PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file - Votre référence

Our file - Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

A Network Services Infrastructure for Telepresence Applications

by

BRUCE D. J. MCLEOD, B.A.Sc

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

Master of Engineering

Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada K1S 5B6
May 2, 1995

© copyright
1995 Bruce D. J. McLeod

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

THE AUTHOR HAS GRANTED AN
IRREVOCABLE NON-EXCLUSIVE
LICENCE ALLOWING THE NATIONAL
LIBRARY OF CANADA TO
REPRODUCE, LOAN, DISTRIBUTE OR
SELL COPIES OF HIS/HER THESIS BY
ANY MEANS AND IN ANY FORM OR
FORMAT, MAKING THIS THESIS
AVAILABLE TO INTERESTED
PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE
IRREVOCABLE ET NON EXCLUSIVE
PERMETTANT A LA BIBLIOTHEQUE
NATIONALE DU CANADA DE
REPRODUIRE, PRETER, DISTRIBUER
OU VENDRE DES COPIES DE SA
THESE DE QUELQUE MANIERE ET
SOUS QUELQUE FORME QUE CE SOIT
POUR METTRE DES EXEMPLAIRES DE
CETTE THESE A LA DISPOSITION DES
PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP
OF THE COPYRIGHT IN HIS/HER
THESIS. NEITHER THE THESIS NOR
SUBSTANTIAL EXTRACTS FROM IT
MAY BE PRINTED OR OTHERWISE
REPRODUCED WITHOUT HIS/HER
PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE
DU DROIT D'AUTEUR QUI PROTEGE
SA THESE. NI LA THESE NI DES
EXTRAITS SUBSTANTIELS DE CELLE-
CI NE DOIVENT ETRE IMPRIMES OU
AUTREMENT REPRODUITS SANS SON
AUTORISATION.

ISBN 0-612-03032-6

Canada

Name

Bruce McLeod

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

System & Comp. Eng.

SUBJECT TERM

0544

U.M.I.

SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture	0729
Art History	0377
Cinema	0900
Dance	0378
Fine Arts	0357
Information Science	0723
Journalism	0391
Library Science	0399
Mass Communications	0708
Music	0413
Speech Communication	0459
Theater	0465

EDUCATION

General	0515
Administration	0514
Adult and Continuing	0516
Agricultural	0517
Art	0273
Bilingual and Multicultural	0282
Business	0688
Community College	0275
Curriculum and Instruction	0727
Early Childhood	0518
Elementary	0524
Finance	0277
Guidance and Counseling	0519
Health	0680
Higher	0745
History of	0520
Home Economics	0278
Industrial	0521
Language and Literature	0279
Mathematics	0280
Music	0522
Philosophy of	0998
Physical	0523

Psychology	0525
Reading	0535
Religious	0527
Sciences	0714
Secondary	0533
Social Sciences	0534
Sociology of	0340
Special	0529
Teacher Training	0530
Technology	0710
Tests and Measurements	0288
Vocational	0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language	
General	0679
Ancient	0289
Linguistics	0290
Modern	0291
Literature	
General	0401
Classical	0294
Comparative	0295
Medieval	0297
Modern	0298
African	0316
American	0591
Asian	0305
Canadian (English)	0352
Canadian (French)	0355
English	0593
Germanic	0311
Latin American	0312
Middle Eastern	0315
Romance	0313
Slavic and East European	0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy	0422
Religion	
General	0318
Biblical Studies	0321
Clergy	0319
History of	0320
Philosophy of	0322
Theology	0469

SOCIAL SCIENCES

American Studies	0323
Anthropology	
Archaeology	0324
Cultural	0326
Physical	0327
Business Administration	
General	0310
Accounting	0272
Banking	0770
Management	0454
Marketing	0338
Canadian Studies	0385
Economics	
General	0501
Agricultural	0503
Commerce-Business	0505
Finance	0508
History	0509
Labor	0510
Theory	0511
Folklore	0358
Geography	0366
Gerontology	0351
History	
General	0578

Ancient	0579
Medieval	0581
Modern	0582
Black	032E
African	0331
Asia, Australia and Oceania	0332
Canadian	0334
European	0335
Latin American	0336
Middle Eastern	0333
United States	0337
History of Science	0585
Law	0398
Political Science	
General	0615
International Law and Relations	0616
Public Administration	0617
Recreation	0814
Social Work	0452
Sociology	
General	0626
Criminology and Penology	0627
Demography	0938
Ethnic and Racial Studies	0631
Individual and Family Studies	0628
Industrial and Labor Relations	0629
Public and Social Welfare	0630
Social Structure and Development	0700
Theory and Methods	0344
Transportation	0709
Urban and Regional Planning	0999
Women's Studies	0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture	
General	0473
Agronomy	0285
Animal Culture and Nutrition	0475
Animal Pathology	0476
Food Science and Technology	0359
Forestry and Wildlife	0478
Plant Culture	0479
Plant Pathology	0480
Plant Physiology	0817
Range Management	0777
Wood Technology	0746
Biology	
General	0306
Anatomy	0287
Biostatistics	0308
Botany	0309
Cell	0379
Ecology	0329
Entomology	0353
Genetics	0369
Limnology	0793
Microbiology	0410
Molecular	0307
Neuroscience	0317
Oceanography	0416
Physiology	0433
Radiation	0821
Veterinary Science	0778
Zoology	0472
Biophysics	
General	0786
Medical	0760

EARTH SCIENCES

Biogeochemistry	0425
Geochemistry	0996

Geodesy	0370
Geology	0372
Geophysics	0373
Hydrology	0388
Mineralogy	0411
Paleobotany	0345
Paleoecology	0426
Paleontology	0418
Paleozoology	0985
Palynology	0427
Physical Geography	0368
Physical Oceanography	0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences	0768
Health Sciences	
General	0566
Audiology	0300
Chemotherapy	0992
Dentistry	0567
Education	0350
Hospital Management	0769
Human Development	0758
Immunology	0982
Medicine and Surgery	0564
Mental Health	0347
Nursing	0569
Nutrition	0570
Obstetrics and Gynecology	0380
Occupational Health and Therapy	0354
Ophthalmology	0381
Pathology	0571
Pharmacology	0419
Pharmacy	0572
Physical Therapy	0382
Public Health	0573
Radiology	0574
Recreation	0575

Speech Pathology	0460
Toxicology	0383
Home Economics	0386

PHYSICAL SCIENCES

Pure Sciences	
Chemistry	
General	0485
Agricultural	0749
Analytical	0486
Biochemistry	0487
Inorganic	0488
Nuclear	0738
Organic	0490
Pharmaceutical	0491
Physical	0494
Polymer	0495
Radiation	0754
Mathematics	0405
Physics	
General	0605
Acoustics	0986
Astronomy and Astrophysics	0606
Atmospheric Science	0608
Atomic	0748
Electronics and Electricity	0607
Elementary Particles and High Energy	0798
Fluid and Plasma	0759
Molecular	0609
Nuclear	0610
Optics	0752
Radiation	0756
Solid State	0611
Statistics	0463
Applied Sciences	
Applied Mechanics	0346
Computer Science	0984

Engineering	
General	0537
Aerospace	0538
Agricultural	0539
Automotive	0540
Biomedical	0541
Chemical	0542
Civil	0543
Electronics and Electrical	0544
Heat and Thermodynamics	0348
Hydraulic	0545
Industrial	0546
Marine	0547
Materials Science	0794
Mechanical	0548
Metallurgy	0743
Mining	0551
Nuclear	0552
Packaging	0549
Petroleum	0765
Sanitary and Municipal	0554
System Science	0790
Geotechnology	0428
Operations Research	0796
Plastics Technology	0795
Textile Technology	0994

PSYCHOLOGY

General	0621
Behavioral	0384
Clinical	0622
Developmental	0620
Experimental	0623
Industrial	0624
Personality	0625
Physiological	0989
Psychobiology	0349
Psychometrics	0632
Social	0451

The undersigned hereby recommend to the Faculty of Graduate Studies and Research acceptance of the thesis

A Network Services Infrastructure for Telepresence Applications

submitted by Bruce D. J. McLeod, B.A.Sc., in partial fulfillment of the requirements for the degree of Master of Engineering.

Chair.

Department of Systems and
Computer Engineering

Dr. Gerald M. Karam.
Thesis Supervisor

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
May 2, 1995

Abstract

A Network Services Infrastructure (NSI) is proposed to provide a simple, flexible architecture for creating networked applications. This work was developed to satisfy the needs of such applications so they could effectively utilize network capabilities. *Telepresence* applications are used in this research as an example of this type of application. Telepresence combines computing and telecommunications technologies so as to provide a sense of community and social proximity that is typically lacking in geographically and temporally disparate groups of collaborators.

This thesis documents the motivation, requirements, design and implementation of the NSI. The models and abstractions used to derive the appropriate network services are discussed, and major architectural components of the system are described. The NSI is evaluated via the development of two telepresence applications that utilize it: (1) the Teleconference Conference Control application, which is a framework for providing room-level control over audio/video teleconferences and their associated devices; and (2), the Postcards System, which provides a sense of background awareness in a remotely distributed workgroup through the use of small visual images displayed on a user's workstation.

Acknowledgements

I would like to thank Dr. Gerald Karam, my academic advisor and thesis supervisor, for guiding me through the graduate degree process at Carleton University, and for providing a stimulating and rewarding environment for my research with the Ontario Telepresence Project.

I would like to thank my friend and counterpart in telepresence research, Gerald Boersma, for countless brain-storming sessions and much needed feedback on NSI concepts and interface issues, as well as the engineers of the Ontario Telepresence Project for their valuable assistance in building the NSI.

I would like to thank my wife Paula, for her loving support, encouragement, and faithful understanding through the challenging and difficult times over the past couple of years. I would also like to thank Wayne and Kathy for the "frequent" use of their facilities for my research.

Finally, I would like to thank NSERC as well as the Department of Systems and Computer Engineering for their generous financial support during my studies.

Contents

1	Introduction	1
1.1	Telepresence	3
1.2	Required Network Services	4
1.3	Providing a Network Services Infrastructure	6
1.4	NSI Case Studies	7
1.5	Thesis Outline	8
2	Background	10
2.1	Related Work	10
2.2	Multipoint Communications Service (MCS)	17
2.2.1	MCS Setup	17
2.2.2	Using MCS	21
2.2.3	Bridging Entities	22
2.3	The Ontario Telepresence Project	22
2.4	Integrated Interactive Intermedia Facility	25
2.5	Postcards	26
3	Telepresence Network Requirements	29
3.1	Scope	30
3.2	Network Goals	32
3.3	Service Requirements	37
3.3.1	Group Communications Services	37
3.3.2	Directory Services	38
3.3.3	Resource Management Services	40
3.4	Architectural Requirements	41
3.4.1	MCS Component	41
3.4.2	NSI Services	42
3.4.3	Multicast Facilities	42
3.4.4	Client Construction	43

3.4.5	Implementation Considerations	44
4	A Network Services Infrastructure	46
4.1	NSI Organization	46
4.2	Functionality	52
4.2.1	Group Communications Service Model	53
4.2.2	Directory Service Model	60
4.2.3	Resource Management Service Model	63
4.2.4	Interface Mechanisms	64
4.2.5	Network Services Framework	72
4.3	NSI Implementation	76
4.3.1	System Layout	77
4.3.2	LLNL MCAST	77
4.3.3	MCS	80
4.3.4	NSI Service Implementation	87
5	Case Studies	93
5.1	Telepresence Conference Control	93
5.1.1	Overview	93
5.1.2	TCC Operation	96
5.1.3	GCC Services of TCC	97
5.1.4	TCC Usage of NSI	98
5.1.5	Results of TCC Case Study	102
5.2	Postcards	103
5.2.1	Existing OTP Implementation	104
5.2.2	NSI-based Postcards	106
5.2.3	Client Design	110
5.2.4	Frame Grabber Design	111
5.2.5	Results of Postcards Case Study	112
6	Conclusions	115
6.1	Research Summary	115
6.2	Future Research	118
A	NSI Client Application Programming Interface	126
A.1	Client Library Functions	126
A.2	Client Callback Definitions	134

List of Figures

1.1	Typical Migration of Network Configurations	2
1.2	CSCW Research Typologies	5
2.1	An MCS Network	18
2.2	Internal Structure of an MCS Agent	20
2.3	Current Postcards Architecture	27
3.1	Telepresence Network Configuration	31
3.2	NSI Communications Substrate	39
4.1	Run-Time Organization of the NSI	47
4.2	NSI Resource Management for Network Devices	51
4.3	Sending Data in Sequenced Mode	56
4.4	Sending Data in Normal Mode	57
4.5	NSI Token Management	59
4.6	Master-Slave Domain Operation	61
4.7	Peer-to-Peer Domain Operation	62
4.8	Primitive Operation Sequences	67
4.9	NSI Operation Primitives	68
4.10	Attach-slave Event Sequence	69
4.11	Join-channel Event Sequence	70
4.12	Send-data Event Sequence	71
4.13	Example of a Finite State Machine	73
4.14	Architecture of NSI Components	78
4.15	Point-to-Point Connection Establishment	84
4.16	Multipoint Connection Establishment	85
4.17	Operation of NSI Services	90
5.1	TCC Architecture	99
5.2	Role of NSI in TCC Operation	101
5.3	Architecture of Original OTP Postcards System	105

5.4	Architecture of NSI-based Postcards System	107
5.5	NSI-based Postcards Information Flow	109
5.6	NSI-based Client/Frame Grabber Operation	113

List of Acronyms

A/V	Audio/Video
ABCAST	Absolutely-Ordered Broadcast
ANSI	American National Standards Institute
API	Application Programmer's Interface
ATM	Asynchronous Transfer Mode
BNR	Bell-Northern Research
BSD	Berkely Source Distribution
CBCAST	Causally-Ordered Broadcast
CODEC	COrder/DECoder
CSCW	Computer-Supported Cooperative Work
DOS	Disk Operating System
DRC	Distributed Resource Controller
FIFO	First In First Out
GCC	Generic Conference Control
GDSS	Group Decision Support System
GSS	Group Support System
GUI	Graphical User Interface
FSM	Finite State Machine
I/O	Input/Output
IBM	International Business Machines
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
IIIF	Integrated Interactive Intermedia Facility
IP	Internet Protocol
IPC	Inter-Process Communication
ISI	Information Sciences Institute
ISO	International Standards Organization
ITU	International Telecommunications Union
LAN	Local Area Network

LDC	Logical Device Controller
LLNL	Lawrence Livermore National Library
MBone	Multicast Backbone
MAN	Metropolitan Area Network
MCAST	Multicast
MCS	Multipoint Communications Service
MMCC	Multiparty Multimedia Conference Control
MS	Microsoft
MTP	Multicast Transport Protocol
NSI	Network Services Infrastructure
NVP	Network Voice Protocol
OEG	Ottawa Engineering Group
OOSE	Object-Oriented Software Engineering
OS	Operating System
OSI	Open Systems Interconnection
OTP	Ontario Telepresence Project
PBX	Private Branch Exchange
PC	Personal Computer
PDC	Physical Device Controller
PE	Protocol Engine
PIP	Picture-In-Picture
PVP	Packet Video Protocol
RPC	Remote Procedure Call
RSVP	Resource ReSerVation Protocol
RTP	Real-Time Protocol
SAP	Service Access Point
ST	Stream Protocol
SCO	Santa-Cruz Operations
TCC	Telepresence Conference Control
TCL	Tool Command Language
TCP	Transmission Control Protocol
TMS	Telepresence Media Space
UDP	Unreliable Datagram Protocol
VAT	Video-Audio Tool
VCR	Video Cassette Recorder
VMTP	Versatile Message Transaction Protocol
WAN	Wide Area Network
XTP	Xpress Transfer Protocol

Chapter 1

Introduction

The demand for reliable and readily usable network-capable applications becomes apparent as computer networks proliferate throughout virtually all institutions of modern society. Once the challenges of connecting relatively small numbers of computers onto Local Area Networks (LANs) have been overcome, the opportunity for interconnecting these isolated networks into larger Metropolitan-Area and Wide-Area Networks (MANs and WANs) inevitably presents itself [52, preface]. Organizations quickly find themselves migrating from a collection of isolated network instances to corporate-wide or enterprise networks consisting of multiple interconnected sub-networks (figure 1.1). Current network technologies such as multiple-client/single-server architectures satisfy the needs of applications executing in small homogeneous networks, but can become unwieldy and inefficient when applied in the case of large inter-networks. Correspondingly, system architectures need to migrate to multiple hierarchical or even peer-to-peer server configurations, or potentially to purely distributed application designs without the need for explicit server entities [43].

To date, no single hardware or operating system platform has become dominant enough to be considered ubiquitous. Successful applications, especially those with networking capabilities, must not only be able to run on different platforms but must

Figure 1.1: Typical Migration of Network Configurations

also be able to interact with other instances executing concurrently on heterogeneous systems, often in real time. Newer breeds of applications are concerned not only with communicating relatively low amounts of discrete program data in the traditional sense, but also with transmitting high-bandwidth information representing different media such as isochronous video and audio streams (teleconferencing systems, for example). As we build and subsequently combine these new kinds of information networks on a global scale, how can they be utilized to most effectively support the communities that they serve? Although it may take many years and much research to address all the implications of this question, we present an infrastructure for providing applications with a convenient set of "high-level" network services. We consider them "high-level" as they hide "low-level" topology, connectivity, and other network management issues that are usually secondary to the operation of the application.

1.1 Telepresence

One application domain which requires effective network services is that of *telepresence*. Telepresence is the application of computing and telecommunications technologies to support a social sense of group proximity among collaborators despite spatial and temporal separation. Telepresence applications use a variety of media (e.g. video, audio, discrete computer data), communications, and presentation techniques to achieve these goals. Researchers at Carleton University and the University of Toronto participated in the Ontario Telepresence Project (OTP) [54], a multi-site collaboration mandated to study the usability and end-user acceptance of telepresence technologies in real-world situations. Telepresence applications must be robust and amicable enough to be used in live field trials, yet must be realizable in relatively short time frames due to their "proof of concept" nature. Furthermore, these applications must also be available on three currently popular computing platforms (IBM PC compatible computers running MS-DOS and Windows, Apple Macintoshes and other Apple products running Mac-OS, as well as different flavours of the UNIX operating system such as SunOS and SCO Unix), and must be able to interact smoothly while running concurrently on these different platforms.

Telepresence may be put into the wider context of Computer-Supported Cooperative Work (CSCW) research, also commonly referred to as *groupware* or *group support systems (GSS)* [27]. Figure 1.2 presents two related typologies often used to categorize CSCW research (based on [51], and [27, page 25]). The common theme in both categorizations is the distinction made along two independent axes, that of temporal separation on one hand and geographical distance on the other. Telepresence provides a sense of social proximity by attempting to integrate the group modes, and their supporting computer-based applications, that arise from these categorizations [44], based in part on the creation of an appropriate *Telepresence Media*

Space [35].

Existing CSCW research efforts often focus on one particular quadrant of the typology, e.g. the synchronous, geographically distributed category of applications at the expense of the other more asynchronous categories [27]. Telepresence research may span many quadrants, usually within the context of a single application. For example, the OTP voice mail system, invoked whenever a group member is not available to answer the phone, automatically forwards a digitized copy of the voice message to the user's electronic mail system, which can then play the message out on the user's current computer terminal if so capable. Other examples of this principle include the video mail attendant application, as well as the Postcards system, which does not fit cleanly into either an asynchronous or synchronous mode.

1.2 Required Network Services

The need for network-capable telepresence applications has prompted investigation into the creation of a common network services framework and interface. The experience of developing existing telepresence applications has led to the identification of three key network services: a group-oriented multipoint communications facility providing the *reliable* exchange of messages (i.e., data is delivered as sent unless the sender is notified of a network error); a simple key-value lookup database or *directory service*; and a network resource management framework. These services are not only useful for independent application but also tend to be used in concert; for example, active multipoint communications facilities and/or available network resources are commonly registered in the directory allowing other application instances to easily locate them.

As much as possible, the low-level communication component of a network service infrastructure should follow existing standards, and attempt to track currently evolving standards in multipoint communications. This allows compatibility with

4-mode CSCW categorization

		Time		
		Same	Different but predictable	Different and unpredictable
Place	Same	Meeting Facilitation	Work shifts	Team rooms
	Different but predictable	Tele/Video/desktop conferencing	Electronic mail	Collaborative writing
	Different and unpredictable	Interactive multicast seminars	Computer bulletin boards	Workflow

6-mode CSCW categorization

Figure 1.2: CSCW Research Typologies

services implemented on different platforms, as well as compatibility with other systems providing similar services. Complicating this issue is the current dichotomy between computer communications and networking standardization efforts put forth by European-based organizations such as International Telecommunications Union and the International Organization for Standardization and International Electrotechnical Committee, as opposed to those put forth by the Internet community.

