the manager from timing out the client and removing the user's dynamic services.\text{"}\text{"}})} \\
) & \quad \text{END alive}
\end{align*}
\]

PROCEDURE lookup (rc, sid, genericservice, user, homedomain, listid, listlen)

PROTOCOL exactlyonce

\[
\begin{align*}
\text{INT} & \quad \text{rc} & \quad \text{OUT} \\
\text{INT} & \quad \text{sid} & \quad \text{IN} \\
\text{STRING} & \quad \text{genericservice MAXLENGTH = sec.maxnamelength} & \quad \text{IN} \\
\text{STRING} & \quad \text{user MAXLENGTH = sec.maxnamelength} & \quad \text{IN} \\
\text{STRING} & \quad \text{homedomain MAXLENGTH = sec.maxnamelength} & \quad \text{IN} \\
\text{INT} & \quad \text{listid} & \quad \text{OUT} \\
\text{INT} & \quad \text{listlen} & \quad \text{OUT} \\
( & \quad \text{\text{"\text{Construct a list of service instances for the specified \textit{genericservice} belonging to \textit{user} at \textit{homedomain}. The list length and list identifier are returned as \textit{listlen} and \textit{listid} respectively. The names on the list must then be retrieved by making subsequent calls to \textit{nextstring}.\text{"}\text{"}})} \\
) & \quad \text{END lookup}
\end{align*}
\]
PROCEDURE who (rc, sid, domain, listid, listlen) PROTOCOL exactlyonce
  INT rc OUT
  INT sid IN
  STRING domain MAXLENGTH = sec.maxnamelength IN
  INT listid OUT
  INT listlen OUT

(*
Construct a list of users that are registered with the office manager at the specified domain. The list length and list identifier are returned as before. *)
END who

PROCEDURE nextstring (rc, sid, listid, name) PROTOCOL exactlyonce
  INT rc OUT
  INT sid IN
  INT listid IN
  STRING name MAXLENGTH = sec.maxnamelength OUT

(*
Given a list identifier, return the current name on the corresponding list. The office manager maintains the state of each list in terms of the current position of the pointer. When all the strings on the list have been read, the list is automatically deleted *)
END nextstring

PROCEDURE peer.lookup (rc, pid, genericservice, user, homedomain, listid, listlen)
  PROTOCOL exactlyonce
  INT rc OUT
  INT pid IN
  STRING genericservice MAXLENGTH = sec.maxnamelength IN
  STRING user MAXLENGTH = sec.maxnamelength IN
  STRING homedomain MAXLENGTH = sec.maxnamelength IN
  INT listid OUT
  INT listlen OUT

(*
Construct a list of service instances for the specified genericservice belonging to user at homedomain. The list length and list identifier are returned as listlen and listid respectively. The names on the list must then be retrieved by making subsequent calls to peer.nextstring. *)
END peer.lookup
PROCEDURE peer.who (rc, pid, domain, listid, listlen) PROTOCOL exactlyonce

INT rc OUT
INT pid IN
STRING domain MAXLENGTH = sec.maxnamelength IN
INT listid OUT
INT listlen OUT

(*
Construct a list of users that are registered with the office manager at the specified domain. The list length and list identifier are returned, allowing the names on the list to be retrieved by making subsequent calls to peer.nextstring.
*)

END peer.who

PROCEDURE peer.nextstring (rc, pid, listid, name) PROTOCOL exactlyonce

INT rc OUT
INT pid IN
INT listid IN
STRING name MAXLENGTH = sec.maxnamelength OUT

(*
Given a list identifier listid, return the current name on the corresponding list. The office manager maintains the state of each list in terms of the current position of the pointer. When all the names on the list have been read, the list is automatically deleted.
*)

END peer.nextstring
References

[Acorn 87a] "CFR Station Chip Datasheet, CFR Repeater Chip Datasheet"
Acorn Computers Ltd, Cambridge, UK (August 1987)