Network services must be delivered so as to reduce the architectural and programming complexity of client applications. Any proposed framework should allow for simple, flexible, as well as more sophisticated implementation styles. A clear, consistently defined interface is necessary, which fits the design and implementation models relevant to the available development environment. However, underlying assumptions about a particular platform's architecture and run-time environment should be avoided as much as possible in the provision of such services.

1.3 Providing a Network Services Infrastructure

The Network Services Infrastructure was defined to address the needs of networked applications. It consists of a number of independent components integrated to provide the network services necessary for telepresence applications. At the core of the NSI is the Multipoint Communications Service (MCS) originally developed at Bell-Northern Research [13], providing reliable point-to-point and multipoint communications using a standardized intra-node protocol. Other major components of the system include the Lawrence-Livermore Berkeley Laboratories MCAST software [40] which provides a multicast socket abstraction, and the NSI service logic which implements the high-level networking abstractions using MCS communication primitives.

Inherently, the NSI currently addresses the communication of discrete control information and low-bandwidth data exchange capabilities. OTP applications to

date use a baseband analog audio and video network controlled intra-site via specialized video switches, with inter-site connectivity provided by existing telecommunications technologies. The low-level network facilities necessary to support high-bandwidth, real-time video and audio streams via commonly used computing platforms is not yet widely available; this is currently an active and controversial research area [15, 61, 66]. There is added complexity in managing such networks, involving quality of service parameters, resource reservation schemes, as well as various trade-offs involving bit error rates, delay tolerance, and other transmission characteristics. Future versions of the NSI should incorporate emerging communications technologies such as Asynchronous Transfer Mode (ATM) [43] that address these issues, with minimal impact on existing applications.

The NSI was built in stages, with each component developed individually, then integrated and tested with the rest of the existing system. Major components interact using a string-based command scripting technology called Tools Command Language (TCL), a popular embeddable command language interface created by John Ousterhout at the University of California, Berkeley [50]. The use of TCL in the NSI facilitates the extension of existing services, as well as the definition of new ones. Most high-level NSI control logic is coded in TCL, including directory services and resource management facilities. The use of X.500-compliant directory services mechanisms should be considered in future versions of the NSI, and although the current naming structure is non-hierarchical, applications are free to impose their own structure (such as X.500-style hierarchy) on the name space provided by NSI directory services.

1.4 NSI Case Studies

Two experimental telepresence applications have been developed using the NSI for evaluation purposes. One involves the design of a new system that uses the NSI for

all communication services; the other takes an existing system and re-engineers it to use NSI capabilities [36].

The first experiment involves the creation of a conferencing framework for telepresence called Telepresence Conference Control (TCC) [10]. This work is an implementation of the International Telecommunications Union - Telecommunications (ITU-T) Generic Conference Control standard [4], designed to provide OTP with both a platform for constructing conferencing applications as well as with an instance of the framework for controlling "room-level" devices in an actual conference. The NSI provided network services throughout all stages of TCC development.

The second experiment takes an existing telepresence application, the Postcards system, and re-implements it using the NSI for its networking requirements. The current design consists of three components: a client for viewing subscribed users' images; a frame-grabber for obtaining these images; and a server network for distributing image and control data. Using the NSI eliminates the need for a server network component, and existing client and frame-grabber code is re-used as much as possible.

1.5 Thesis Outline

The second chapter provides the necessary background for this work. Included is a review of recent literature documenting research related to the development of the NSI. The history, motivation, and structure of MCS, the core communications technology used by the NSI, is explored. The Ontario Telepresence Project provides context for this research and is also described in this chapter, as well as the Telepresence Conference Control system and the Postcards application.

Chapter three discusses the goals and requirements for the NSI. Related work is evaluated in the context of providing communications and networking facilities to telepresence applications, and an approach for satisfying telepresence network

requirements is developed. The various interface models and service abstractions at the heart of the NSI are presented.

The design and implementation of the NSI and its components is covered in chapter four. The use of the NSI's framework and interface by an application is illustrated. The overall architecture and major components of the NSI are described, including how its components operate and interact to achieve the required service functionality.

Chapter five describes the two case studies used to evaluate the NSI. The Telepresence Conference Control system is a new application by Gerald Boersma [10] that uses the NSI for its network services. The Postcards system is an existing telepresence application that has been re-engineered to use NSI services instead of its own application-specific networking design.

The last chapter contains a summary of conclusions and major contributions made by this research. Recommendations for possible enhancements to the existing NSI implementation in many areas are made. Future directions are identified for research in the areas of networking frameworks and interfaces, as well as in the area of low-level multipoint/multicast communications.

Chapter 2

Background

2.1 Related Work

A review of current literature revealed various approaches for providing network services to CSCW-based group support systems and related applications. The research covered a diverse set of group communication and networking problems, ranging from reliable, ordered broadcast protocols to full-blown conferencing systems. Some research offered a complete solution to a particular group communication problem such as video-conferencing over a packet-switched network. Other efforts identified and addressed “low-level” multipoint communication issues such as multicast routing or spanning tree algorithms. Relatively few attempts were made to provide a framework and/or an interface encapsulating a set of useful network abstractions for creating GSS applications or groupware.

Operating System Support for Group Communications

Cheriton and Zwaenepoel [17] describe the design and implementation of process groups in the V kernel. The supporting application-level group administration and IPC mechanisms are built on top of network services that provide only a minimum of broadcast/multicast facilities. Conceived as an extension to Har-

mony/Rendezvous style Send-Receive-Reply IPC, reliable multicast communications in the context of this discussion means only that at least one reply/acknowledgement is received in response to a group message send. More reliable communications may be implemented using the GetReply primitive which explicitly waits to receive replies/acknowledgements from other group members, however group membership is not tightly controlled and is difficult to ascertain exactly. This approach seems to be more geared towards "system level" distributed algorithms; for example, it lacks user-defined membership policies, which would likely be required by Telepresence Applications such as secure video-conferencing.

Frank, Wittie, and Bernstein [26] discuss support for process groups in Micros, an operating system for the *Netcomputer*, which is a large number of hosts interconnected via a network of broadcast buses. A well-developed, ground-up discussion of the background and motivation for multipoint communication is provided, as well as a description of techniques available for implementing such facilities. As with other work there are implicit assumptions made as to the nature of the problem, in this case a packet-switched network architecture, the need for a fully distributed system architecture, and so on.

Birman, Schiper, and Stephenson [8] provide complete descriptions, motivations, justifications and performance measurements of the multicast protocols CBCAST and ABCAST that form the basis for the distributed ISIS [7] tool-kit developed at Cornell University. *CBCAST* (Causally-ordered BroadCAST), the most primitive of the two, is a reliable multicast protocol guaranteeing that two causally related messages in the system are delivered to all group members in the correct causal order. *ABCAST*, (Absolutely-ordered BroadCAST), is a totally-ordered version of CBCAST. It is built on top of CBCAST using a token management protocol to preserve an arbitrary total ordering on non-causal message sends as received by all members; i.e. message delivery order is consistent among all members.

Multicast Facilities for IP Networks

Deering [21] proposes extensions to the Internet for supporting multicast capabilities at the lower layers of the Internet Protocol Suite. Both Local Area Network and inter-network mechanisms for providing unreliable datagram-style (i.e. UDP) multicast communication facilities to end-user applications are covered. A non-deterministic "best guess" approach is taken to group communication issues such as deciding group membership, handling group address collisions, as well as routing decisions, making it difficult to determine the exact state of the multicast network at any given moment. A portion of the existing public Internet, the MBone, is now multicast-capable and has been used as the network substrate for many Internet conferencing systems such as the Video-Audio Tool or VAT [34].

Armstrong, Freier, and Marzullo [5] present an attempt by the Internet community to define a reliable multicast protocol on top of IP Multicast facilities. They describe the requirements, format, and behaviour of the Multicast Transport Protocol (MTP). MTP implicitly guarantees the necessary synchronization for a totally-ordered message delivery system. This somewhat complex protocol involves multiple layers of timers and frequent transmission of heart-beat messages. Implementations of the protocol are presently forthcoming [49].

Cheriton [16] describes the Versatile Message Transaction Protocol (VMTP), including the motivation for its creation and design as well as the full VMTP protocol specification. VMTP is a transaction-based transport protocol with multicast capabilities, as well as support for secure real-time, asynchronous streaming communication or message exchanges. The specification is explicit about being modular enough to allow subset implementations. No currently available implementation fully supports the specified functionality. VMTP provides for RPC-style client-server communications, where a particular client may communicate with an arbitrary number of servers in a multicast sense. Use of VMTP by the Telepresence environment would involve porting the existing implementation to the current SunOS release

level, VMTP kernel modifications, as well as IP Multicast capabilities. VMTP multicast services are not guaranteed to be reliable with respect to eventual delivery or failure notification. Also, there is conceptual mismatch between the "application level" services provided by VMTP and those needed for telepresence applications, due mostly to the RPC-style client-server nature of the VMTP interface.

Internet Based Conferencing Support Systems

Lashkari et al [39] provide a specification, description and motivation for the multicast communication system which underlies the Vartalaap Multimedia Conferencing system. This system uses a basic client/server architecture, composed of local servers on each host, communicating with a single global server via TCP/IP stream sockets. Connections with local host clients may also be established with sockets, or with other UNIX IPC mechanisms such as pipes and FIFOs, implying that delivery of data throughout the network is reliable; also, since a global server is used, all information is received by all clients in the same order. Scalability and robustness are open issues with this architecture, due to the reliance on a single centralized global server.

Schooler and Casner [59] document the overall architectural design of the ISI-sponsored BBN Diamond/MMConf Multimedia Conferencing system. Although targeted to run over the Internet, the system currently uses a Wide-band Satellite network (WBNet) due to the Internet's low-bandwidth and unpredictable delay characteristics. The end-station technology is similar to that of current telepresence applications, including microphones, mixers, headphones, speakers, video codecs, cameras, monitors, and the like. This architecture assumes a common network infrastructure for all data and control information, but this does not seem to impede the usefulness of the protocols above the raw network level. The system uses the MMCC (MultiMedia Conference Control) program [60] for "high-level" conference control and device management, while utilizing a combination of conventional

TCP/IP as well as the Network Voice Protocol (NVP) [19] and the Packet Video Protocol (PVP) [20] over the Internet Stream Protocol (ST) [15] for lower layer network access. Schooler [60] focuses on the MMCC program component of the system, as well as issues related to conference control, remote device management and network configuration. This report shows how different parts of the MMCC program utilize different aspects of the MMConf protocol architecture (e.g. NVP/PVP and ST versus TCP/IP). Future applicability to "personal", informal desktop conferencing applications similar to those in telepresence is explored.

Multipoint Protocol Requirements and Specifications

The XTP Protocol Definition [29] is a thorough description of the Xpress Transfer Protocol. It is referred to as a *transfer layer protocol*, encompassing the traditional OSI layers 3 (network) and 4 (transport) in one entity. It is an evolution of such Internet protocols as NetBLT [18], VMTP [16], and TCP/IP, encompassing many of the requirements and services of these protocols into one "super protocol", thus simplifying the protocol architecture design for real-time applications such as distributed multimedia. The XTP/PE implementation is a significant commercial product, and there are apparently no freely available versions of the protocol for research purposes, making it a complex and costly alternative for use in telepresence applications.

Braudes and Zabele [11] address the generic issues facing multicast protocols designed for group communication applications, and demonstrates how current protocols do not address all the requirements of multicast applications. It then sets out to propose a "high-level design" for a multicast transport protocol. Although it provides more of a requirements specification rather than a design, it does provide a relevant discussion of address management, group membership and service registration issues.

The Multipoint Communications Service Specification [46] documents the origi-

nal BNR-produced version of MCS. MCS is described in section 2.2.

ITU-T recommendations T.122 and T.125 [1, 3] document the evolving Multipoint Communications Service (MCS) standard. It takes the existing MCS implementation described in section 2.2 and attempts to generalize it for use over T.123 [2] compliant point-to-point networks. It makes significant changes to the protocol used for intra-MCS communications and some changes to the service model, but does not go as far as to separate "low-level" connectivity issues from "high-level" multipoint communications abstractions. This effort will render the UNIX MCS implementation performed as part of this research somewhat obsolete from a standards conformance point of view, although the changes necessary for this or any other existing MCS implementation should be contained to two or three "modules" (files of C code). However it should be noted that MCS is one of many components of NSI, and the design of the NSI's architecture should isolate overall impact of MCS standardization to just the MCS component and some of its interfaces. In general, the NSI was designed to use any available multipoint communications service, so that the rest of the system relied on a particular component's implementation (and interface) as little as possible.

Other Research

Leedom [40] provides a relatively complete software distribution containing a pre-packaged, ready to compile C/UNIX implementation of a generic reliable multicast transport service. It currently uses a multiple client/single server architecture similar to that of the Vartalaap system, but plans to allow use of IP Multicast facilities for its future release. The documentation contains a discussion of alternative paradigms for multicast communications, in terms of group-scoped unicast member addresses, ability to send to non-joined groups, address space management, and name registration. This software is used in the NSI as a socket-based communications service for the initial UNIX implementation of MCS.

Arango et al [6] describe the *Touring Machine* communications platform for Bellcore's suite of distributed multimedia applications [25]. This research does not assume a common underlying network structure, and probably has the most similarity to present Telepresence technology. They discuss the Touring Machine system model in fair detail, covering the separation of management policy for devices, conferences, and other system objects from the mechanisms that enforce these policies. The model consists of *connectors*, components of a dedicated session for a particular media type (e.g. audio, video, or data) which have physical network terminals called *endpoints*. These endpoints are logically mapped onto device *ports* at end-stations, representing inputs and outputs of the end-station devices. This model provides for user client registration, session management, network access control, name server query facilities, and an inter-client message passing facility using an object-based software architecture. The architecture consists of station objects, session objects, resource manager objects, resource objects, physical and logical transport objects, and name server objects. A description of Bellcore distributed multimedia applications implemented on top of the Touring Machine platform is provided. Potential evolution paths for the system are explored, including active research on support for hybrid analog/digital networks.

Roseman [58] outlines the GroupKit real-time groupware tool-kit developed in his University of Calgary M.Sc. thesis [57]. This research identifies and develops network requirements and goals for groupware similar to those necessary for telepresence applications. In the end, much of GroupKit's networking responsibilities are handled by the underlying network abstractions of the InterViews tool-kit [41]. GroupKit seems to imply that conference management is a fundamental part of a group communications infrastructure as opposed to an application of it. The exact nature of the communications infrastructure provided by GroupKit is not revealed, although a "conference control" model is implied.

2.2 Multipoint Communications Service (MCS)

MCS encompasses the following aspects: a service definition; an application programming interface (API) providing MCS services; and a communications protocol used by peer MCS service entities to achieve the necessary functionality over a network. An MCS implementation must integrate all of these elements to deliver the proper services to MCS applications.

The original MCS model, prior to ITU-T Recommendations T.122 and T.125, consisted of MCS *users*, MCS *agents*, *links* (local and remote), *attachments*, *sessions*, and software *bridging entities* (figure 2.1). The key coarse-grained communications entity in MCS is a *session*, a collection of multicast, unicast, and token management resources provided over a common channel-ID space. The key architectural component is the MCS *agent*, a software entity actively providing MCS service at each node (workstation or PC). These elements are combined to provide a "mixed" (high and low) level service API in terms of communications abstraction, since an MCS application using these services must handle both low-level point-to-point connection management duties as well as group-level session and channel management functions. The original MCS specification does however explicitly break up the API primitives into three groups: network setup services; multipoint-session services; and miscellaneous routines, as listed in the specification's first appendix. The following is a general description of how an application sets itself up to use MCS for network services.

2.2.1 MCS Setup

First, local links between the application and the MCS *agent* (the locally executing software entity providing MCS services) are allocated and used to connect the application to the agent. Often two or more links are used, one for control information and one or more for data exchange between MCS users. Then the application

Figure 2.1: An MCS Network

allocates one or more remote links, and connects them to remote MCS agents using point-to-point communication services such as NetBIOS, AppleTalk, or TCP/IP. The remote link connection protocol is asymmetric: one end of the link waits, listening for a connection request while the other end calls the listening agent as soon as it is ready. There must be a pre-determined choice of which role is performed by each agent, usually determined by a some sort of a configuration file read in by application code. Local and remote links represent flow-controlled, buffer-oriented FIFO queues implementing "inter-process"¹ communication: local links handle communication between an MCS agent and its client applications on the same node, while remote links handle communication between MCS agents on different nodes. Both local and remote forms of communication share common buffer management facilities. Thus each MCS agent operates more like a bridge or router as opposed to a traditional OSI-style protocol stack, with each agent brokering a unicast and multicast communications service between peer-level applications on the local side and other MCS agents on the remote side (figure 2.2).

After the appropriate local and remote links have been established, they must be associated with a session identified by a unique pre-defined character string. This association is performed via the attach/detach primitives, which also control the construction of the MCS hierarchy: when an MCS agent requests attachment on a remote link, a "direction" or orientation (up or down) for that link is specified. This determines the flow of information through the MCS network, such that information pertaining to centralized services (as implemented by MCS, for example token management) is automatically routed to the main bridge at the top of the hierarchy.

Once established, a session provides the context within which a distributed application functions over the network. Only an "empty leaf" node, whose MCS agent entity has no MCS agents attached to it from below and no channel-IDs allocated

¹The term *process* is applied loosely in this context to mean conceptually independent threads of execution, since not all OTP platforms explicitly support processes in the UNIX sense.

Figure 2.2: Internal Structure of an MCS Agent

itself, may attach "up" to an MCS agent higher in the MCS hierarchy. This prevents channel-ID assignment collisions and other resource allocation conflicts. Unfortunately, this constitutes additional pre-defined configuration knowledge for each individual node that must be read in by applications using MCS services. Although this is the most straight-forward way of bringing up the MCS network, it does restrict the generality and flexibility provided by the MCS protocol.

2.2.2 Using MCS

Once a session has been successfully established, application entities are ready to start communicating with each other using MCS services. This is done by joining the appropriate channels. There are 256 available channels, partitioned into four categories:

1. three pre-defined multicast channels, FULL, EMPTY, and CONTROLLER (0, 1, 2);
2. twenty-nine user-defined multicast channels (3 to 31);
3. 32 "token" unicast channels (32 to 63); and
4. 192 user-id unicast channels (64 to 255).

The list of channels over which application entities communicate, as well as which channels to use for which purposes, is often hard-coded by the application. However, it may be read in from some sort of configuration file, or possibly negotiated through the controller channel or some other pre-designated channel.

The first category defines three special-purpose channels with the following semantics: no-one can join the EMPTY channel; everyone by default is joined to the FULL channel; and, the CONTROLLER channel is used for receiving session-monitoring information such as leave-channel indications. The second category of

channels is used as necessary by the application for multicast communication. Any MCS user attached to a session may send to any channel, however it must be joined to the channel to receive from it. The 32 token channels in the third category are used for token synchronization and serialized message sends utilizing the token-send primitive, while the remaining 192 unicast channels are allocated on request to newly attached users so that existing users may address new users individually.

2.2.3 Bridging Entities

A *bridging entity* is the software responsible for creating the appropriate local and remote MCS links and managing them on the application's behalf. It may also provide higher level communications services based on the MCS model. Some sort of bridging functionality is always necessary at the top node or *main-bridge* of an MCS hierarchy, and at any potential *sub-bridge* nodes (intermediate nodes in the hierarchy). It is always possible for an application to implicitly provide its own bridging function. From this point of view, the NSI may be considered a sophisticated, generalized bridging entity.

2.3 The Ontario Telepresence Project

Telepresence is the application of computing and telecommunications technology to support a sense of social proximity among collaborators despite being separated by space and time. The term was first coined by Xerox Parc [64], and is also demonstrated by Touring Machine [6, 25] research.

Telepresence has emerged as a result of major developments in computer and electronic technologies. In the 1960s and 1970s, the cost of video technology limited the effectiveness of the technology [45]. Recent advances in this technology has made low-cost video conferencing services available, and has expanded it beyond specially equipped conference rooms. The decreasing cost of video technology

makes it affordable to bring cameras, monitors, microphones, and other audio-video equipment to each user's work area. Appropriate connections between collaborators' audio-video equipment, based on a user's community of interest, allow users to create and manage their own *media spaces* [9].

The Ontario Telepresence Project (OTP) developed the concept of telepresence by creating applications that are not only concerned with the technical issues of implementing the required technology, but that also investigate social issues of people using the technology in their work environment. Telepresence demonstrates the social and technical aspects of applying technology to group interactions [54, 44, 14].

The OTP spanned three locations: two in Ottawa, namely the OTP Ottawa Engineering Group (OEG) office and the Department of Systems and Computer Engineering at Carleton University, as well as a single location at the University of Toronto. All three locations were connected by the Internet for control information as well as low-bandwidth digital data exchange. Audio-video links were established between sites using non-dedicated "Switch56" telephone lines. There was also a dedicated Asynchronous Transfer Mode (ATM) [43] connection installed between Carleton University and the University of Toronto, which handled both TCP/IP as well as audio-video traffic simultaneously. Each site was equipped with a number of multimedia devices, in a configuration similar to that of the Touring Machine project.

Video and audio connections made over non-dedicated telephone lines were established using audio-video coder/decoders (*codecs*), which ship audio-video signals over analog telephone lines using standard H.261 or proprietary compression and transmission algorithms. Two standard telephone lines provided a bandwidth of 112 kbp/s, sufficient to ship a signal with approximately a half-second delay. This delay often introduced noticeable confusion with social protocols [42], but such delay effects can be eliminated with the introduction of broadband telecommunications technologies such as ATM. The University of Toronto/Carleton University 45 Mbp/s

ATM connection provided two full-duplex audio-video channels, each of which provide instantaneous, full-motion video with no significant delay.

This infrastructure is used in the design and realization of the *Telepresence Media Space* (TMS) [35]. Each media space within a location was equipped with a number of multimedia devices, including cameras, monitors, speakers, and microphones. These devices were connected directly to an audio-video switch, which provides flexible, PBX-like control over connections between the analog devices. To establish the proper configuration of device-switch connections, devices must be classified according to their ability to *source* (produce) and *sink* (consume) signal information. Cameras and microphones act as sources, while monitors and speakers act as sinks. Some devices may take a dual role: VCRs can be a source and sink of signals. Local media spaces were established and managed by simply connecting the appropriate sources to the appropriate sinks. Multi-location media spaces such as the TMS must also use the inter-site connection facilities described above to support remote users in a natural and seamless fashion. This requires a complex layer of switching and device control logic at all sites, including the management and replication of distributed configuration information.

Many devices are controlled through RS232 connections to computer workstations, with the specific control messages provided by the device manufacturer. Devices may be connected to any suitable workstation located in the laboratory. Computers in the OTP environment were connected to an Local Area Network at each location, and all locations were further interconnected via the Internet.

Multipoint conference support was provided using analog video and audio mixing technology at each site. The video signal of each of the sources of the participants involved with the conference was shipped to a common site, spatially combined into one analog signal using a *picture-in-picture* (PiP) unit, and then shipped back to all sites. In the case of a local baseband connection, the signal was distributed by appropriately configuring the local audio-video switch. In the case of a remote con-

nection using a codec at each end, the individual signals were transmitted over a pair of special "Switch56" telephone lines, while the combined, multi-party audio/video signal was then sent back over the same lines.

2.4 Integrated Interactive Intermedia Facility

The Integrated Interactive Intermedia Facility (IIIF) [24] hides low-level connection management activities for telepresence users by providing appropriate abstractions of a user's media space. A user can request IIIF to connect the local user to a specific person, and IIIF will make all the necessary device connections so that the required media space is created. Abstractions ranging from the *plug* (individual sink or source point of a device) level to the *person* (individual user) level are offered. Different abstraction levels may be used in the same media space if it makes sense, thus it is legal for a *person* to connect to a *device* like a VCR so that a user may watch a pre-recorded meeting, for example. Individual users indicate their accessibility by specifying a *door state* (one of *locked*, *closed*, *ajar*, or *open*), which IIIF uses for determining access permissions to a person's media space.

The services of IIIF are provided to users within a site using client/server technology. Clients use TCP/IP socket connections to submit requests to the local IIIF server. The IIIF server then carries out the request, and replies to the user with the results of the operation.

One of the major contributions of IIIF is the abstraction of device communication paths into connections between media spaces. IIIF also provides some capability for the remote control of devices. These facilities are integrated into the media space abstractions at the *device* level as an integral part of the server, requiring the server to be modified, re-compiled, and restarted to add or remove new devices. The device control abstraction in IIIF is not as powerful as the personal media space abstractions, consisting simply of *grab*, *operate*, and *release* functions. No

asynchronous feedback mechanisms for unsolicited device indications are available.

Although IIIF has been expanded to support the inter-connection of two sites, the mechanisms for achieving this functionality do not support a generalized approach for inter-connecting an arbitrary number of sites together in an efficient or elegant manner. Currently, only devices under the control of the local IIIF server can be accessed remotely, preventing inter-site remote device control. Also, IIIF configuration is statically determined by configuration files read in during system initialization, so that changes in site configuration (e.g. adding a person or removing a device) require a complete system restart.

2.5 Postcards

The current Postcards application is a re-implementation (and re-design) of Xerox Portholes [22]. The Xerox Portholes system supports background distributed group awareness by providing a palette of small "postage stamp" images of persons subscribed to as being part of the user's community of interest. The current Postcards system implemented at Telepresence consists of three major components: *clients*, one per user; *servers*, usually one per site; and *frame grabbers*, allocated to exactly one server in a site (figure 2.3). Servers communicate in a spanning tree in order to forward images to clients that are at different sites, as a user's community of interest may be spread over a number of sites.

Features that are supported by the Telepresence Postcards application include image registration and image access control (image viewing permissions). Each of these is handled by exchanging information between servers arranged into a spanning tree, where each server maintains a database consisting of the following information: images available for subscription; images being requested for subscription; and user permissions for image subscription. Every change to the database information about a user at one site is propagated to all other servers through the spanning tree; in

Figure 2.3: Current Postcards Architecture

effect, it is a distributed database replicated at each site.

Current Postcards frame-grabber processes operate by contacting the appropriate, statically defined server process, and waiting for image subscription messages (or update-image messages) from it. When image subscriptions are received, a frame-grabber process adds the appropriate subscription information into its schedule of frame-grabs and subsequent distribution for subscribed users. An update-image message indicates to the frame-grabber process that an image for the indicated user should be grabbed and distributed as soon as possible.

Chapter 3

Telepresence Network Requirements

Telepresence applications, such as the Integrated Interactive Intermedia Facility (IIIF) system, the Video Mail and Video Attendant, the integration of Voice Mail with Internet electronic mail, and the Postcards system, provide a useful context from which to start analyzing and identifying telepresence network requirements. Investigation of related research showed no existing network interface and/or framework implementation which could be effectively used as-is for developing telepresence applications. Systems such as Rendezvous [38]/Touring Machine [6], MMConf [59], and GroupKit [58] are examples of efforts that assume a larger scope of responsibility and functionality than required by telepresence applications, including, for example, formal and informal conferencing and registration, as well as explicit device registration and management. Other research addresses a portion of telepresence network requirements, but fails to provide the complete level of service needed by telepresence applications, or are too complex and involved for the benefits they would yield (e.g. requiring major porting efforts or extensive operating system modifications). Examples of such research include ISIS [7], the Versatile Message Transaction Protocol (VMTP) [16], the Xpress Transfer Protocol (XTP) [29], and the Multicast Transport Protocol (MTP) [5].

3.1 Scope

Telepresence applications interface with and control a wide variety of multimedia devices, usually connected to a dedicated analog baseband or hybrid analog/digital network [54]. The control portion of these applications conventionally communicate over a separate IP-based network, so that control information is actually transported *out-of-band* relative to application data streams such as audio and video channels (figure 3.1). However there are telepresence applications which transmit both control and application data over the IP network, for example the Postcards system; in such cases the bandwidth demands are, by design, low enough to be handled easily by such networks. Telepresence applications' foremost need is for effective network abstractions for communicating control and/or low-bandwidth application data directly over the IP network. A separation of control channels and application data streams will likely prevail in the short-term, even if digitally-encoded video and audio signals are transported on the same network that is used directly by the controlling applications. In the extreme case, it doesn't matter whether a particular channel carries information used to setup and control application data streams such as audio/video signals, or the actual audio/video data streams themselves, as long as mechanisms are available to allocate and control these physical or logical channels. Thus the NSI concentrates on providing useful service abstractions over an IP-based network, using conventional TCP sockets as well as multicast facilities where available.

Although telepresence applications that use the NSI execute on three computing platforms (UNIX-based workstations, DOS/Windows-based computers, as well as Apple Macintosh products), most applications use a TCP/IP socket or R²C-based client-server architecture. This allows lightweight client user-interfaces to be implemented on all three platforms while the main server functionality need be implemented only once on a single platform of choice (UNIX Workstations run-

Figure 3.1: Telepresence Network Configuration

ning SunOS 4.1, in the case of OTP). Applications written in the C programming language in the UNIX environment have an established, natural programming interface to TCP/IP networking functions via the sockets abstractions (even though more robust and usable socket-based network interfaces are now appearing for PC platforms, based on Microsoft's "WinSock" specification). Similarly, device drivers for controlling the many types of multimedia devices involved in telepresence applications are usually implemented in the C/UNIX environment. Thus the natural implementation environment for the NSI is also the C/UNIX environment, for the main "server" functionality at least.

Choosing the UNIX computing platform facilitates use of high-level TCP/IP data communications abstractions like sockets and LLNL MCAST, however TCP/IP-based networks have limitations of their own that are inherited by the NSI. In general, no real-time response or bandwidth guarantees can be made by low-level IP network protocols (although there is much research effort underway to meet such needs, such as the Real-Time Protocol (RTP) [61] and the Resource Reservation Protocol (RSVP) [66]). In particular, the Internet is currently inadequate for carrying interactive high-quality synchronized audio and video streams, even though the Internet Engineering Task Force coordinates on-going efforts in this area via the Audio Video Transport Working Group. The telepresence philosophy for networked applications is designed around this fact, using separate communications substrates for control/low-bandwidth application data and high-bandwidth real-time audio and video streams. The design of the NSI should allow for application data transmission within its framework, while providing mechanisms to manage distinct yet associated data and control channels.

3.2 Network Goals

Most network interfaces share, to varying extents, at least the following goals:

- **simplicity:** the interface and operational models should allow easy access to communication primitives, without forcing applications to deal with unnecessarily complex interaction mechanisms;
- **usefulness:** service abstractions and communication primitives provided by the network interface should be powerful enough to be immediately useful, without the need for extra programming to achieve the desired functionality;
- **scalability:** growth of the network should not be arbitrarily restricted by internal structure or performance limitations (within reason); and,
- **standards compliancy:** communication protocols should adhere as much as possible to existing and emerging standards, allowing greater inter-operability and flexibility for end-users.

Different network implementations make different trade-offs with respect to these goals, depending on both the nature of the network's design as well as the applications it will need to support. Telepresence applications also share these goals for the design of a network infrastructure, but OTP has refined its own set of network goals and requirements based on the experience of developing existing telepresence applications, as itemized below. These requirements apply to the overall data communications needs of the OTP, in that they may be applied to the effective combination of all data and control network substrates seen as a whole (i.e., from the end-user's viewpoint). Nevertheless they are also key considerations in designing a network infrastructure for telepresence applications.

1. **Bandwidth/Channel Transparency:** Telepresence applications and their users must be able to define and select logical communications channels (both point-to-point *and* multipoint), each supporting an arbitrarily wide range of bandwidth as well as *quality of service* requirements (i.e., according to error rate, end-to-end delay, and other related network parameters). Ideally, these

channels should also be dynamically re-configurable with respect to bandwidth and quality of service, with minimal affect to any services that are currently active on the channel. The network channel interface should hide all "physical" connectivity details from applications, including the mapping of logical channels to actual network communications paths, as appropriate.

2. **Multipoint Collaboration Transparency:** Telepresence applications include support for work group collaborations, both *dynamic* or temporary collections as well as static, long-term groupings. Groups must be able to be defined independent of location and equipment resources, and the ability to communicate at a group level in addition to an individual level must be supported. This includes appropriate abstractions necessary for work group participants to learn membership and status information about each other, exchange data and control information, maintain group awareness, and manage logical sessions. Again, this should be done independent of the details of physical location and underlying network topologies.
3. **Site-end Heterogeneity:** The equipment configuration at the *site-ends* (i.e., terminal ends) of the network will be in some sense completely arbitrary, due to factors such as competitiveness, lack of standardization, varying levels of user sophistication, and other significant reasons that are not likely to be resolved in the near term, if ever. Therefore network support for telepresence applications must handle different equipment configurations and site-end capabilities smoothly. OTP specifies five representative types of site-end configurations:
 - **basic end** - a codec with monitor, camera, speaker, and microphone;
 - **station end** - a workstation (Sun, PC or Mac) with integrated audio/video capabilities (A/V), residing on the Internet, and a codec;

- **local-station end** - a workstation (Sun, PC or Mac) with integrated A/V, residing on the Internet, but with no codec, thus it can connect with other A/V devices by local connections only (baseband A/V);
 - **IIIF end** - a site with an IIIF server and a codec; and,
 - **local-IIIF end** - a site with an IIIF server but no codec, thus it can connect with other A/V devices by local connections only (baseband A/V).
4. **Security:** Apart from regular security issues inherent in traditional computer-based networks, OTP presents unique challenges for security due to the dynamic, collaborative, and distributed nature of telepresence applications. For example, all members of a collaborative work group may need equivalent access to a pre-defined set of resources independent of the physical location of either the resource or the group members, whereas access to a resource by a user physically co-located yet not associated by a work group may be denied. Security and access rights need to exist on many levels, including a user level (independent of where the user is accessing resources from), a group level, and a site level.
5. **Robustness:** Networks and applications based on network capabilities by default strive for robustness as a primary goal, constrained by trade-offs such as the amount of link and node redundancy. Telepresence applications complicate the robustness issue in many ways, for example:
- Multipoint sessions supporting group collaborations require suitable mechanisms to ensure group integrity, with respect to: (1) collecting, maintaining, and distributing group membership information; (2) providing reliable acknowledged and unacknowledged multicast services; (3) ensuring consistency of state information needed by distributed applications;

(4) managing logical and physical session resources, both static and dynamic.

- Site-end heterogeneity requires effective abstractions to manage different hardware (and software) network interfaces and protocols, due to the differing operational philosophies embedded in the different kinds of configurations and equipment present even in identical configurations.

6. Incremental Extensibility: The distributed and dynamic nature of telepresence applications mandates the ability for new sites to be introduced to the network with minimal impact on current network operations. This capability includes the major tasks of: (1) physically connecting the new device to the network; and (2) making both the new site accessible to the rest of the network and conversely making the rest of the network resources available to the new site. The concept of incremental extensibility also applies to multipoint collaboration, where new members must be dynamically admitted to (or removed from) existing operational collaborative groups smoothly, with minimal impact.

These represent the primary goals for realizing a network services infrastructure for telepresence applications. There are, of course, numerous other goals that any network infrastructure or, indeed, any telepresence system or other application implicitly strives for: optimal performance, code readability and re-usability, error-free design and implementation, etc.; these kinds of goals relate more to commercial or production system development than "proof of concept" prototyping activities typically found in a research environment.

No network implementation can easily meet all these goals completely. Certain trade-offs must be made according to the priorities apportioned to each of the goals. This list is meant to emphasize the relative importance of network goals according to the environment and structure that telepresence applications function within, by

explicitly identifying a subset of goals and putting each in the context of the existing telepresence architecture.

3.3 Service Requirements

We have discussed the network goals for an infrastructure supporting networked telepresence applications. While this provides a design philosophy and a set of guidelines for building the NSI, it does not define actual functional requirements of a network services interface and framework. At the highest level, three categories of network services for supporting telepresence applications have been identified: *Group Communications*, *Directory Services*, and *Resource Management*. As with NSI's network goals (section 3.2), these service requirements have evolved out of the experience of OTP engineers building telepresence applications since the beginning of the project. Independent of the inter-relationships between these service definitions (i.e., whether or not one considers a particular service a basic or primitive platform on top of which the other services may be built), the need for these three categories of services has become evident.

3.3.1 Group Communications Services

Although many applications may naturally lend themselves to a client-server architecture, systems are often built using a client-server technology simply to facilitate access to shared data and resources, or to manage inter-client communications. Independent of the structure of cooperating applications (i.e., whether or not they communicate directly with instances of each other or indirectly through a single server or a network of servers), the increasing complexity of communication inherent in telepresence and other application domains brings software developers face-to-face with some fundamental UNIX communications and resource limitations.

A prime example of this is the commonly encountered UNIX open file descriptor

limitation: each distinct reliable communications channel or *socket* [63, chapter 15] opened by a UNIX application process occupies at least one file descriptor, and in many UNIX implementations there is a somewhat low (e.g., fifteen to twenty-five) maximum number of open file descriptors allowed per process [37]. Thus the NSI should not arbitrarily tie socket resources to its own communications abstractions, rather it should allow the application to manage its socket connections independent of the services that the socket end-points represent. In general, any interaction with the NSI should not require the use of large amounts of network resources.

To address some of the higher-level “architectural” issues in group communications, a dual-mode multipoint group communications paradigm is provided by the NSI, supporting both *centralized* master-slave system architectures and more truly *distributed* peer-to-peer architectures, as well as the ability to easily multiplex many distinct communications channels over a single socket connection using the mechanism described above.

Where possible, the multipoint communications service should be able to take advantage of lower-level network multicast facilities, yet should not be totally dependent on such capabilities. If possible, a combination of point-to-point and multipoint transport capabilities should be supported within the same network at the same time (figure 3.2). Lower levels of network management detail, such as keeping track of group membership information, maintaining reliable sequenced delivery of data even in the face of dynamic group membership, and allocating and reclaiming appropriate network resources efficiently on behalf of client applications, should be handled transparently by the networking infrastructure.

3.3.2 Directory Services

Networked, co-operative applications like those in the telepresence application domain need a means by which to discover and contact related instances of each other. Remote/distributed *network resources*, such as centralized servers and shared phys-

Figure 3.2: NSI Communications Substrate

ical devices, or even communication resources themselves, need to be advertised and made available to interested parties. Many systems currently use configuration files and/or environment variables read during initialization, or even hard-coded information, to determine how the application should behave with respect to other network entities.

The NSI should provide simple, reliable, "global" (at least within the context of telepresence) database services, appropriately distributed for easy access by network-capable client applications. Such a directory service should most likely favour efficient look-up at the expense of directory modification efficiency, as more dynamic information distribution is more appropriately handled by dedicated group communications resources. To the extent possible, NSI directory services should directly use, or at least be compatible with, established existing and evolving standards in the area, such as the ITU-T X.500 recommendation [53] and the Internet Domain Name Service [53].

3.3.3 Resource Management Services

Once the appropriate *network resources* have been discovered by telepresence application entities, the NSI should facilitate provision of the appropriate mechanisms by which they may be safely and securely accessed and utilized. Before being made available, resources must be instantiated and declared for network availability (often referred in telecommunications terminology as being put "in-service"), and if necessary they must be allowed to become unavailable to the network (similarly referred to as being taken "out-of-service"). Even before this can all happen, logic must be designed and coded to handle the different types of resources extant in the telepresence domain, each requiring different modes and levels of access, control, and sharing. To this end the NSI should provide a framework for its own extension to handle new and different classes of resources.

Two different categories of network resources have been identified:

- resources with a definite physical realization, such as cameras, VCRs, monitors, microphones, and other computer-interfaced peripherals (referred to as *network devices*); and,
- resources with a no physical realization, but exist purely as virtual/distributed network resources with no explicit ownership, such as a shared whiteboard or worksheet, or an informal teleconference (referred to as *network entities*).

Appropriate mechanisms for the provision and management of both categories of network resources should be provided by the NSI.

3.4 Architectural Requirements

The NSI is designed to meet the functional communications and resource management needs of telepresence applications no matter what internal structure these applications exhibit, however some architectural goals and requirements have been identified for its construction. Some of these relate to the ramifications of interfacing NSI services to existing applications as well as new applications written in the current telepresence environment, while others have more to do with modularity and re-use.

3.4.1 MCS Component

An existing product called Multipoint Communications Service (MCS) [13] (described in section 2.2), produced originally by a development group in Bell-Northern Research and now owned and marketed by Future Labs, provides a service which fits well the group communications requirements for the NSI. If a UNIX MCS implementation were to be used in some form as a component of the NSI, it would be desirable to construct the component in a modular fashion to be functionally usable independent of the NSI system. This would facilitate the UNIX re-implementation

of some established MCS-based products such as Vis-a-Vis, and would also allow the creation of new MCS-based products by experienced MCS developers in a UNIX environment.

3.4.2 NSI Services

Whatever multipoint communications technology is used in the NSI, the implementation of network services should be relatively independent of that technology's particular interface. For example, it should be possible to use a completely different multipoint architecture such as VMTP [16] in place of MCS with minimal impact on the implementation of the rest of the NSI services. There will no doubt always need to be some re-design and re-implementation, however, depending on the scope of differences between the two multipoint communication services.

3.4.3 Multicast Facilities

Much attention has been given recently to network multicast capabilities, especially in the Internet community [21, 11, 5]. Multicast capabilities already exist at the lower levels of many protocol stacks, such as IEEE 802.3 Ethernet protocol multicast facilities. Newer telecommunications technologies such as ATM [43] also promise to provide some form of multicast capabilities. The design of the NSI should not preclude the use of low-level multicast technology in communications, yet neither should it require the presence of multicast facilities for it to function properly. Ultimately a hybrid of point-to-point and multipoint transports could be supported, used concurrently side-by-side even within the same network.