[Adams 84] "Universe Network Protocol Architecture"
CJ Adams et al
Universe Report Number 2 (August 1984)

[Adams 85] "Voice Experiments in the Universe Network"
CJ Adams, S Ades
Proc IEEE ICC 85, Chicago (1985)

[Adams 87] "OUTLINE - A General Multimedia Structure"
CJ Adams
Project Unison working paper UR030 (March 1987)

[Ades 87] "An Architecture for Integrated Services on the Local Area Network"
S Ades

Adobe Systems Inc (August 1987)

[Aguilar 85] "CCWS: A Computer-based, Multimedia Information System"
L Aguilar et al
IEEE Computer (October 1985)

[ANSA 89] "The ANSA Reference Manual"

[Burren 89a] "Project UNIVERSE - An Experiment in High-speed Computer Networks"
JW Burren, CS Cooper
[Burren 89b] "Flexible Aggregation of Channel Bandwidth in Primary Rate ISDN"
JW Burren
Proc ACM Sigcomm, Austin, Texas (September 1989)

[CCITT 85] "I-Series Recommendations"
International Telecommunications Union
CCITT VIIIth Plenary Assembly Fascicle III.5 (1985)

[Chambers 86] "Introduction to the ARM Related Chip Set"
AM Chambers
Project Unison working paper UA006 (October 1986)

[Chapman 89] "Access to Services in Distributed Systems"
MD Chapman

[Chin 89] "Statistics of Video Signals for Viewphone-type Pictures"
HS Chin et al

[Clark 86] "Unison - Communications Research for Office Applications"
PF Clark et al

[Cooper 84] "Infrastructure"
CS Cooper et al
Project Universe Report Number 19 (October 1984)

[Cooper 89] "S-less Connection Setup in Unison"
CS Cooper
Project Unison working paper UR065 (August 1989)

[Falconer 85] "Orwell: A Protocol for an ISLN"
RM Falconer, JL Adams
BT Technology Journal Vol 3, No 4 (1985)

[Griffiths 84] "Images"
JWR Griffiths et al
[Hamilton 84] "A Remote Procedure Call System"
KG Hamilton

[Hardman 89] "Two-party Wideband Speech Conference System for use on the Unison Network"
VJ Hardman
Project Unison working paper, Loughborough University of Technology (1989)

[Harita 89] "Dynamic Bandwidth Management of Primary Rate ISDN to Support ATM Access"
B Harita, IM Leslie
Proc ACM Sigcomm, Austin, Texas (September 1989)

[Hodges 89] "A Construction Set for Multimedia Applications"
ME Hodges et al
IEEE Software (January 1989)

[Hoffner 88] "A User Interface Architecture"
Y Hoffner, J Monk
Proc UK IT 88, Swansea, UK (July 1988)

[Hopper 86a] "Local Area Network Design"
A Hopper, S Temple, RC Williamson
Addison-Wesley (1986)

[Hopper 86b] "The Cambridge Fast Ring Networking System"
A Hopper, RM Needham
University of Cambridge Computing Laboratory Technical Report No 90, Cambridge, UK (June 1986)

[Hopper 88] "The Olivetti Pandora Project"
A Hopper, D Milway, I Wilson
International Workshop on Packet Video, Turin (September 1988)

[Hunter 89] "ODA: A Document Architecture for Open Systems"
R Hunter et al
Computer Communications Vol 12, No 2 (April 1989)
References

[Intel 82] "iSBC 86/14 and 86/30 Single Board Computer Hardware Reference Manual"
Intel Corporation (1982)

[Knight 82] "Portable Systems Software for Personal Computers on a Network"
BJ Knight

[Lazar 85] "MAGNET: Columbia's Integrated Network Testbed"
AA Lazar et al
IEEE JSAC Vol 3, No 6 (November 1985)

[Lazar 87] "Packetized Video on MAGNET"
AA Lazar, JS White
Optical Engineering Journal Vol XXV1 (July 87)