Multicast network capabilities have many advantages besides higher efficiency due to reduced transmission of duplicate information. Higher-level network layers (e.g. the NSI and its clients) do not have to manage multiple connections and related structures such as spanning trees, and routing between intermediate nodes

is handled by the lower layers. An intermediate network substrate such as the NSI can be made inherently more reliable in the face of node failures if each node is singularly attached to (i.e., a leaf-node of) a multicast network instead of potentially being an intermediary node of, for example, a spanning tree. Building a useful group communications abstraction should be easier given some sort of multicast "group" oriented interface to the network.

There are, however, some drawbacks to multicast network technologies. If the given multicast facilities are unreliable and/or unordered (i.e., packets or datagram may be delivered in different sequences to different receivers), building reliable, ordered transport services out of them can be difficult [11]. There are numerous arguments against ubiquitous access to network multicast facilities, involving such issues as possible congestion and broadcast storms, increased router complexity, and increased exposure to application errors capable of disabling not only a single LAN but potentially disrupting whole Internet WANs [52, page 258].

3.4.4 Client Construction

Using the NSI should not imply a great deal of compile/link or run-time complexity on a client application. If NSI group communications and/or directory services involve complex processing and interactions among many components that are for the most part irrelevant to client applications, then a client-server architecture should be used to isolate clients from the complexities of multipoint communication and distributed database operations at a designated server location.

Telepresence applications designed using the NSI should not be restricted to follow a prescribed style or architecture, however the NSI should provide mechanisms for dealing with the complexities of designing network-capable applications. Clients should interact with NSI services in the same way they communicate with other types of external entities and services; in many UNIX environments (such as OTP's) this means using the BSD-style socket I/O facilities. This allows clients flexibility

in how they interact with the NSI, since input and output may be handled the same as any other socket connection. The NSI should also provide a general framework for telepresence applications to deal with multiple input and outputs, including but not limited to standard interactive inputs and outputs (i.e., "stdin" and "stdout"), NSI interface points, plus any other non-NSI network interface points.

One of the issues complicating networked application design is the need to deal with unsolicited, asynchronous network input. Application code may, at pre-defined points, poll for available network input, process it, and go arbitrarily about its usual business (which may involve indeterminate blocking and/or pauses). However such programs run the risk of inadvertently blocking other application instances, losing data due to buffer overflow, or even worse, encountering highly undesirable side-effects such as intermittent, hard-to-reproduce communication failures and deadlock.

Thus telepresence applications, as well as applications in related domains such as telecommunications software, often adopt what is commonly referred to as an *event-driven* or *state-based* design. After the initialization phase of these systems, program operation reduces to waiting for input events, updating internal state as appropriate, generating output if necessary, and then repeating the whole cycle until the system is terminated. Blocking operations are avoided or limited in duration as much as possible during event processing, to prevent the occurrence of the above-mentioned problems. The NSI interface and framework should support such a programming style, alleviating the need for applications to poll the network for input events. A callback mechanism, by which the client application provides its own functions to handle NSI input events, is desirable.

3.4.5 Implementation Considerations

At the raw implementation level, telepresence applications should interface with the NSI in a simple, well-known fashion commonly used by most other service interfaces. For UNIX applications written in C or C++, this means providing a ".h" header

file for interface and callback function prototype definitions, as well as a C/C++ linkable library containing the NSI client interface implementation. A TCL [50] interface to the NSI would also be useful, since many existing OTP applications already use TCL as a embeddable scripting language interface.

Chapter 4

A Network Services Infrastructure

4.1 NSI Organization

Figure 4.1 shows the basic system elements of the NSI from an operational point of view. The individual components implementing the NSI are detailed in section 4.3. At the core of the NSI's operation is a network of "peer-level" servers (from a functional viewpoint) that co-operate to provide communications, directory, and resource management services. Client applications access designated NSI servers indirectly through a NSI client library interface, although the actual client-server mechanism is to a large extent hidden from client applications through the use of an *NSI Service Access Point* abstraction. Clients are not responsible for server initiation or management of any sort, and can assume a fully established active communications substrate is available at all times. Clients are also not responsible for providing any run-time NSI service logic, in particular resource management functions, as this functionality is pre-defined within the server component of the NSI. A network resource to be accessed by a client application must be of a pre-determined type known to the NSI, which must have the appropriate executable service logic built into the servers before they are initiated.

Figure 4.1: Run-Time Organization of the NSI

Server Organization

Before NSI facilities may be invoked by a client application, the appropriate NSI server process must be active. The usual methods for locating servers for new clients in a UNIX environment are employed. Inter-server NSI communications are provided by MCS both for NSI service operation as well as for group communications between client applications. When a particular NSI server initiates, it negotiates with existing NSI servers to become a member of the NSI server MCS session. All NSI servers for a particular network are usually active before clients appear, however there is no reason new servers cannot join an already existing NSI network in full operation, as this does not in any way affect the services already in progress.

Communications Domains

The basic large-grained unit of group communications service in the NSI is called a *domain*, similar to the concept of a session in MCS (section 4.2.1); in fact there is a one-to-one mapping between NSI domains and MCS sessions when MCS is used as the group communications service for the NSI. A background NSI server session is reserved for inter-server communications, while an MCS session is allocated for each NSI domain referenced by client applications. All communications functionality and resources are provided in the context of an explicit NSI domain.

Directory Services

NSI *directory services* are currently implemented by a distributed database replicated at every NSI server node. This provides fast lookup operations on the database at the expense of maintaining consistent replicated data at each server. MCS group communications services facilitate this task by providing fully reliable, sequenced multicast transmission primitives that are used to propagate database updates. NSI directory services are considered a special case of resource called NSI *network entities*

(see below).

Network Resources

Corresponding to the requirements outlined in section 3.3.3, there are two classes of network resources managed by NSI services, specifically *network devices*, explicit “physical”¹ resources made accessible over the network for remote control and operation; and *network entities*, virtual network resources with no explicit physical realization, whose control and operation is distributed across the network. In the NSI architecture, network devices are managed via a combination of *physical device controllers* (PDCs) co-located with an actual computer-based device interface (e.g., UNIX device driver) and *logical device controllers* located at the NSI server nodes which handle clients that have expressed an interest in the network device. Network entities are managed by *distributed resource controllers* located at NSI server nodes handling interested clients. As previously mentioned, NSI *directory services* are actually just a special case of the “NSI network entity” class of resources which exist at all server nodes.

The NSI has no inherent or default knowledge of any type or *class* of network resource; rather, it provides an internal framework for the implementation of service logic within the NSI to manage specific classes of resources. So when an NSI network initializes, there are no resource *instances* in existence, yet all necessary resource *classes* must be defined in the NSI service logic. Resources of a pre-defined class are instantiated via a function call in the NSI client library interface. An identifier is returned by the NSI which may subsequently be used by any client application that needs to obtain access to the resource instance. When a *network device* is instantiated, a physical device controller (PDC) is created at that NSI

¹ *Physical* is used in the sense that a single identifiable controlling location for the resource exists; this covers quasi-logical computer-based resources like centralized databases and other traditional server entities as well as tangible computer-interfaced devices like cameras and VCRs.

server node where the actual device interface is located, and logical device controllers (LDCs) are subsequently created at nodes handling clients requesting access to the device (figure *reffig:rsrcinit*). Similarly, when a *network entity* is instantiated, a distributed resource controller (DRC) is created at that node handling the request, then subsequently at nodes handling clients which initiate access to the entity.

Client Structure

The use of NSI facilities imposes no unnecessary restrictions on the structure of client applications, even though the interface mechanisms and implementation framework provided by the NSI facilitates an “asynchronous”, state-based/event-driven approach to client design (section 4.2.5). An NSI *Service Access Point* (NSI SAP) abstraction is used to access network services. All NSI service functionality is accessed through an explicitly identified NSI SAP, which is the first parameter to all NSI client library functions and callback definitions (except the NSI *register* function which creates an NSI SAP and returns its ID).

NSI Service Access Points are in reality just TCP/IP socket end-points (often referred to as *network file descriptors*), a fact hidden from the client application behind the NSI client interface. It would need to know about the socket-based nature of NSI SAPs only if the application did not utilize the NSI network programming framework (described in section 4.2.5), in which case it may need to explicitly use a UNIX-style “select/poll” system call to determine when input is ready to be processed from an NSI SAP. Thus an NSI client application implicitly incorporates just the standard TCP/IP socket-based client-server mechanism into its process space, with minimal front-end “wrapper” processing and parsing done only for the purposes of communicating over the client-server connection. Any complex, potentially time-consuming NSI service logic processing is currently done at the server node.

1. After NSI server initialization, no resource instances exist even though all resource classes are defined.

2. Client A creates a device of type "VCR" and gets back a resource ID

3. Client A registers "VCR_A" in directory services; client B looks up and opens "VCR_A" in mode "OM"

Figure 4.2: NSI Resource Management for Network Devices

4.2 Functionality

In order to describe the function of the NSI, we look at three different views of the system. First we examine the high-level *service abstraction model* of the NSI, covering the three main categories of services currently offered: *group communications*; *directory services*; and *resource management*. Next, the lower-level NSI interface mechanisms are explored, including details of the NSI Service Access Point abstraction. Relevant NSI *Application Programming Interface* (API) functions and callback definitions are presented, along with a description of their use, in concert with NSI SAPs, to access network services. Finally, the framework provided by the NSI to enable the rapid development of networked applications is illustrated.

The model for group communications in the NSI is influenced by the choice of MCS as the underlying multipoint communications technology. In fact, one of the most compelling reasons for choosing MCS in this role is the relevance of its service model and its “synergy” with respect to the requirements for networking telepresence applications. Unfortunately, client applications directly utilizing the MCS API are presented with an uneven mix of low-level and higher-level networking abstractions, and must also deal with interface mechanisms that are often irrelevant to unsophisticated applications. It is important to note that, in the context of NSI functionality, the term “client application” does not equate to a single UNIX process; rather, it is defined as a single NSI Service Access Point as allocated by client application code. Thus it is possible for a single UNIX process to represent more than one “client application” to the NSI; conversely, through the sharing of file descriptors, it may be possible for more than one UNIX process to represent just one “client application” to the NSI.

The model for directory services is designed to be a much simplified subset of the fundamental functionality provided by the ISO Directory Services International Standard 9594-1 [31] and the related ITU-T Directory Services Recommendation

X.500 [32] (both described in [56]). NSI directory services should ultimately be compatible with these standards and possibly even be based on the ISO Development Environment UNIX implementation [55] in future versions.

The resource management model provided by the NSI is based on the types of resources found in the context of OTP applications. In telepresence, management of network resources fall into two general categories: (1) explicit remote control of computer-interfaced devices such as video cameras, VCRs, audio/video switches, and the like; and (2) distributed access and control of virtual entities such as shared screens or documents, as well as informal audiographic collaborations involving, for example, shared whiteboards and drawing packages.

4.2.1 Group Communications Service Model

The basic coarse-grained unit of NSI group communications facilities is the *domain*². They are identified within a global name space of integer domain IDs shared by all types of domains. Domain IDs are assumed to be well-known predetermined values assigned globally to each type of application utilizing the NSI, so there is no need for dynamic allocation or registration of domain IDs. All communication services within the NSI are based in the context of uniquely distinguished domains; in fact, the domain ID is the second parameter of every NSI communications library function and callback definition.

Domains

A *domain* is a collection of communication resources for a particular application which consists of the following:

²Conceptually, an NSI *domain* is the equivalent of an MCS *session*, however the term *session* has been replaced by the term *domain* in the recent T.120 series of ITU-T recommendations.

1. a set of *multicast channels*, including three which are pre-defined and the rest of which are available to client applications on a static (i.e., predetermined) or dynamically allocated basis;
2. a set of *unicast channels* which may also be used either on a static or dynamically allocated basis;
3. a set of *user IDs*, which are actually unicast channels uniquely assigned to each client application connected to the domain; and,
4. a set of *tokens* used in conjunction with conventional token operations for internal resource arbitration and management purposes.

NSI clients can *attach* to and subsequently *detach* from (i.e., become associated/disassociated with) any number of domains through an NSI SAP (the NSI interfacing mechanisms are described in section 4.2.4). There are actually four domain operations provided, as there are three *modes* or ways in which a client may attach to the two different kinds of domains available; namely, the operations for domains are: *attach-master* and *attach-slave* for attaching to *master-slave* types of domain; and *attach-peer* for attaching to *peer-to-peer* types of domain.

Channels

NSI *channels* are uniquely identifiable, reliable, and distinct communication paths within a domain. Application data is always sent on exactly one channel at a time by a client; similarly, any data received by a client comes from exactly one channel. The operations defined for channels are simply *join* and *leave*, as the *join* operation currently handles both static and dynamically allocated channels. Client applications may join (and subsequently leave) specified NSI channels to receive data sent on that channel, although it is not necessary to join a channel to be able to send data on it. No admission checking is done for channel membership, thus any

client application may join or leave any valid channel at any time. An *empty channel* is defined as a channel to which no client application has joined; a unicast channel can only be joined if it is empty. Dynamic allocation of channels is based on the "emptiness" of channel; i.e., an empty channel is considered unallocated, and once joined it is implicitly allocated to the joining client and does not become unallocated again until it is empty. Data transmission is done on the basis of variable-length datagrams.

Unicast channels may have only one NSI client joined to them at any one time. User IDs are unicast channels dynamically allocated by the system to represent a client attachment to an NSI domain, such that each NSI client has at most one user ID *per domain*. Clients that attach to a domain do not automatically receive a user ID, however such clients cannot allocate any communications resources or make any request which implies ownership of a resource (e.g., a sequenced send request). The system ensures that user ID are reclaimed if a client becomes *detached* from (i.e., terminates its association with) a domain for any reason. In fact, the system ensures the appropriate recovery of all resources allocated to a particular client no matter how or when the client terminates.

Multicast channels may have any number of client applications joined to them, all of which will receive all data sent on that channel while it is joined. There are two modes in which application data is sent on multicast channels: *normal* (unordered); and *sequenced* (totally ordered). All datagrams sent on a particular channel in *sequenced* mode are guaranteed to arrive in the same order for all receivers (figure 4.3); datagrams sent in *normal* mode may arrive in different orders for different receivers (figure 4.4). Thus there are two operations provided for data transfer: *sequenced-send* and *send*.

There are three pre-defined multicast channels in the NSI: the *EMPTY* channel, to which no client can join and thus no data is ever received on; the *FULL* channel, to which all client applications are implicitly joined; and the *MAINTENANCE* channel,

SEQUENCED SEND OPERATION: Clients A, B, and C all send datagrams at about the same time. Using sequenced sends, all clients joined to the channel will receive the datagrams back in the identical order, including any datagrams originally sent by that client.

Figure 4.3: Sending Data in Sequenced Mode

NORMAL SEND OPERATION: Clients A, B, and C all send datagrams at about the same time. Clients D and E may receive the datagrams in a different order from each other. Note also that A, B, and C do not receive their own datagrams back.

Figure 4.4: Sending Data in Normal Mode

which is designated by the system for broadcasting channel membership information. The latter allows interested clients to monitor domain membership by joining the *MAINTENANCE* channel, since user IDs are also unicast channels ³. Note that client applications can also transmit user data on any of these channels if so desired. It is an error to join or leave either the *EMPTY* or *FULL* channel.

Tokens

Tokens are arbitration resources within the context of a domain that are used, for example, in the implementation of network resource management and synchronization mechanisms. Tokens exist in one of two states : *free* (unallocated); and *grabbed* (allocated to a specified user ID). The following operations are provided:

- **token grab** - obtain exclusive allocation to the given *free* token
- **token release** - give up exclusive allocation to the given *grabbed* token
- **token reset** - reset the state of the given token to *free*
- **token query** - obtain the current state of the given token

No queuing of token grab requests is performed by the system, so that an attempt to grab a non-free token will simply return an error (figure 4.5). As with other NSI communications resources, tokens are automatically reclaimed by the system as appropriate whenever an NSI client with one or more tokens allocated disassociates itself with a particular domain.

³NSI *leave-indications* are generated by the system for clients joined to the *MAINTENANCE* channel when a client with a user ID becomes detached from a domain. It is left up to attaching clients to generate the appropriate message on the *MAINTENANCE* channel when a client first obtains a user ID.

1) Client A and Client both request to grab token X at the same time.

2) Assuming client B's request is processed first, client A's request is rejected while client B's request is granted.

Figure 4.5: NSI Token Management

Domain Modes

The NSI provides two mutually exclusive modes of active ⁴ domain operation. These modes are distinguished by how a domain's member clients are organized relative to each other, which is determined by how the clients *attach* to (i.e., become associated with) the domain. *Master-slave* domains have exactly one *master* NSI client, in which case the domain remains inactive until the master client actually attaches and ceases to exist after the master client has detached or terminated. The rest of the clients must attach to the domain as *slaves*, and every slave attach request is verified by the master client. Thus even though slave clients may temporally attach to a domain before the master does, the attach does not get confirmed until the master itself attaches and explicitly verifies the attach request.

Peer-to-peer domains have no such master-slave relationship between NSI clients. Operationally, a peer-to-peer domain becomes active as soon as the first client successfully attaches to it and is deactivated when no more clients are attached to it. Since there is no *master* client, no application-level verification mechanism is present, and the requesting client automatically becomes part of the domain if its request is otherwise valid. Thus care should be taken to ensure proper allocation and management of peer-to-peer domains, so that domain collisions are avoided; i.e., to ensure that no two client applications attempt to use the same domain for conflicting purposes.

4.2.2 Directory Service Model

NSI directory services aim to provide a simple global naming and lookup mechanism in the context of NSI network and client resources. The operations provided are *add*, *remove*, *modify*, and *lookup*. There are no ownership or security facilities associated with directory entries, so that all of these operations are legal for any NSI client

⁴A domain is considered *active* when it has at least one member client associated with it.

Figure 4.6: Master-Slave Domain Operation

For a peer-to-peer domain, all valid attach requests are confirmed immediately - there is no application-level "indication/response" confirmation mechanism.

Figure 4.7: Peer-to-Peer Domain Operation

at any time. In general, directory services are completely independent of other NSI service concepts such as communication domains and user IDs.

The NSI directory is conceptually a text-based dictionary, with each entry consists of two arbitrary strings: a *key* and a *value*. Key and value strings should contain only regular characters, but since any arbitrary string may be used as a key, it is possible for applications to build arbitrarily complex key/value patterns or formats. For example, a hierarchical naming convention similar to that used in X.500, with segments separated by periods or commas, can be used by an application for dictionary keys. However, since keys and values are just arbitrary strings in the context of NSI directory services, no parsing or format checking can be done by the NSI. Checking for implied formatting or structure of keys or values must be done by the client application.

4.2.3 Resource Management Service Model

As described in section 3.3.3, the NSI resource management facilities divide network resources into two categories: *network devices* which provide remote control of centralized resources and computer-interfaced devices (e.g., cameras, VCRs and audio/video switches); and *network entities* which allows distributed control of virtual resources such as shared screens and whiteboards as well as other kinds of informal audiographic conferences. Each resource type made known to the NSI (by programmatical extension) fits into one of these two categories.

The functional interface to both categories of resources is quite similar. Functions provided by the NSI for resource management are:

- *instantiate* - create an instance of the given resource type;
- *open* - request access to the given resource instance;
- *data send* - send a operational request to the given resource instance;
- *close* - terminate access to the given resource instance; a id,
- *remove* - delete the given resource instance.

The NSI also provides the following callback definition for resource management:

- *data indication* - information sent from the resource "asynchronously" back to a client application.

Instantiate is the only operation that differs between the two categories of NSI resources: *network device* instantiation takes both a *device type* parameter as well as a *device instance location* parameter, whereas *network entity* instantiation takes only a *device type* parameter, since there is no actual "physical" location of an network entity. The *instantiate* operation returns a *resource ID* that is used to indicate the appropriate resource instance to be passed as a parameter to the other operations.

send operations specify an arbitrary command string to be submitted to a device or entity, while *indication* operations contain arbitrary feedback strings generated by the indicated resource for the receiving client.

Facilities for extending the internal NSI service logic to handle specific types of network resources are described in section 4.3.4.

4.2.4 Interface Mechanisms

NSI interface mechanisms are designed to help client applications deal with the non-deterministic nature of network input. In general, when data arrives for a client from the network, the application has no *a priori* information about what kind of data is arriving or its source, and thus can make no assumptions about how to deal with it until after it has been received. Rather than forcing applications to poll the network to discover when and what type of information is arriving, the NSI provides a *callback* mechanism for dealing with the problem of arbitrary network input. This callback mechanism, detailed below, may be invoked manually by calling the appropriate NSI client library function when network input has been detected on an NSI SAP, or is invoked automatically by a simple NSI event-handling framework (section 4.2.5) if it is used.

Service Access Points

As mentioned previously, NSI Service Access Points (NSI SAPs) are client connections to the NSI service layer implemented using BSD-style TCP/IP sockets. It is possible to manipulate NSI SAPs in the same way as other kinds of UNIX file descriptors, however they should not be written to or read from directly by client code; the NSI client interface handles raw input and output of NSI SAPs. Each NSI SAP is treated as a distinct NSI client by the system, and thus a single UNIX application process may allocate several SAPs and look like several different clients

to the NSI.

Each SAP provides ubiquitous access to NSI services, so a client may attach to any number of domains and even obtain individual user IDs for all attached domains over a single NSI SAP, if so desired. NSI SAPs are completely independent of service-level constructs such as domains and channels. This allows a client application complete control over how it chooses to multiplex its communications requirements over the available network resources.

API Functions

Output over NSI SAPs is generated by calling the appropriate client library API function. Two types of functions are provided: *requests*, which are usually unsolicited in nature; and *responses*, which are prompted by specific "indication" callback invocations (see below). NSI API functions are simply responsible for submitting client requests or responses to the appropriate NSI server; any feedback generated at the server, either spontaneously or as a result of a request, is returned asynchronously to the client through a callback procedure.

Callback Definitions

Callbacks are procedures that have a pre-determined prototype or signature *defined* by the NSI but that are *implemented* by the client application. The client's callback procedures are invoked automatically by the NSI when appropriate (i.e., when input received on an NSI SAP is ready to be processed by a client). There are two types of callbacks in the NSI: *indications*, which are spontaneous or unsolicited in nature; and *confirms*, which return the results of some earlier request.

Interface Operation

All NSI client library functions and callback definitions represent a particular operational *primitive* described above, such that a client library function is either an operational *request* or an operational *response*, and a callback definition is either an operational *indication* or an operational *confirm*. In general, the causal flow of events for an operation is *request-indication-response-confirm*, although some operations use a *request-confirm* or a *request-indication* event sequence (figure 4.8). All NSI operations and their applicable primitives are tabulated in figure 4.9. This *request-indication-response-confirm* mechanism, as well as the NSI interface operation in general, is modeled after the standard OSI convention for describing layered communications software operation [30].

As an example of an NSI operation that uses all four primitives, we use the case of a client application attaching to a master-slave domain in slave mode. We assume the domain is already active with a master client. The sequence of events is as follows (figure 4.10).

1. The slave client calls the NSI *attach-slave* request function.
2. The master client receives the attach request in the form of an invocation of an NSI *attach-indication* callback.
3. Assuming the master client application decides to accept the attach request, it subsequently responds by calling the NSI *attach-response* function indicating success.
4. The original slave client receives a confirmation of the attach request via the invocation of an NSI *attach-confirm* callback.

An example of a *request-confirm* primitive sequence is found in the NSI *join-channel* operation (figure 4.11).

Event sequences for a) request->indication->response->confirm operations, b) request->indication operations, and c) request->confirm operations.

Figure 4.8: Primitive Operation Sequences

Operation	Primitive			
	<i>request</i>	<i>indication</i>	<i>response</i>	<i>confirm</i>
attach-master	X			X
attach-slave	X	X	X	X
attach-peer	X			X
detach	X	X		
join-channel	X			X
leave-channel	X	X		
send	X	X		
sequenced-send	X	X		
token-grab	X			X
token-release	X			X
token-reset	X			X
token-test	X			X
directory-add	X			X
directory-remove	X			X
directory-modify	X			X
directory-lookup	X			X
resource-instantiate	X			X
resource-open	X			X
resource-data	X	X		
resource-close	X			X
resource-remove	X			X

Figure 4.9: NSI Operation Primitives

An example of a request->indication->response->confirm operation: 1) A client requests to attach in slave mode. 2) When the master client has attached to the domain (if it hasn't already done so), an attach-indication is presented to the master. 3) After processing the attach request, the master sends back an attach-response to the requesting slave. 4) The slave client is presented with the attach-confirm, which indicates success or failure of the request. Note that primitives 1 and 3 are NSI client library function calls, while primitives 2 and 4 are NSI callback invocations.

Figure 4.10: Attach-slave Event Sequence

1. An attached client calls the NSI *join-channel* request function specifying a list of channels to join.
2. The NSI verifies and processes the request, and causes the invocation of an NSI *attach-confirm* callback.

An example of a request->confirm operation: 1) a client requests to join the given channel(s). 2) the appropriate join-confirm callback is generated, indicating the success or failure of the request to join the given channel(s) by the client.

Figure 4.11: Join-channel Event Sequence

Finally, the NSI *send* operation provides an example of a *request-indication* primitive sequence (figure 4.12).

1. An client calls the NSI *send* request function specifying a datagram to be sent on the given channel.
2. The datagram is distributed to all clients joined to the given channel, and causes the invocation of an NSI *send-indication* callback for the appropriate clients.

An example of a request->indication operation: 1) a client sends data on a given channel. 2) send-indication callback is generated for any and all clients joined to the given channel in the domain.

Figure 4.12: Send-data Event Sequence

4.2.5 Network Services Framework

The NSI provides two mechanisms for processing network input, one of which is referred to as the NSI *network services framework* or network programming framework, described herein. It provides automatic, *multiplexed* (simultaneous) monitoring of activity of NSI network SAPs as well as any other file descriptors used by the application. If an application does not use this framework, it instead uses the alternative NSI network input processing mechanism (implemented by a single NSI client library function `NSI_Process_Event`), that simply processes available events on a given NSI SAP when the client application has determined input is ready on the SAP file descriptor. Note that this framework is quite distinct from the framework provided by the NSI for implementing new network resource classes, described in section 4.3.4.

This framework is provided to facilitate the implementation of software constructed using a *state-based/event-driven* approach. This approach to developing software is popular in many real-time application domains such as telecommunications systems as well as, more recently, in the development of sophisticated Graphical User-Interface (GUI) software. The X Window System and its applications, popular on many current UNIX platforms, is designed in this manner.

In general, state-based/event-driven software goes through two stages: (1) an initialization phase performed when the system is invoked; and (2) an event-processing phase which runs indefinitely until the system terminates. The first phase sets up the system's configuration and allocates the necessary resources, while the second phase continually executes a "input-event and process" loop, performing output actions and updating internal *global state* as necessary. Once past the initialization phase, the system usually maintains some form of global state that determines how future input events are to be handled. From an analysis and design perspective, this approach is more formally embodied in a concept commonly referred to as Finite State Machines (FSMs) (figure 4.13). FSMs play an important role in many software

engineering methodologies ,including the real-time extensions Structured Analysis and Design [65], as well as in Computer-Aided Software Engineering tools such as Harel's StateCharts [28] and ObjecTime [62].

Figure 4.13: Example of a Finite State Machine

The NSI network services event-processing framework consists of the following elements: (1) two client library functions for binding user-defined processing functions with a given file descriptor; and (2) the NSI_Process_Events client library function for initiating the event-processing portion of the framework. The first bind-

ing function, `NSI_Bind_Fd_Input_Handler`, associates the given user-implemented handler procedure with a specified file descriptor, such that whenever input is ready on the specified file descriptor the given handler procedure is invoked. Similarly, the second binding function `NSI_Bind_Fd_Output_Handler` accomplishes the same feat for output readiness on a file descriptor, usually only necessary when using non-blocking write-mode I/O. Once the `NSI_Process_Events` procedure is invoked, the NSI event-processing framework continually loops over all active NSI SAPs and file descriptors with handling procedures bound-in. Note that only file descriptors with bound-in handling procedures for the appropriate mode (input or output) are checked for readiness, but that this binding may occur at any time before *or after* the event-processing loop has been invoked.

Although not necessary for most non-complex client applications, the output handler mechanism is useful in implementing "high-performance" non-blocking server applications. For example, if non-blocking write-mode I/O were active for a given file descriptor X, and the application could not write all its output on X due to potential blocking, then the application would bind in the appropriate output handler routine for the file descriptor, save the leftover output data, and simply continue on as usual. Later, when the NSI event-processing framework detected that the file descriptor was ready for output, the user-defined output handler would be invoked to attempt the output of the leftover data. If all leftover data were successfully written, then the output handler would be unbound (effectively removed by binding in a NULL function pointer) so that it wouldn't be invoked again until necessary.

The following sample code fragment shows how the important elements comprising the NSI network services framework may be used (note that this example does not use non-blocking write-mode I/O, as described above):

•
•
•


```

#include <nsi.h>
.
.
.

typedef enum {INITIAL, SOMESTATE, ANOTHERSTATE} global_state;

static global_state my_state;

void handle_keyboard_input(int fd) {
    .
    .
    /* process input received from the keyboard */
    switch (my_state) {
        case INITIAL:
            .
            .
        case SOMESTATE:
            .
            .
        case ANOTHERSTATE:
            .
            .
    }
    .
    .
}

void NSI_Attach_Indication(int sap,
                           int domain,
                           char *reason_string,
                           unsigned char *optdata,
                           int datasize) {
    .
    .
    /* user-provided implementations for */
    /* all NSI callback definitions      */
    switch (my_state) {
        case INITIAL:
            .

```

```

        .
        case SOMESTATE:
        .
        .
        case ANOTHERSTATE:
        .
        .
    }
    .
    .
}

int main(int argc, char *argv[]) {
    .
    .
    /* perform initialization tasks */
    .
    .
    sap = NSI_Register();
    NSI_Bind_Fd_Input_Handler(STDIN_FILENO, handle_keyboard_input);
    .
    .
    /* attach to application domain(s), join channel(s), etc. */
    .
    .
    my_state = INITIAL;
    NSI_Process_Events(); /* enter event processing loop */
}

```

4.3 NSI Implementation

The NSI was constructed using a combination of top-down and bottom-up approaches. Analysis and design of the system proceeded for the most part in a top-down fashion, as a direct result of the rationale described in previous chapters. This allowed a code-level NSI interface to be specified early in the development cycle, permitting the detailed design and implementation of applications such as GCC (section 5.1) in parallel with NSI development. Coding of the NSI components

started at the lowest levels (i.e., LLNL MCAST and MCS) and proceeded up to the highest level of the system (i.e., NSI service logic and the client-server mechanism). As each component was developed, it was integrated with lower-level components and, where possible, tested using its Tools Command Language (TCL) [50] interface. The implementation of significant NSI components is outlined in this chapter.

4.3.1 System Layout

The overall architecture of the current NSI implementation, with respect to its major components, is shown in figure 4.14. MCS is used as the communications cornerstone of the NSI, as all NSI services directly or indirectly make use of its facilities. The high-level NSI service logic is itself an "MCS application", written in TCL except for some additional C-coded TCL primitives added where necessary. The standard socket-based NSI client-server mechanism is coded in C, consisting of: (1) a server *daemon* process, handling multiple client connections and managing the execution of the TCL-based service logic; and (2) a client library interface executing in the client's process space, that is responsible for establishing and maintaining communications with the appropriate NSI server. At the lower levels of the system, MCS is linked directly with the LLNL MCAST library of multicast socket routines; the LLNL MCAST implementation in turn uses the network (point-to-point TCP/IP) socket services provided by the UNIX kernel itself.

4.3.2 LLNL MCAST

The Lawrence-Livermore National Laboratories Multicast Transport System (LLNL MCAST) [40] presents a multipoint BSD-style socket communications abstraction. Its API consists of multicast versions of most standard, point-to-point socket interface routines, distinguished by pre-pending each function name with "mc_" (e.g. `mc_socket`, `mc_bind`, `mc_read`, `mc_write`, `mc_close`, etc.). LLNL MCAST allows

Figure 4.14: Architecture of NSI Components

applications to open *multicast* sockets capable of sending to/receiving from more than one far-end. Appropriate calls to the `ioctl` function (`mc_ioctl`, in this case) are used to control the nature of the multicast behaviour; e.g., the `MCO_LOOPBACK` option selects whether or not data sent from a multicast socket end-point is looped back to subsequently be read again at the sending end-point. The UNIX MCS implementation uses multicast sockets of type `SOCK_STREAM` to achieve reliable sequenced⁵ delivery of data within the multicast socket-based substrate.

LLNL MCAST currently uses a conventional *unicast* socket-based client-server architecture that does not require any true IP multicast facilities to be present. This implies that the efficiencies of using true multicast transmission capabilities, down to the lowest network level, are not fully realized. However it does have the advantage of allowing a fully functional *multicast* socket services implementation, without the network and operating system modifications necessary to implement true IP multicast facilities in the current Telepresence environment. LLNL MCAST is designed to take advantage of lower-layer IP multicast facilities in future versions, when such facilities become more standardized and widely available.

The LLNL MCAST package comes in a freely-available source-code form. LLNL MCAST is written in C, and uses BSD-style UNIX networking primitives, so minimal effort was required to port it to the SunOS platform used in OTP. Subsequent modifications were necessary, however, to properly support non-blocking internal LLNL MCAST client-server interactions and to otherwise ensure correct operation in the OTP environment.

⁵ *Sequenced* in this context implies that all receivers receive datagrams in the same order, without any guarantees on what that order may be; however datagrams sent from the same sender will always arrive at receivers in the order that they were generated.

4.3.3 MCS

The key component in the NSI architecture for providing a reliable multipoint communications service is MCS. It bridges the gap between the implementation of high-level NSI abstractions provided to client telepresence applications and the low-level network communications substrate provided by unicast/multicast socket implementations. MCS is an extant product supporting networked end-user applications such as Vis-a-Vis, but current implementations of MCS exist only for PC DOS/Windows and Mac computing platforms.

UNIX Version

One of the first steps in building the NSI was the creation of a UNIX version of MCS. All existing MCS implementations share the same API, complete with polling routines, packet counters, buffer management functions, low-level connection establishment as well as higher-level session association primitives, all gathered together in the same interface. The DOS version of MCS was chosen as the starting point since it contained the cleanest and simplest implementation of the main MCS functionality. Although the DOS-based NetBIOS network interface had to be completely replaced, it did provide clues as to the function of an MCS *network* interface, including how transactions going out to and coming in from the underlying network substrate were handled.

The MCS API documented in the original BNR specification [46] is preserved in the UNIX version of MCS. This stand-alone UNIX implementation could potentially be used to adapt existing MCS-based products into the UNIX environment with minimal rework. At the core of the MCS system is an efficient transaction processing engine which manages multipoint data-packet distribution among multiple local links (at the *application* interface) and remote links (at the *network* interface) with a minimum amount of copying and duplication. Much of the effort for providing

this core MCS functionality in a UNIX environment went into areas such as memory management and pointer manipulation, storage alignment and related data structure modifications, function prototyping, and argument formatting. Fortunately, the original MCS implementation was written in an ANSI-compliant version of the C programming language, which meant syntactic differences as well as library function prototype mismatches and semantic inconsistencies were kept to a minimum.

The primary modification to the the MCS API was the addition of a non-polling, "multiplex-able" form of network event notification. This comes in the form of an externally visible array of file descriptors which may then be waited on for input and output readiness, usually using the BSD-style `select` system call, for example. A special `mcs_process_event` function may then be invoked until all low-level network input/output has been processed. This mechanism relieves applications from having to continuously poll for network events, while maintaining complete compatibility with the rest of the existing MCS API.

As the C/UNIX MCS API is completely backwards compatible with the PC-based MCS API, it still contains some low-level interface mechanisms that were necessary for the original implementation environment. These mechanisms are for the most part irrelevant for clients using a socket-based network interface. For example, the originally specified interface to MCS included send and receive buffer management mechanisms, which is necessary for effective use of the inherent MCS end-user data flow-control capabilities. However, these issues should be handled implicitly when using a socket-based network interface. A Tools Command Language (TCL) [50] interface was added to the UNIX version of MCS to help isolate applications from these differences in interface mechanisms as well as to facilitate MCS testing and NSI prototyping. TCL is an embeddable scripting language allowing application-specific primitives (MCS API function points, in this case), to be seamlessly integrated with a simple, powerful, script-based interpretive programming language environment. Using TCL has the added benefit of making accessible

many freely available TCL packages and extensions, including a real-time script-level debugger.

It is apparent from MCS documentation (as well as the source code itself) that most of the effort for implementing MCS on a new platform goes into developing the appropriate *network* interface. Implementing a BSD-style socket-based network interface does not pose any fundamental challenges, due to the high level of networking support provided by the C/UNIX programming environment. However, supporting a true multipoint network substrate such as that provided by the LLNL MCAST system poses significant challenges for existing MCS protocols and procedures. Providing a capability for allowing both point-to-point and multipoint transports to be used simultaneously, potentially by a single application, is also a goal of related research on MCS at Technische Universitat Berlin and other European Universities [48].

Multipoint Protocol Adaptation

In the context of the following discussion, *MCS protocol* refers to the inter-node or inter-agent⁶ protocol used to communicate user data and control information over network transport links. The original MCS protocol was designed to work only across point-to-point transport links, meaning that a particular node knew there was exactly one and only one other node at the end of an active network link. Connection establishment followed a simple caller/listener model. Uses true *multipoint* transport links changes this fundamental assumption, so that now there may be any number of nodes at the other end of an active link, even zero. This implies the development of new models for connection establishment and tear-down, that must also handle potentially more complex relationships among active nodes. All of this is a direct consequence of the paradigm shift from a directed point-to-point model to a group-oriented multipoint philosophy at the underlying network

⁶MCS system instances are referred to as MCS *agents*.

level.

There are two choices for integrating multipoint links into the MCS system, in terms of network link management: (1) present the multipoint link to the rest of MCS as a collection of many point-to-point network links; or (2) allow MCS to see each real network link as though it was still point-to-point, hiding the fact that any number of other MCS agents or nodes existed at the other end of the link. Both approaches have their merits and disadvantages, but the fact that the current implementation of the MCS transaction processing engine makes implicit assumptions about the maximum number of available resources (in particular, 32 links available in total, including both local and network links) made the second option seem more appealing. Thus the core MCS engine sees only one actual network link per multipoint link; a hierarchically superior or "master" MCS agent or node thinks there is only one inferior or "slave" node at the other end of a network link, even though there may be any number. There is only ever one master node connected to a multipoint network link.

The connection establishment model for the original MCS protocol is a data communications analog of a simple two-party telephone call. One node in the connection is pre-determined to be the *caller*, and the other is the *listener* responsible for answering or accepting calls. The eventual attachment orientation of the link is immaterial at this stage, although the convention is for hierarchically inferior nodes to always actively call up to listening superior nodes before dealing with their own inferior node connections. This ensures that slave nodes have not already allocated potentially conflicting resources when upwards connections are made.

However, the connection establishment model for multipoint transport links does not have *caller* and *listener* roles, but instead must engage in a low-level negotiation protocol to ensure that for the duration of the connection, only one master attachment is present on a multipoint transport link. This negotiation mechanism also allows slave nodes to attach before a master node is present on a link, and ensures

Figure 4.15: Point-to-Point Connection Establishment

2 of/de 2

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

that the link is activated only when a master node is present. In this case buffering and flow control mechanisms must be managed so that slave nodes do not have to know explicitly when a master node has finally attached. As noted above, multiple attempts to attach as a master must be detected, and the duplicate master must be notified and removed from the connection without disturbing the rest of the multipoint link. Finally, nodes must be allowed to detach without affecting the rest of the nodes attached to the link, unless the node detaching is the master node.

Figure 4.16: Multipoint Connection Establishment

Beyond connection establishment, the general strategy for adapting the MCS protocol to a multipoint transport is to unicast all upward bound information directly to the hierarchically superior or "master" MCS agent, which then remembers which transactions came from which "slave" MCS agent so that any responses or confirms may be unicast back to the appropriate node. The exception is *unsequenced* (normal) send requests from slave nodes, which are multicast to both the master

node and the other slave nodes, however sequenced send requests must be unicast to the master node so that sequencing can be done by the "top" MCS agent. Any requests or indications coming directly from a master node are also multicast to all slave nodes.

As a result of using multipoint network links, a master node will receive possibly many attach indications over a multipoint link, one for each slave node that attempts to attach to the session. This is handled by allowing the first successful attach to activate the network link, while subsequent attach attempts, whether they are accepted or rejected by the master client, do not change the state of the network link. Similarly, detach indications received from slave nodes on a multipoint link do not affect its state; only the master client's detachment from a multipoint link will deactivate it.

Aside from these extensions to the inter-node MCS communication protocol, the main task of the MCS multipoint network interface is to manage the mapping of multipoint network links onto the point-to-point interface to the rest of MCS. This job consists primarily of keeping track of necessary state information for individual slave nodes, and based on this, deciding how to propagate or filter the information that gets passed onto the core MCS function. To illustrate, consider how channel membership is managed. When a slave node joins or leaves a channel on a multipoint link, the network interface needs to propagate the *join-channel* request to the core MCS engine only if no other slave nodes on the multipoint link are joined to the channel. Core MCS functionality is not aware of more than one other MCS agent or node on the other end of a network link, since it considers the members of a multipoint network link as a whole.