[Lee 86a] "Image Transmission over the Cambridge Ring"
BS Lee

[Lee 86b] "Image Station"
BS Lee
Project Unison working paper UL004 (1986)

[Leslie 83] "Extending the Local Area Network"
IM Leslie

[Logica 81a] "Interface Unit Manual, Multibus Intelligent Interface Unit, VMI-1"
Logica VTS Ltd (1981)

[Logica 81b] "Network Manual"
Logica VTS Ltd (1981)

[Lu 89] "Graphics and Video Communications over an Integrated Services Network"
G Lu
"Maintaining the Time in a Distributed System"
K Marzullo, S Owicki
ACM symposium on Principles of Distributed Computing, Quebec, Canada (August 1983)

"Unity: An RPC Mechanism"
DR McAuley
Project Unison working paper UC027 (March 1987)

"MSNL Interworking with ATM"
DR McAuley
Project Unison working paper UC039 (April 1989)

"Office Communications"
JS McConnell, C Manros
Computer Communications Vol 12, No 2 (April 1989)

"Microtek MSF-300C Intelligent Image Scanner Operation Manual"
Microtek International Inc (October 1987)

"MS-SCSI/C Adapter Operation Manual"
Microtek International Inc (June 1987)

"A Coroutine Mechanism for BCPL"
K Moody, M Richards

"Video Transfers on the Unison Network"
BJ Murphy
Project Unison working paper UL018 (October 1987)

"Multimedia Applications over a Wide Area Network"
BJ Murphy et al
Proc UK IT 88, Swansea, UK (July 88)

"Agora - An Experiment in Multimedia Message Systems"
N Naffah et al
IEEE Computer (May 1986)

"The Cambridge Distributed Computing System"
RM Needham, Herbert
Addison-Wesley (1982)
[Richards 79] "TRIPOS - A Portable Operating System for Minicomputers"
M Richards et al
Software - Practice and Experience (June 1979)

[Richards 80] "BCPL - The Language and its Compiler"
M Richards, C Whitby-Strevens
Cambridge University Press (1980)

[Siddiqui 89] "Performance Measurement Methodology for Integrated Services Networks"
M Siddiqui

[Shrimpton 87] "Multicast Services: Initial Ideas"
DH Shrimpton
Project Unison working paper UR043 (November 1987)

[Shrivastava 88] "A Technical Review of Arjuna - A System for Reliable Distributed Computing"
SK Shrivastava et al
Proc UK IT 88, Swansea, UK (July 1988)

[Swinehart 83] "Adding Voice to an Office Computer Network"
DC Swinehart et al
PROC IEEE Globecom (1983)

[Tennenhouse 86] "The UDL Protocol Specification"
DL Tennenhouse
Project Unison working paper UC022 (September 1986)

[Tennenhouse 87] "Exploiting Wideband ISDN: The Unison Exchange"
DL Tennenhouse et al
Proc IEEE Infocom (March 87)

[Tennenhouse 88] "Site Interconnection and the Exchange Architecture"
DL Tennenhouse

[Tennenhouse 89] "A Testbed for Wide Area ATM Research"
DL Tennenhouse, IM Leslie
PROC ACM Sigcomm, Austin, Texas (September 1989)

References
• References
[Wilson 87a] "The Olivetti Research RPC Package"
ID Wilson
Olivetti Research Laboratory, Cambridge, UK (October 1987)

[Wilson 87b] "Stubgen: An Automatic RPC Stub Generator"
ID Wilson
Olivetti Research Laboratory, Cambridge, UK (October 1987)

[Yamaguchi 86] "A 64 kbit/s Integrated Visual Communication System - A New Communication Medium for the ISDN"
H Yamaguchi et al
IEEE JSAC Vol 4, No 8 (November 1986)

[Yin 86] "Wideband Speech Interface for the Cambridge Ring"
MT Yin
MSc Thesis, Loughborough University of Technology, Loughborough, UK (September 1986)