Although this strategy shields the core MCS transaction processing function from the responsibilities of handling multipoint network transport, there are still some modifications necessary to the core of MCS to support multipoint links. For example, detach indications from subordinate slave nodes must be propagated to

the master client application through the MCS core, yet the connection must remain completely intact since there may still be other valid slave nodes on the connection.

4.3.4 NSI Service Implementation

As previously noted, the primary NSI service functionality is implemented in TCL such that the NSI is a TCL-based MCS application from the viewpoint of the MCS component. Each NSI server node involves a single instance of a MCS service provider agent. Currently, a single UNIX process encapsulates one NSI server node, managing the execution of a single TCL interpreter which is fed input from multiple NSI clients. Each client is allocated a unique MCS *local* or application link for each attached session, and a mapping is kept to manage information flow back to the appropriate client socket connection.

Role of MCS

MCS, in its original form, deals well with low-level multipoint communications issues such as flow control, point-to-point link and directed hierarchy management, and recovery from low-level communications failures. It provides a robust implementation and service model appropriate for the "raw" and sometimes non-existent networking environment provided by the PC DOS/Windows and Mac platforms. However, it provides applications with an uneven level of multipoint service abstraction, due in part to circumstances mentioned above. The service model of MCS, along with the specified API, forces the application into a certain degree of MCS *agent* management and resources, as well as more general MCS network administration tasks such as arranging and connecting the agent hierarchy.

To deal with this, MCS specifies the use of a software *databridge* entity to isolate the application from network management tasks. Aside from providing the required service abstractions discussed in previous chapters, part of the NSI service

logic's responsibility is handling these databridge duties. But in addition to this, the NSI service logic is responsible for bridging the gap between the manually buffered, polling-style network interface used by PC-based applications and socket-based network interface commonly found in UNIX environments.

The NSI allocates its own "background" MCS session to handle these behind-the-scenes network management issues. This *NSI control* session is also used for implementing all top-level NSI functionality such as directory services and resource management, as well as administrating NSI communication facilities. Although the NSI can directly use many MCS primitives for providing its own communications services, much of the required functionality must be realized by additional NSI service logic. For example, the original MCS implementation obeys the full *request-indication-response-confirm* cycle for only the first application attach request for a given domain at an MCS agent node; subsequent attach requests are immediately confirmed by the local MCS agent without performing the indication/response part of the cycle. NSI must therefore implement its own *out-of-band* attach mechanism to present uniform attach functionality to client applications.

NSI Services Architecture

The TCL-based NSI service logic program is a distributed application, in that all NSI server instances are operationally identical. However MCS requires that one MCS agent node be the "top" MCS agent node, so NSI servers must negotiate a master during initialization⁷. If the NSI server executing as the top MCS agent ever fails, the remaining nodes simply re-negotiate for a new top MCS agent and continue on. NSI server processes may join or leave an active NSI network at any time.

All NSI services are implemented using replicated data structures maintained at all NSI server nodes. Data consistency is maintained by utilizing the reliable, ordered

⁷The NSI currently uses only a single MCS multipoint network link for each session, so in this case the MCS hierarchy degenerates into a single level tree.

data transmission facilities of MCS. Whenever an update to the globally replicated NSI database is required, it is multicast in sequence to all server nodes on the NSI control session and therefore is processed identically at all NSI servers. Any client confirmation or indication feedback is then performed when the request is properly processed, in order, by the server that originated the update (see figure 4.17). The NSI server nodes can thus immediately satisfy database read requests locally without further network processing. NSI server processes which join the NSI network after it has started operation may be sent a copy of the current NSI database via the optional data parameter of the MCS attach-response/confirm mechanism, when the new server attaches to the NSI control session.

To ensure long-term replicated data consistency in mission-critical applications, the "master" NSI server may periodically poll all other active NSI server processes for a checksum of each database. Then if a minor discrepancy is found, and the master's version of the database seems to be valid, the master NSI server transmits its version of the appropriate database to the corrupted server(s). If the master itself is corrupted, then it instead requests a copy of the database be sent from one of the other valid NSI server processes. If no clear decision can be made as to which version of the database is valid, the network may have to be re-initialized.

NSI Domain administration functions, such as dealing with domain mode collisions and multiple master attach attempts are handled in this manner. This distributed "architecture" (from an operational viewpoint) of NSI services also allows pending slave attach requests to be held locally at each server node for each NSI client it receives such a request from, instead of queueing them at a central location.

Directory Services

The NSI directory services database is intended to hold information of a more *static* nature, and so has been designed to deliver efficient lookup performance at the expense of update efficiency. Internally, the database uses the distributed architecture

a) A user-request requiring only a read of an NSI database is confirmed immediately.

b) A user-request requiring an update to an NSI database is submitted to all servers and processed in sequence.

Figure 4.17: Operation of NSI Services

described above, with all entries replicated at all NSI server nodes. Ultimately the NSI directory services should comply with and provide equivalent functionality to recent directory services standards issued by the International Standards Organization and the ITU-T [31, 32, 56] which is based on fundamentally similar operational requirements.

Resource Management

Administration of the NSI resource management framework is also accomplished using the NSI distributed services architecture described above. The NSI's special control session is used for allocating resource management facilities. Twenty-nine multicast channels, tokens, and unicast channels from the NSI control session are reserved to provide facilities for a maximum of twenty-nine active network resources. When created, each instance of a network resource is assigned one dynamically allocated multicast channel and one token. This channel ID of this multicast channel serves as the resource ID of the newly created resource. *Network devices* are additionally assigned a unicast channel for centralized communication purposes. Resources are managed using the distributed NSI services architecture, but *controller entities* (i.e., PDCs, LDCs, and DRCs) are instantiated only where necessary (i.e., where clients have expressed interest in them via resource *create* and resource *open* operations).

This framework provides the communications resources necessary for implementing network resources that are typically found in the OTP environment. Resource implementors are free to allocate more resources as deemed necessary in each case of a particular resource class. The nature of controller entities, including when and how they are created, is also left up to the implementor's discretion. Physical/Logical Device Controllers and Distributed Resource Controllers (PDCs, LDCs, and DRCs) may take the form of separate UNIX processes forked off or connected to as necessary, or may simply be instances of internal data structures along with

the appropriate update and processing logic. Note that the NSI Directory Services is conceptually a distributed resource implemented as a *network entity*.

Chapter 5

Case Studies

The utility of the NSI for developing telepresence applications has been investigated in two case studies: (1) the development of a conference control application called Telepresence Conference Control (TCC) and its use in the construction of a multipoint conference room device manager, and (2) the re-engineering of the OTP Postcards application. Each of these uses the NSI as its basis for network services, yet each represents two quite different forms of application: TCC illustrates conference control and its use to coordinate A/V devices, video codecs, and other similar devices in a multiple conference room environment; and, Postcards illustrates multipoint control, data sharing, and user services with periodic transmission of digital images [36].

5.1 Telepresence Conference Control

5.1.1 Overview

Conferences provide a forum for participants to work together to achieve a common goal. Activities are coordinated in the context of the conference; a conference *chairperson* is often delegated to coordinate the contributions of participants according to an agenda. Participants may also use some form of media to communicate their

ideas to the other members of the conference, such as a slide projector or a white board. Each community has its own policy on how the conference is to be conducted, as, for example, a conference of researchers often differs from the way a conference of business people is conducted. Teleconferences use electronic means to connect participants at different *sites* or locations in a conference. Sites are connected using an electronic transmission medium, over which video and audio signals are exchanged. Services are implemented by entities at one site, as service providers, to all other sites in the conference, as service users. A distributed, shared editor or drawing application may provide the electronic equivalent of a whiteboard. A site within a conference can thus be defined as a set of *service entities* which participate in a teleconference as a whole on behalf of the participants at that location.

The objective of the Telepresence Conference Control system (TCC) is to develop a software architecture that provides a set of logical services to control physical devices and shared software applications, within the context of a video conference between any number of sites connected by a computer network. The audio, video and/or shared application data need not use the same computer network as the controlling TCC software. Usually, each physical device is connected to and controlled by a single workstation, and workstations within a site are connected by a Local Area Network (LAN). Any number of sites, inter-connected by the Internet or some equivalent network technology, should be able to hold any number of such conferences simultaneously. Each site may have any number of audio and video *sources* and *sinks* using various types of communications links for audio, video, and application data, and each site may also be involved in a number of different conferences at the same time. Thus a site is defined as a set of audio and video sources and sinks treated as a group, either formally (in the context of an official organization or corporation) or informally (in the context of a physically co-located group of people somehow affiliated in their task, and in need of some form of collaboration).

The TCC software architecture presents, to the users at each site, a set of ab-

stract services which controls the state of all conferences in which a site is participating. The software architecture is flexible enough to allow resources, sites, and participants to be added and removed from the architecture without disturbing conferences in progress. The architecture also provides the mechanism for conferencing between differently configured sites and also accommodates different conference policies. Sites in a conference are not required to install a different software architecture for each type of conference in which they participate. Inter-operability between different implementations of conferencing software is facilitated by globally-accepted standards, which define a standardized set of services and network protocols. The TCC architecture uses the Generic Conference Control standard, as defined by ITU, to support inter-operability between software components located at different sites.

The TCC system architecture provides a framework for developing and deploying "room-level" teleconferencing services (i.e., coordinating and controlling multiple nodes or sites with potentially sophisticated collections of audio-visual equipment). Mechanisms are supplied for managing all aspects of a conference over its life-span, from its creation, registration, the inclusion of additional nodes, services and participants, the control and arbitration of devices used, to the de-registration and deletion of the conference. It is a framework because, although the appropriate mechanisms for conference control are provided, the particular policies for administering conferencing services such as participant admission and floor control are not pre-defined. Default policies may be provided, but since a primary goal of TCC is complete flexibility with respect to policy implementation, they are not hard-coded. The TCC system was designed from the start as an application of the NSI, since it uses NSI group communications services for all data communications. The design and implementation of TCC was performed by Gerald Boersma at Carleton University as part of the Ontario Telepresence Project, with the author's input and assistance as required.

5.1.2 TCC Operation

Server processes, or *daemons* running on UNIX workstations at each site, provide the set of abstract services for managing conferences at that site. Clients of this server may be any platform which supports TCP/IP, such as Macintosh or DOS-based machines with the appropriate protocol stacks. This client-server architecture provides the flexibility of allowing clients to attach and detach at will without affecting the services implemented in the server, and eases the implementation of TCC services on heterogeneous platforms. Peer-level conference control servers co-operate to establish conferences between sites. The conference control service is thus distributed among the sites participating in the conference.

A *conference control service* is available to all service entities located at a site via a *conference control service provider*. This entity is responsible for the control aspects of a conference, enforced in a way consistent with the policies of the conference. A primary concern for a conference control service is that of service inter-operability, or the capability of service entities to establish a co-operating relationship with peer entities at other sites without prior agreement among the conference participants outside the conference. *Horizontal* inter-operability is the inter-operability of peer service entities located at *different* sites in order to provide a service to a user. *Vertical* inter-operability is the coordination of service entities (potentially providing independent services) within a *single* site to achieve a coherent reaction with all service entities at other sites, as a response to the behaviour of a participant in a conference. A conference control service is concerned with both aspects of inter-operability, within the context of the conference [47].

Inter-operability of service entities at different sites is facilitated by globally-accepted standards. *Generic Conference Control* (GCC) [4] is a standard being developed by ITU that defines a set of *services* that a conference control service should provide to service entities, as well as a *protocol* between instances of the conference control service providers located at each site. The services specified

include conference management, entailing functions such as creating, joining, and terminating conferences, and querying the conferences in progress. Service entities may also query the list of participants in a conference. Conference *conductors* can be assigned or released by a site within the conference. Service entities are also registered and invoked by the conference control service.

5.1.3 GCC Services of TCC

In GCC, each site has a node controller, which is responsible for forwarding requests from a participant at a site to its local conference control service provider. The conference control service provider forwards the request to other sites, where the conference control service provider coordinates its own node controller and applications according to the specific request. A site may have one or more of a number of different roles, such as *conference convener*, *conference conductor*, and *administrator*. A site which creates a conference is the *conference convener*. This site can approve or disallow other sites from joining a conference, thus providing a means to authenticate the membership of a conference. It also terminates the conference, or the conference is automatically terminated when all sites disconnect from the conference. Finally, it specifies the rights other sites in the conference.

As a *conference conductor*, a site indicates to all other site in the conference that the service entities at the conducting site have priority over all service entities at other sites.

One site is also delegated the role of the *administrator*. It approves or denies requests from sites to create new conferences and to query the list of current conferences so that there is control over what conferences are allowed to proceed; the administrator provides a means for resolving conflicts between resources required for two conferences at the same time.

The procedure of joining a conference and a site announcing its presence in the conference is specified as two different services in GCC. Sites can thus join a

conference without actively participating in it; an example would be a conference where a lecturer may broadcast a session where participants can only listen to the contents of the broadcast.

The emphasis of GCC is to provide a general set of services for the mechanism of conference control; the *policy* of the conference is enforced in the operation of the node controller. As a result, the standard is quite general. However, this allows different sites, which can interconnect with GCC compliant conference control service providers, to interpret events in the conference according to their own, site-specific conferencing policy.

The Telepresence Conference Control (TCC) architecture of figure 5.1 implements a subset of the services specified by the GCC standard, in particular, those related to conference establishment and termination, querying the participants and sites in a conference and the current conferences, and conference conductorship. As OTP is a research and development oriented project, group interactions are informal and loosely structured - the node controller for TCC reflects this policy. The site which is responsible for the administration of conferences does not enforce a strict membership policy, although the administrative site is informed when a new site joins a conference. However, because the implementation of conference control policy is limited to the node controller, it is simple for other GCC-compliant sites to teleconference with TCC sites without needing to distribute appropriate applications to the site before the conference is held.

5.1.4 TCC Usage of NSI

GCC, as an ITU standard, assumes an ITU-defined MCS model of multipoint communications, documented in recommendation T.122 [1]. The totally ordered communications capabilities of NSI provide a generalized means to handle state synchronization between the sites in a conference. Other CSCW applications rely on implementing a state synchronization protocol at the application level [23, 60], which may

Figure 5.1: TCC Architecture

increase the complexity of an application. Except in extreme cases where real-time efficiency is of utmost importance in such applications, uniform message ordering for such synchronization purposes may be left to lower layers in the network architecture (i.e. the NSI, in this case).

To implement conference control using the NSI, a conference is mapped directly to a NSI domain ¹. The conference control service provider is responsible for creating and terminating the domain; service entities may only join or leave the domain. Within the domain, one multicast channel, one token, and one unicast channel are reserved for the conference control service provider to implement the protocol between conference control service providers. All other communications resources are available for use by the service entities within the conference. These service entities use the services of NSI directly to communicate data with peer-level service entities at other sites and to access devices and resources available through NSI; only requests related to the control of the conference is directed to the conference control service (see figure 5.2).

All instances of service entities within the same conference share the same channel and token name space. To avoid communications resource utilization conflicts, each service entity must advertise its use of NSI channel resources to all other service entities by means of the NSI directory service. The directory service is organized into a hierarchical tree, which contains information on which service entities are available to participants in a conference as well as resource utilization during the course of the conference. The directory service must therefore support the advertisement of transient information that changes over the course of a conference.

The directory service can also aid in supporting horizontal inter-operability of service entities. Entities announce services in the directory that they provide to service users. Service users at other sites are then aware of the services available.

¹NSI domains provide the same functionality as ITU-T T.122 MCS domains for the purposes of GCC.

Figure 5.2: Role of NSI in TCC Operation

Peer-level service providers at different sites may also negotiate their capabilities so that a common set of services is provided to each site in the conference, regardless of the specific implementation of the service entity at each site.

5.1.5 Results of TCC Case Study

The conference control service in TCC provides a general framework for the development of conference applications by using the facilities of the NSI. An application first announces the services that it is providing to other service entities and negotiates these services with peer-level service entities through the use of NSI directory services. The application then subscribes as a client to both the NSI and the conference control service provider and then waits for events from the service providers and from the user. As events are received by the application from one of these three sources, it acts on the event appropriately. All requests made to service providers are *asynchronous* in that clients are informed of results of requests and given indications and confirmations without having to explicitly poll or otherwise block waiting for them. This facilitates the development of multi-threaded applications which help guarantee timely handling of events, but it does increase the responsibility of the application for managing events. To simplify applications which do not require this sophistication, an interface to the NSI may in the future be provided that allows for synchronous interactions, even though it is theoretically possible for clients to build synchronous interaction mechanisms on top of an asynchronous interface on an per-application basis.

The NSI facilitated the construction of TCC in the following ways:

- It offers integrated, network-capable services for the group communications and directory services needs of TCC in one uniform package. The NSI presents a common, SAP-based interface to the network via high-level service abstractions that encapsulate group administration tasks and isolate applications from

low-level network management details. Without such services, TCC would have had to manage multiple point-to-point connections manually, implementing its own *multipoint* group communications facilities, and providing its own mechanism for partitioning and combining these facilities so as to provide its own services for client applications.

- The asynchronous, callback-based NSI interface mechanisms enables the construction of efficient, multi-threaded server applications such as TCC that reflect real-time, object-oriented development methods. This allowed TCC to be designed using state-of-the-art design methodologies and tools such as Timethreads [12] and Object-Oriented Software Engineering (OOSE) [33]. It also facilitated the use of lightweight threads in developing TCC which, in combination with the NSI's asynchronous callback-based interface mechanisms, enabled TCC's object-oriented design to be implemented in a more direct manner.
- The NSI group communications facilities provide services that are functionally compatible with ITU-T T.122 MCS services, allowing MCS-based applications like GCC (which forms the basis for TCC) to be built using the NSI. This is especially important for TCC, as part of its responsibility is to provide to its clients multipoint communications services based on the ITU-T T.122 MCS specification, which TCC accomplishes by giving clients direct access to NSI group communication facilities. Thus TCC does not have to implement any ITU-T T.122/T.125 functionality of its own.

5.2 Postcards

The Postcards system supports background awareness in geographically distributed groups by providing a palette of small "postage stamp" images of persons subscribed

to as being part of a user's community of interest. Remote *grabber* processes control frame grabbing hardware and appropriately placed cameras via OTP's Integrated Interactive Intermedia Facility (IIIF) system to obtain low-grade video images at participating users' nodes. Users can then invoke a Postcards client to display these images on their local computer screen. Users' subscription information is read when a Postcards client initializes, specifying the names of people which users would like to see in their palette. These images are then continuously updated at regular intervals until either the client terminates or a particular image becomes unavailable (e.g., the corresponding user logs out of the IIIF system, or changes to a *closed* or *locked* door state). Postcards is a re-implementation of the original XEROX Portholes system [22] created by Paul Dourish.

5.2.1 Existing OTP Implementation

The current Postcards system implemented at the Ontario Telepresence Project consists of three major components: the clients (one per user), the servers (usually one per local area of potential Postcards clients, but possibly serving only an intermediary "store and forward" function), and the frame grabbers (allocated to exactly one server) (figure 5.3). As described in section 2.5, clients are invoked at users' nodes to initialize a local display of Postcards images according to users' subscription information.

At the heart of the system is a static network of Postcards *server* processes, responsible mainly for the efficient distribution of images to a number of geographically distributed clients. These server processes are manually arranged into a spanning tree, in an attempt to reduce the amount of data transmission needed to distribute copies of available images as required for client consumption. Routing of images between servers is analogous to Internet Protocol packet routing strategies, in that a particular server only talks to immediately adjacent servers to obtain the required images. Similarly, *client* processes only know about and communicate with their

Figure 5.3: Architecture of Original OTP Postcards System

designated server process, and have no information about how the images get to that server. In other words, server processes know only the adjacent servers and/or frame grabbers images from which an image is obtained, and subsequently propagate images to clients as well as other servers without any indication of the actual image source.

Current Postcards frame grabber processes operate by contacting the designated server process, specified as a command-line parameter when the frame grabber process is invoked, and then waiting for image subscription messages (or update-image messages) from that server. When image subscription messages are received, a frame grabber process simply adds/deletes the appropriate subscription information in its schedule of frame-grabs and subsequent image distributions. An update-image message simply indicates to the frame grabber process that an image for the indicated user should be grabbed and distributed as soon as possible.

5.2.2 NSI-based Postcards

One possibility for an NSI-based Postcards case study would be to completely redesign the Postcards system to use the NSI from scratch. We chose instead to replace the server-network component of the existing Postcards design with an appropriately engineered NSI interface, such that the need for one or more server processes arranged into a spanning tree is avoided; the NSI layer takes care of such details implicitly. Thus the NSI-based Postcards design has just two major components: the *clients* (again one per user), and the *frame grabbers* (one or more per local area of clients) (figure 5.4).

The existing client and frame grabber code is reused as much as possible, with the necessary network modifications made to communicate directly over NSI SAPs instead of standard TCP sockets. Through the development of the IIIF system and other OTP applications, an asynchronous socket-based networking framework has already evolved; the existing OTP Postcards system currently uses this mecha-

Figure 5.4: Architecture of NSI-based Postcards System

nism for communications between components (clients and servers, as well as frame grabbers and servers). NSI interface mechanisms fit into this model of networked application construction quite naturally. This approach demonstrates the ease with which existing OTP applications may be modified to use the NSI without major re-engineering of existing clients and device driver entities.

The design of an NSI-based Postcards system becomes a matter of appropriate allocation and utilization of NSI communications resources and services. A single NSI domain is pre-allocated for use by the Postcards application providing all necessary data and control information exchange (see figure 5.5). Multicast channels are allocated for each available user's image source, and the pre-defined *controller* channel is used to monitor for user registration updates. Image distribution channel identifiers, frame grabber channel identifiers, and optionally user-defined access control lists are registered as necessary with NSI directory services.

User registration and image availability is currently determined via IIIF mechanisms which are independent of the Postcards system. The structure of the existing image availability mechanism is effectively limited to two states: either anyone can access and display a user's image, or nobody can, depending on the user's *door* state.

There are many different ways in which a more flexible *access control* mechanism may be provided to include, for example, an explicit list of users which may be allowed (or denied) access to a registered user's image. Possibilities include: (1) extending the existing method to allow access control list creation and management through IIIF; (2) creation of a new user interface application whose purpose is register and de-register users with the Postcards system and set up the required access control; or, (3) extend the current Postcards client to include the necessary functionality. The latter option was chosen for this case study, with the simplistic operation of implicit registration upon client invocation as well as implicit de-registration upon client termination. During initialization, the client reads in the user's configuration file, optionally containing access control information for that user.

Figure 5.5: NSI-based Postcards Information Flow

Independent of which registration method is chosen, design of NSI-based Postcards registration/de-registration functionality is the same: first allocate a multicast channel for the user, then register the image source (along with the appropriate frame grabber name) with directory services under the user's name, optionally registering with directory services any access control information, and finally sending a registration message over the controller channel. This message allows already active clients to start receiving images for subscribed clients which register after they initialize. De-registration is simply a matter of removing the appropriate directory services entries and sending a de-registration message over the controller channel indicating that the image source is no longer available.

5.2.3 Client Design

The NSI-Based Postcards client process inherits the *static* image subscription behavior (i.e., initially read in from a configuration file and not subsequently modifiable) of the current Postcards system. Dynamic subscription capabilities are now being added to the current Postcards client, which could be merged into the NSI-based Postcards client at a later date if so desired.

Upon invocation, the client performs the user registration process described in the previous section. The NSI-based Postcards client then operates as follows: for each subscribed-to user image, the client accesses the NSI directory services to check for access control information. If present, the list is used to determine if the client has permission to view that user's image. If permission is granted, directory services are then consulted to determine the multicast channel on which the image is being broadcast, as well as the name of the source frame grabber for that image. The appropriate multicast channel for obtaining that user's image is joined, and a message is sent to the image source's frame grabber (whose unicast user-ID channel is contained within its NSI directory services entry) requesting an update of the image. Once all appropriate multicast channels have been joined, the client simply

receives periodic image updates on these channels and displays them (figure 5.6).

The client also joins the predefined *controller* channel to listen for users joining or leaving the system; i.e., to monitor for Postcards user registration/de-registration. When a registration message is received by a client process, it can then check to see if the user has subscribed to the newly registered image. If it has, and access control restrictions are not violated, the indicated image distribution channel is joined. When a de-registration message is received, the client removes the indicated image from the display and is obliged to leave the multicast channel corresponding to the indicated exiting user, which in turn allows the multicast channel resource to be properly reclaimed for re-use by the Postcards system. Note that it should not be difficult to add dynamic subscription facilities to the NSI-based Postcards client, by using the controller channel's user registration/de-registration messages.

The only remaining duty for the NSI-based Postcards client is to leave any joined image distribution channels upon termination, so that frame grabber processes know when to terminate transmission of a user's image (i.e., when no clients are listening for it). However, even this is taken care of implicitly by the NSI, as the MCS system automatically reclaims resources belonging to any users that don't release them explicitly. Thus the appropriate *leave-channel* indications are always generated on the predefined controller channel for the frame grabbers to receive and act upon.

5.2.4 Frame Grabber Design

The frame grabber process for NSI-Based Postcards system is similar to the existing frame grabber design, except that there is no Postcards server for a newly invoked frame grabber process to contact; rather the process simply obtains a unique user-ID channel from the NSI upon invocation, and then registers it under its unique name with NSI directory services. It would be desirable for clients to listen for the pre-mature termination of frame grabber processes and react accordingly, however for the moment this possibility is ignored.

In the NSI-based design, the frame grabber only ever receives one type of message on its own user-ID channel: a "new-client-subscription" message that contains the name of the user whose image is to be accessed. The frame grabber process then obtains the multicast image distribution channel for the given user from directory services. If it is already generating images on that multicast channel, then this is effectively an update-image message as previously defined, and is treated as such. Otherwise, the frame grabber process knows that this is the first subscription request for the user's image, and thus treats it as an image subscription message, scheduling frame-grabs for the image source and distributing them (on the indicated multicast channel, in this case) as appropriate. The frame grabber process must also join the predefined controller channel on which *leave-channel* indications are generated, so that it knows when to terminate transmission of a user's image on the indicated channel (see figure 5.6).

5.2.5 Results of Postcards Case Study

The Postcards system has been retro-fitted to use the NSI at the network interface. This enabled *complete* removal of the Postcards server network, one of the three major components in the existing OTP Postcards architecture (see figures 5.3 and 5.4). Not only was the overall complexity of the Postcards system significantly reduced, but it became possible to reflect a more "problem domain"-oriented model of operation directly in the implementation. Moreover, inherent flexibility in the socket-based NSI interface mechanisms allowed much of the original client and frame grabber process structure to remain intact. Code could be added to handle NSI interactions in a modular fashion with minimal impact on the existing implementation. For example, logic to properly buffer concurrent multiplexed image reception could be included in such a way as to insulate the rest of the client implementation from this responsibility.

At the rawest level, the implementation of the Postcards application was reduced

a) Client A, B, and the Frame Grabber used by client B, all register themselves in the NSI directory.

b) Client A wants B's image, and so looks up B's access control list. If access is allowed, A looks up B's multicast distribution channel and frame grabber ID.

c) Client A joins the specified channel to receive B's image. A also sends a "new client subscription" message to B's frame grabber. When A receives a leave-indication that B has de-registered, it must leave B's multicast distribution channel so that B's frame grabber knows when to stop distributing the B's image (i.e., when it receives the leave-indication for the multicast channel).

Figure 5.6: NSI-based Client/Frame Grabber Operation

from over 21,000 lines of code down to just over 15,000, a reduction of approximately 30 per-cent, due to the elimination of the Postcards server component. Further reduction could also have been achieved by the complete removal of all IIIF system support files related to network interfacing and parsing, as the use of the NSI obviated need of these IIIF services. These services are quite sophisticated in nature, forcing the Postcards application to deal with an unnecessary level of interface complexity.

More importantly, the conceptual complexity of the problem was made an order of magnitude less, since the NSI provided all the necessary network services independent of the geographical topology of clients. Further, the NSI directory services mechanism allows system components to communicate "static" global system information without the need for explicit message passing and parsing. The collection of useful communications resources into a single conceptual NSI *domain* allows trivial set-up and management efforts to produce powerful and flexible communications services: attaching to a single domain and joining/leaving a handful of channels as necessary is the crux of the NSI-based Postcards design.

Finally, new features as well as extensions to existing functionality is facilitated by a set of high-level communications and networking abstractions embodied in the NSI. For example, the dynamic subscription capabilities described above would need no extra messaging or other network functionality to be added to the NSI-based Postcards system, since all necessary information is already provided in the existing network design. The client would simply need to be modified to act on existing messages already being processed on the controller channel, and then provide an appropriate GUI subscription mechanism for manipulation by the user.

Chapter 6

Conclusions

6.1 Research Summary

In the process of developing numerous telepresence applications that must function over local and wide-area network technologies, OTP engineers have discovered the need for a general-purpose network services infrastructure. Common interaction models, communications designs, and network services abstraction are repeatedly encountered in the telepresence application domain. The NSI is designed to accommodate the needs of networked telepresence applications, to ease their creation and to provide a common framework for their development. An appropriate framework, with the right combination of services and interface mechanisms, makes telepresence application development significantly more efficient, and hopefully, due to significant reductions in system complexity, more reliable.

Above and beyond the usual set of generic goals for any network interface, more specific goals and requirements for a networking infrastructure have been identified for the telepresence application domain. The telepresence environment puts special emphasis on some aspects of these goals, adds some goals and requirements of its own, and even implicates multiple interpretations of them in different dimensions. Although much research has gone into the areas of groupware support and CSCW,

no currently available developments uniformly address these goals and requirements while providing the necessary services in an appropriate manner. There are however many efforts that address and at least partially satisfy non-trivial portions of the problem, and appropriate use of these contributions has been made, even to the extent of re-using actual implementations where possible.

Generalized service models for high-level networking abstractions have been synthesized to satisfy telepresence network requirements. Three distinct service abstractions are modeled: (1) multipoint communications; (2) directory services; and (3) resource management. The operation of these services, how they may be used, as well as the mechanisms provided for clients to interface with these services are described in detail.

A Network Services Infrastructure has been developed to meet the requirements and goals identified for telepresence network applications, in at least a prototypical form. The NSI was constructed in such a way as to produce independently reusable components, for example a UNIX version of the MCS system. A socket-based *Service Access Point* interface to the NSI provides asynchronous feedback of network events to applications. The client-server architecture of the NSI application interface permits the construction of NSI client applications on a variety of computing platforms. The NSI application framework offers designers facilities for implementing NSI-based software in C/C++ and TCL languages.

A dual approach for managing network resources in the context of the NSI has been developed. Resources managed by the NSI fall into one of two categories: those that have an identifiable physical realization at a specified location, which we call *network devices*; and those for which no single explicit realization exists, but are managed in more of a distributed or virtual sense, which we call *network entities*. Depending on which category a certain type of resource belongs to, different internal mechanisms exist for extending the NSI to handle new resource types. However, the service abstraction presented at the NSI user interface for handling instances of a

particular resource is common for both categories of resources.

A multipoint communications abstraction based on MCS service specifications [13, 1] is made available in the UNIX environment. The UNIX MCS implementation not only preserves the original polling-style network interface but also provides an exported array of network file descriptors, allowing the use of standard UNIX I/O multiplexing techniques to deliver an asynchronous notification mechanism to networked applications. Underlying network multicast facilities are utilized where feasible, even allowing for inter-working with conventional point-to-point communications technology as needed (see below). A TCL interface to MCS primitives is also provided which further isolates MCS applications from some low-level MCS interface mechanisms.

In creating the UNIX MCS component of NSI, a method of combining reliable multicast facilities with conventional point-to-point transport links was developed. The protocol used between MCS service providers has been extended and adapted to function properly over both multipoint and point-to-point links. Although not tested with true multicast technologies such as IP multicast due to limited deployment and operating system support, this nonetheless demonstrates the ability to utilize both kinds of underlying data communications technology side-by-side in a unified networking service.

The MCS implementation required approximately 11,000 lines of C source code, based on a (heavily modified) base of 6,000 lines of code in the original DOS MCS implementation. The NSI implementation required approximately 5,000 lines of C source code plus 1,700 lines of TCL source code logic.

The experience of constructing the NSI, and the subsequent evaluation of its suitability for both Telepresence Conference Control (TCC) and the Postcards system have demonstrated that the NSI provides essential networking facilities in the telepresence application domain. TCC makes sophisticated, exclusive use of the NSI's multipoint communications facilities, relying on the NSI for all inter-site com-

munications. As TCC had its own specialized mechanisms for global information exchange and resource management (based on ITU-T recommendation T.124 [4]), it did not need to make use of the NSI's directory services or resource management facilities. The Postcards application made use of both the NSI's communications facilities and its directory services in an integrated manner, such that all global application data was stored in the NSI directory, including information about multipoint communications resources provided by the NSI.

NSI-based Postcards development has shown that a significant savings in development time could have been realized if the NSI were used for the original design. The TCC has shown how the NSI model supports the conference control application, a somewhat expected result due to the similarity of NSI services with those suggested by GCC. But beyond GCC, the NSI services allow all telepresence applications to co-exist, not just conference control applications. It is hoped that the NSI will also prove useful for constructing applications outside of the telepresence domain, especially in the closely related and often overlapping areas of groupware development and CSCW research.

6.2 Future Research

Additional experience in developing telepresence applications is needed to fully evaluate the utility of the NSI. Architectures different from those currently used in the Ontario Telepresence Project need to be considered to ultimately test the flexibility of the NSI framework and interface. The relationship of IIIF functionality to NSI services, and the possible migration of IIIF applications or a subset of them to the NSI platform should be investigated. Applicability of the NSI to applications outside the telepresence domain needs to be addressed, as the context of Telepresence technology was only used to inspire and guide its construction, and not in any way to constrain its usefulness.

Stable implementations of reliable multipoint transport services based on true multicast facilities such as IP multicast [21] are just now emerging [48]. When available, the NSI would need to be modified to incorporate these facilities. In the current architecture, this would imply the replacement of the LLNL MCAST system with the appropriate multipoint socket abstraction, or else the modification of MCS to inter-work with the appropriate reliable multicast transport interface.

Further investigation of resource management facilities and mechanisms is indicated. Actual case studies involving both remote device control and distributed entity management are needed to evaluate the usefulness of NSI resource management framework. These would also assist in validating the NSI resource management model, as well as the resulting service abstraction and programming interface.

The implications of providing a more synchronous programming interface to NSI services need to be explored. Although turning an asynchronous interface into a synchronous one may be simple on an application to application basis, providing a general scheme for synchronous interaction is a non-trivial problem. In addition, NSI communication services may need to support quality of service parameters and external channel management capabilities in future versions.

The NSI implementation currently exists in somewhat of a prototypical form, with most of the high-level service logic coded for and executing in an interpreted TCL environment. The MCS component is derived from a specification which is being obsoleted by the T.120 series of ITU-T recommendations for MCS. Different options for re-engineering the NSI need considering, including the possibility of a more integrated, light-weight single-component implementation that may eliminate the need for a client-server architecture. Also, the T.120 series of recommendations themselves may need to be revisited in view of increasingly more popular distributed and less functionally hierarchical network technologies such as the Internet.

Bibliography

- [1] ITU Study Group 8. T.122: Multipoint communications service for audiographics and audiovisual conferencing service definition. ITU Telecommunications Standardization Sector, October 1993.
- [2] ITU Study Group 8. T.123: Protocol stacks for audiographic and audiovisual teleconference applications. ITU Telecommunications Standardization Sector, October 1993.
- [3] ITU Study Group 8. T.125: Multipoint communication service protocol specification. ITU Telecommunications Standardization Sector, October 1993.
- [4] ITU Study Group 8. T.124: Generic conference control for audiovisual and audiographic terminals and multipoint control units. ITU Telecommunications Standardization Sector, 2-5 May 1994.
- [5] S. Armstrong, A. Freier, and K. Marzullo. MTP: An atomic multicast transport protocol. Technical Report X9100359, Xerox Webster Research Center, March 1991.
- [6] M. Arango L. Bahler and P. Bates et. al. Touring machine system. *Communications of the ACM*, 36(1):68-77, January 1993.
- [7] K. Birman, R. Cooper, T. Joseph, K. Marzullo, M. Makpangou, K. Kane, F. Schmuck, and M. Wood. The ISIS system manual, version 2.0. Technical report, Department of Computer Science, Cornell University, Upson Hall, Ithaca, NY 14853, May 1989.
- [8] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(8):272-314, August 1991.

- [9] S. Bly, S. Harrison, and S. Irwin. Media spaces: Bringing people together in a video, audio, and computing environment. *Communications of the ACM*, 36(1):28-47, January 1993.
- [10] G. Boersma. A software architecture for telepresence conference control. Master's thesis, Carleton University, Department of Systems and Computer Engineering, December 1993.
- [11] R. Braudes and S. Zabele. Requirements for multicast protocols. Internet RFC 1458, August 1993.
- [12] R. Buhr. Pictures that play: Design notations for real-time and distributed systems. Technical Report SCE 93-04, Department of Systems and Computer Engineering, Carleton University, February 1993.
- [13] T. Bulkovshteyn. The MCS protocol. In *The ITCA Teleconferencing Yearbook*, 1991.
- [14] W. Buxton. Integrating shared task and person spaces. In *Proceedings of Graphics Interface '92*, Vancouver, British Columbia, May 1992.
- [15] S. Casner, C. Lynn, and P. Park. Experimental internet stream protocol: Version 2 (ST-II). Internet RFC 1190, October 1990.
- [16] D. Cheriton. Vmtp: Versatile message transaction protocol - protocol specification. Internet RFC 1045, February 1988.
- [17] D. Cheriton and W. Zwaenepoel. Distributed process groups in the v kernel. *ACM Transactions on Computer Systems*, 3(2):77-107, May 1985.
- [18] D. Clark, M. Lambert, and L. Zhang. NETBLT: A bulk data transfer protocol. Internet RFC 998, March 1987.
- [19] D. Cohen. A network voice protocol NVP-II. Technical report, USC/Information Sciences Institute, April 1981.
- [20] E. Cole. PVP - a packet video protocol. Technical report, USC/Information Sciences Institute, August 1981.
- [21] S. Deering. Host extensions for IP multicasting. Internet RFC 1112, August 1989.

- [22] P. Dourish and S. Bly. Portholes: Supporting awareness in a distributed group. In *Proceedings of ACM Conference on Human Factors in Computer Systems CHI '92*, Monterey, CA, May 1992.
- [23] C. Elliott. A "sticky" conference control protocol. Technical report, BBN Systems and Technologies, Cambridge, MA, December 1993.
- [24] T. Milligan et al. The telepresence integrated interactive intermedia facility (IIIF). Technical Report OTP-93-04, Ontario Telepresence Project, Toronto, Canada, 1993.
- [25] R. Fish, R. Kraut, R. Root, and R. Rice. Video informal communication. *Communications of the ACM*, 36(1):49-61, January 1993.
- [26] A. Frank, L. Wittie, and A. Bernstein. Multicast communication on network computers. *IEEE Software*, 2(3):49-61, March 1985.
- [27] J. Grudin. Computer-supported cooperative work: History and focus. *IEEE Computer*, 27(5):19-26, May 1994.
- [28] D. Harel. Statecharts: A visual formalization for complex systems. *Science of Computer Programming*, 8, 1987.
- [29] Protocol Engines Inc. XTP protocol definition, revision 3.6. December 1988.
- [30] International Organization for Standardization and International Electrotechnical Committee. *Information Processing Systems - Open Systems Interconnection: Service Conventions*, 1987. Technical Report 8509.
- [31] International Organization for Standardization and International Electrotechnical Committee. *Information Processing Systems - Open Systems Interconnection - The Directory - Overview of Concepts, Models, and Service*, December 1988. International Standard 9594-1.
- [32] ITU Telecommunications Standardization Sector. *The Directory - Overview of Concepts, Models, and Service*, December 1988. Recommendation X.500.
- [33] I. Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Publishing, Reading, MA, 1992.
- [34] V. Jacobson and S. McCanne. vat: X11-based audio teleconferencing tool. UNIX Manual Pages, Lawrence Berkeley Laboratory, Berkeley, CA.

- [35] G. Karam. Telepresence — current and future technologies for collaboration. Technical Report SCE-95-01, Department of Systems and Computer Engineering, Carleton University, January 1995.
- [36] G. Karam, B. McLeod, and G. Boersma. A network software infrastructure for telepresence applications. In *Second International Symposium on Autonomous Decentralized Systems (ISADS)*, Phoenix, AZ, 25-27 April 1995. To appear.
- [37] B. Kernighan and D. Ritchie. *The C Programming Language*. Prentice Hall, 1978.
- [38] Bellcore Information Networking Research Laboratory. The rendezvous system. *Communications of the ACM*, 36(1):62-67, January 1993.
- [39] Y. Lashkari, V. Ramachandran, S. Malpani, and S. Mehndiratta. Vartalaap: a distributed multicast communication system. *Software - Practice and Experience*, 23(7):799-811, July 1993.
- [40] C. Leedom. LLNL multicast distribution version 1.2. Technical report, Lawrence Livermore National Laboratory, January 1993.
- [41] M. Linton, P. Calder, and J. Vlissides. InterViews: A C++ graphical interface toolkit. Technical Report CSL-TR-88-358, Stanford University, Toronto, Canada, 1988.
- [42] M. Mantei, R. Baecker, A. Sellen, W. Buxton, and T. Milligan. Experiences in the use of a media space. In David Marca and Geoffrey Bock, editors, *Groupware: Software for Computer-Supported Cooperative Work*, chapter 6, pages 203-208. IEEE Computer Society Press, Los Alamitos, California, 1992.
- [43] F. McClimans. ATM vs. FDDI: Comparing functionality, features, and benefits. *TELECOMMUNICATIONS*, 28(2):27-33, February 1994.
- [44] G. Moore. Social science models and strategies for telepresence user studies. Technical Report OTP-93-01, Ontario Telepresence Project, Toronto, Canada, 1993.
- [45] G. Moore and K. Schuyler. Videoconferencing 1990's style: Sharing faces, places and spaces. Technical Report OTP-94-03, Ontario Telepresence Project, Toronto, Canada, July 1994.
- [46] Bell-Northern Research Ltd. Networks Technology Division. Multipoint communication service specification. March 1992.

- [47] J. Ott. Service interoperability in teleconferences. Extended abstract. Technische Universitat Berlin, Germany, February 1994.
- [48] J. Ott and C. Bormann. Multicasting the ITU MCS: Integrating point-to-point and multicast transport. In *1994 IEEE Singapore International Conference on Communication Systems (ICCS '94)*, volume 3, pages 1013-1017. IEEE, 1994.
- [49] J. Ott, C. Bormann, H. Gehrcke, T. Kersch, and N. Seifert. MTP-2: Towards achieving the S.E.R.O. properties for multicast transport. In *1994 International Conference on Computer Communications and Networks (ICCCN '94)*, 1994.
- [50] J. Ousterhout. TCL: an embeddable command language. In *Winter 1990 USENIX Conference*, Washington, D.C., January 1990.
- [51] J. Palmer and N. Fields. Computer-supported cooperative work. *IEEE Computer*, 27(5):15-17, May 1994.
- [52] R. Perlman. *Interconnections: Bridges and Routers*. Addison-Wesley Publishing, Reading, MA, 1992.
- [53] J. Postel. Domain name system structure and delegation. Internet RFC 1591, March 1994.
- [54] R. Reisenbach. The ontario telepresence project. In *CHI '94 Conference Companion, ACM Conf. on Human Factors in Computing Systems*, pages 173-174, Boston, MA, April 1994.
- [55] M. Rose. *The ISO Development Environment: User's Manual*. The Wollongong Group, March 1989. Edition 5.0.
- [56] M. Rose. *The Open Book: A Practical Perspective on OSI*. Prentice Hall, 1990.
- [57] M. Roseman. Design of a real-time groupware toolkit. Master's thesis, University of Calgary, Department of Computer Science, February 1993.
- [58] M. Roseman and S. Greenburg. GroupKit: A groupware toolkit for building real-time conferencing applications. In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW)*, 1992.
- [59] E. Schooler. A distributed architecture for multimedia conference control. Technical Report ISI/RR-91-289, USC/Information Sciences Institute, November 1991.

- [60] S. Schooler and S. Casner. An architecture for multimedia connection management. In *IEEE 4th Comsoc International Workshop on Multimedia Communications*, pages 271-274, Monterey, CA, April 1992.
- [61] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. IETF Internet Draft (work in progress), November 1994.
- [62] B. Selic, G. Gullekson, J. McGee, and I. Engelberg. ROOM: An object-oriented methodology for developing real-time systems. In *Proceedings of CASE '92*, pages 230-241, Montreal, Canada, July 1992.
- [63] W. Stevens. *Advanced Programming in the UNIX Environment*. Addison-Wesley Publishing, 1992.
- [64] R. Stults. Media space. Technical report, Xerox Palo Alto Research Centre, 1986.
- [65] P. Ward. The transformation schema: An extension to data flow diagrams to represent control and timing. *IEEE Transactions on Software Engineering*, SE-12(2):198-210, February 1986.
- [66] L. Zhang, S. Deering, and D. Estrin. RSVP: a new resource ReSerVation Protocol. *IEEE Network*, pages 8-19, September 1993.

Appendix A

NSI Client Application Programming Interface

The following describes the Application Programming Interface (API) to the NSI as used by a client application written in the C or C++ language. The NSI API is split up into two main categories: the function prototypes of the NSI interface routines that are called directly by the client code; and, the callback definitions for routines supplied by the client invoked automatically by NSI to handle asynchronous network events.

A.1 Client Library Functions

Initialization and Registration Functions

```
EXTERN int NSI_Register(void);
```

This function registers an NSI client application for unsolicited network notification, returning an NSI sap (actually a file descriptor) which is used to receive network indications/confirmations. Note that multiple NSI saps may be allocated by a single NSI client application by calling this function multiple times, thus allowing for multiple concurrent and independent NSI connections to a single application pro-

cess (useful for ensuring delay-free flow of control and/or high-priority information independent of user-level data and/or lower priority information).

```
EXTERN int NSI_Deregister(int sap);
```

The inverse of the NSI registration function, this function deallocates the given NSI sap and frees its associated resources (i.e., closing the file descriptor and deleting the appropriate NSI client/server connection).

Domain Administration Functions: Attaching and Detaching

```
EXTERN int NSI_Attach_Master(int sap,  
                             int domain,  
                             unsigned char *optdata,  
                             int datasize);
```

This function attaches the given sap to the specified domain as the master, guaranteeing that the invoking client application will be the one and only master of the domain, so that it may be used or designated to perform centralized functions such as authorization and verification (since all domain attach indications will be received by the master) or central data storage. An optional data area is passed starting at optdata, if it is not a NULL pointer, for datasize bytes, if datasize is greater than zero (i.e. optdata is NOT a conventional null-terminated C string), which is simply returned to this same sap in the attach confirm callback parameter of the same name. Note that it is an error to attempt to attach more than one master to a master-slave domain, or to use this function to attach to a peer-to-peer domain; note also that a master-slave domain will not start normal operations until a master has attached to the domain.

```
EXTERN int NSI_Attach_Slave(int sap,  
                             int domain,  
                             unsigned char *optdata,  
                             int datasize);
```

This function attaches the given sap to the specified domain as a slave. All client applications except for one (which calls the above attach master routine) call this routine to attach to a master-slave domain. An optional data area is passed starting at optdata, if it is not a NULL pointer, for datasize bytes, if datasize is greater than zero (i.e. optdata is NOT a conventional null-terminated C string), which appears to a master in the attach indication callback parameter of the same name. Note that it is an error to attempt to use this function to attach to a peer-to-peer domain; note also that a master-slave domain will not start normal operations until a master has attached to the domain.

```
EXTERN int NSI_Attach_Peer(int sap,
                           int domain,
                           unsigned char *optdata,
                           int datasize);
```

This function attaches the given sap to the specified domain as a peer instead of using a master-slave arrangement. An optional data area is passed starting at optdata, if it is not a NULL pointer, for datasize bytes, if datasize is greater than zero (i.e. optdata is NOT a conventional null-terminated C string), which is simply returned to this same NSI sap in the attach confirm callback. Note that it is an error to attempt to use this function to attach to a master-slave domain.

```
EXTERN int NSI_Attach_Respond(int sap,
                              int domain,
                              NSI_Result result,
                              unsigned char *optdata,
                              int datasize);
```

This function responds to the oldest outstanding "unanswered" NSI attach indication. The requested attach succeeds if result is SUCCESS, otherwise it fails (result equals FAILURE). An optional data area (independent of the data area received in an attach indication) is passed back to the requesting application starting at optdata, if it is not a NULL pointer, for datasize bytes, if datasize is greater than zero (i.e. optdata is NOT a conventional null-terminated C string).

```
EXTERN int NSI_Detach(int sap,
                     int domain);
```

This is the inverse of the NSI attach functions, detaching the specified domain from the given sap.

Domain Administration Functions: Channel Management

```
EXTERN int NSI_Join_Channels(int sap,
                             int domain,
                             int id_requested,
                             unsigned char *channel_list,
                             int channel_count);
```

For the specified domain accessed through the given sap, this function joins the channel_count channels in channel_list. If id_requested is non-zero, a unique "private" user-id channel is allocated which no one else can join.

```
EXTERN int NSI_Leave_Channels(int sap,
                             int domain,
                             unsigned char *channel_list,
                             int channel_count);
```

This is the inverse of the NSI_Join_Channels function.

Domain Functions: Data Transmission

```
EXTERN int NSI_Send(int sap,
                    int domain,
                    int channel,
                    int confirm_expected,
                    unsigned char *data,
                    int size);
```

For the specified domain accessed through the given sap, this function sends information on the indicated channel starting at data, if it is not a NULL pointer,

for `size` bytes, if `size` is greater than zero (i.e. data is NOT a conventional null-terminated C string). If `confirm_expected` is non-zero, a response will be solicited from all recipients of the send request specifying a positive or negative acknowledgment (counts of both are indicated back to the sender). Data sent on a multicast channel to which the client application is joined to is not delivered back to the same sap from which it was sent.

```
EXTERN int NSI_Sequenced_Send(int sap,  
                               int domain,  
                               int channel,  
                               int confirm_expected,  
                               unsigned char *data,  
                               int size);
```

This function is similar to the normal NSI send function, except that all receivers will receive data sent on a multicast channel in the identical order (as long as they use this function instead of the normal NSI send function). Also, data sent on a multicast channel using this function is delivered back to the same sap from which it was sent if it is also joined to the channel.

```
EXTERN int NSI_Send_Respond(int sap,  
                             int domain,  
                             NSI_Result result);
```

If a receive indication resulting from a normal NSI send request specifies that a response is expected, this function is used to issue a positive (if `result` is SUCCESS) or negative (`result` equals FAILURE) response back to the sender.

```
EXTERN int NSI_Sequenced_Send_Respond(int sap,  
                                       int domain,  
                                       NSI_Result result);
```

This is the "sequenced" version of the `NSI_Send_Respond` function, used when responding to a sequenced send request indicates that a confirm is expected.

Domain Functions: Token Management

```
EXTERN int NSI-Token_Grab(int sap,  
                           int domain,  
                           int token);
```

For the specified domain accessed through the given **sap**, this function grabs the indicated token on behalf of the invoking client application.

```
EXTERN int NSI-Token_Release(int sap,  
                              int domain,  
                              int token);
```

This is the inverse of the **NSI-Token_Grab** function, releasing the indicated token.

```
EXTERN int NSI-Token_Reset(int sap,  
                            int domain,  
                            int token);
```

For the specified domain accessed through the given **sap**, this function resets the status of the indicated token to "not grabbed" (i.e. free).

```
EXTERN int NSI-Token_Test(int sap,  
                           int domain,  
                           int token);
```

For the specified domain accessed through the given **sap**, this function does not actually modify the indicated token but rather requests an indication of its current status.

Directory Services Functions

```
EXTERN int NSI_Directory_Add(int sap, char *key, char *value);
```

This routine requests to add the arbitrary string in **value** to the NSI directory at the position specified by **key**. Both keys and values in the NSI directory should not contain null or newline characters.

```
EXTERN int NSI_Directory_Remove(int sap, char *key, char *value);
```

This is the inverse of the NSI_Directory_Add routine, requesting the removal of the directory entry specified by **key** with the given **value**.

```
EXTERN int NSI_Directory_Modify(int sap,  
                                char *key,  
                                char *old_value,  
                                char *new_value);
```

This routine requests modification of the NSI directory entry at **key** with the given **old_value** to subsequently contain **old_value** instead.

```
EXTERN int NSI_Directory_Lookup(int sap, char *key);
```

This routine requests a lookup in the NSI directory the value associated with the given **key**.

Resource Management Functions

```
EXTERN int NSI_Resource_Create_Device(int sap,  
                                       NSI_Resource_Type type,  
                                       FILE *resource_loc);
```

This routine requests the instantiation of a network device of the given **type** whose device file is specified by **resource_loc**.

```
EXTERN int NSI_Resource_Create_Entity(int sap, NSI_Resource_Type type);
```

This routine requests the instantiation of a network entity of the given **type**.

```
EXTERN int NSI_Resource_Open(int sap, int resource, char *mode);
```

This routine requests the opening of the given **resource** in the (application-defined) **mode**.

```
EXTERN int NSI_Resource_Send(int sap, int resource, char *data);
```

This routine sends of the character string in `data` to `resource`, which must already be open on the given `sap`.

```
EXTERN int NSI_Resource_Close(int sap, int resource);
```

This is the inverse of the `NSI_Resource_Open` routine.

```
EXTERN int NSI_Resource_Remove(int sap, int resource);
```

This is the inverse of the NSI resource creation routines described above.

General Framework and Event Management Functions

```
EXTERN int NSI_Bind_Fd_Input_Handler(int fd,  
                                     NSI_Fd_Handler_Proc handler);
```

This function binds the given user-defined input handler routine with the specified file descriptor `fd`. The NSI network programming framework (initiated by calling `NSI_Process_Events`) will call this handler routine whenever input is ready on the specified file descriptor.

```
EXTERN int NSI_Bind_Fd_Output_Handler(int fd,  
                                     NSI_Fd_Handler_Proc handler);
```

This function binds the given user-defined output handler routine with the specified file descriptor `fd`. The NSI network programming framework (initiated by calling `NSI_Process_Events`) will call this handler routine whenever output “capability” is ready (i.e., space is available on the file descriptor for output) on the specified file descriptor.

```
EXTERN int NSI_Process_Events(void);
```

This function initiates the NSI network processing framework, continuously reading and processing input “asynchronously” from the network and all other sources as it arrives. Non-NSI input (and possibly output) file descriptors used by the program must be registered by one the the two NSI binding functions to a callback routine that will be invoked when input or output is ready on that file descriptor.

```
EXTERN int NSI_Process_Event(int sap);
```

This function reads and processes information returned asynchronously from the network that is directed to the given NSI *sap* (actually a file descriptor). The client application's appropriate callback routine (as defined below) is then invoked with the appropriate parameters. Note that this routine is actually provided by the NSI API, such that the client application simply invokes it when necessary (i.e., after a select, or a non-blocking/polling read, indicating that data is ready on the specified NSI *sap* file descriptor).

A.2 Client Callback Definitions

The following function definitions are *callback* prototypes for routines that a client must provide implementations for. They are routines which are invoked via the asynchronous NSI network programming interface and framework; specifically, they may be invoked after a client application has called either of the *NSI_Process_Event* or the *NSI_Process_Events* library functions described above. A client may provide a simple *null* implementation for callbacks which are of no interest to the application.

Domain Administration Callbacks: Attaching and Detaching

```
EXTERN void NSI_Attach_Indication(int sap,  
                                int domain,  
                                char *reason_string,  
                                unsigned char *optdata,  
                                int datasize);
```

This callback indicates that an attach indication has been received on the indicated *sap* for the given domain, with extra information in *reason_string*. Optional data passed by a client application to an NSI attach request starts at *optdata*, if

it is not a NULL pointer, for `datasize` bytes, if `datasize` is greater than zero (i.e. `optdata` is NOT a conventional null-terminated C string). Note that the implementation of this function, and all other callback functions described in this section, MUST be provided by the client application, otherwise it will not build (i.e., link) successfully.

```
EXTERN void NSI_Attach_Confirm(int sap,
                               int domain,
                               NSI_Result result,
                               char *reason_string,
                               unsigned char *optdata,
                               int datasize);
```

This callback confirms (positively or negatively) a previous attach request made by the client application over the indicated `sap` for the given `domain`. `result` indicates SUCCESS or FAILURE (i.e., the attach request has succeeded or failed), with extra information in `reason_string` in both cases. Optional data passed by a client application to an NSI attach response starts at `optdata`, if it is not a NULL pointer, for `datasize` bytes, if `datasize` is greater than zero (i.e. `optdata` is NOT a conventional null-terminated C string). Note that this is new information passed back to the requesting client application directly from the granting/denying application, which is independent from the optional data originally sent by the requesting application in an attach request.

```
EXTERN void NSI_Detach_Indication(int sap,
                                  int domain,
                                  char *reason_string);
```

This callback indicates that a remote client application has detached from `domain` for the reason described in the given `reason_string`.

Domain Administration Callbacks: Channel Management

```
EXTERN void NSI_Join_Confirm(int sap,
```

```

int domain,
NSI_Result result,
char *reason_string,
int user_id,
unsigned char *channel_list,
int channel_count);

```

This callback confirms (positively or negatively) a previous join request made by the client application over the given sap for the `channel_count` channels in `channel_list`. `result` indicates SUCCESS or FAILURE (i.e., whether the join request has succeeded or failed), with extra information provided in `reason_string` in both cases. A unique non-zero user-id channel is indicated in `user_id` if one was successfully requested.

```

EXTERN void NSI_Leave_Indication(int sap,
                                int domain,
                                char *reason_string,
                                unsigned char *channel_list,
                                int channel_count);

```

This callback indicates that the `channel_count` channels in `channel_list` in `domain` have become empty. `reason_string` documents the cause of the leave indication, as such indications are generated not only due to explicit leave requests but also when client applications detach a sap from a domain on which it still has membership in one or more channels, for example, or if an NSI network suffers from some internal disconnection or partitioning, or some other unrecoverable network failure. Note that leave indications are provided to all saps joined to the controller channel for such failure cases.

Domain Functions: Data Transmission

```

EXTERN void NSI_Receive_Indication(int sap,
                                    int domain,
                                    int response_expected,

```

```

int source_id,
int dest_channel,
unsigned char *data,
int size);

```

This callback indicates that `size` data bytes starting at `data` have been received, via the given `sap`, on `dest_channel` originating from `source_id`. If `response_expected` is non-zero then the receiving client application is expected to generate an appropriate NSI send response.

```

EXTERN void NSI_Sequenced_Receive_Indication(int sap,
                                             int domain,
                                             int response_expected,
                                             int source_id,
                                             int dest_channel,
                                             unsigned char *data,
                                             int size);

```

This is the sequenced version of the `NSI_Receive_Indication` function.

```

EXTERN void NSI_Send_Confirm(int sap,
                             int domain,
                             NSI_Result result,
                             char *reason_string,
                             int channel,
                             int pos_count,
                             int neg_count);

```

This callback confirms a previous NSI send request over the indicated `channel` made by the client application on the given `sap`. `result` indicates SUCCESS or FAILURE (i.e. the send request has succeeded or failed), with extra information in `reason_string` in both cases. `pos_count` is a tally of positive responses (i.e., clients responding with positive result parameters to the NSI send response function), while `neg_count` is a tally of negative responses. Note that the positive/negative responses have no bearing on the `result` indicated as a parameter to this callback.

Domain Callbacks: Token Management

```
EXTERN void NSI-Token_Op_Confirm(int sap,  
                                int domain,  
                                int token,  
                                NSI-Token_Result result,  
                                char *reason_string);
```

This callback confirms the result of a previously issued NSI token command (grab, release, reset, etc.) issued for the specified token on the given sap. `result` indicates the result of the attempted token operation, while `reason_string` describes some any error condition encountered.

```
EXTERN void NSI-Token_Test_Confirm(int sap,  
                                   int domain,  
                                   int token,  
                                   NSI-Token_State state,  
                                   char *reason_string);
```

This callback confirms the result of a previously issued NSI token test issued for the indicated token on the given sap. `state` indicates the current state of the indicated token.

Directory Services Callbacks

```
EXTERN int NSI_Directory_Add_Confirm(int sap,  
                                     char *key,  
                                     char *value,  
                                     NSI_Result result,  
                                     char *reason_string);
```

This callback confirms an `NSI_Directory_Add` request positively (if `result` is `SUCCESS`) or negatively (if `result` is `FAILURE`). `reason_string` will contain a descriptive error message upon failure of the request.

```
EXTERN int NSI_Directory_Remove_Confirm(int sap,  
                                         char *key,
```

```

char *value,
NSI_Result result,
char *reason_string);

```

This callback confirms an `NSI_Directory_Remove` request positively (if `result` is `SUCCESS`) or negatively (if `result` is `FAILURE`). `reason_string` will contain a descriptive error message upon failure of the request.

```

EXTERN int NSI_Directory_Modify_Confirm(int sap,
char *key,
char *old_value,
char *new_value,
NSI_Result result,
char *reason_string);

```

This callback confirms an `NSI_Directory_Modify` request positively (if `result` is `SUCCESS`) or negatively (if `result` is `FAILURE`). `reason_string` will contain a descriptive error message upon failure of the request.

```

EXTERN int NSI_Directory_Lookup_Confirm(int sap,
char *key,
char *value,
NSI_Result result,
char *reason_string);

```

This callback confirms an `NSI_Directory_Lookup` request positively (if `result` is `SUCCESS`) or negatively (if `result` is `FAILURE`). If successful, the corresponding entry for the given `key` is contained in `value`. `reason_string` will contain a descriptive error message upon failure of the request.

Resource Management Callbacks

```

EXTERN int NSI_Resource_Create_Device_Confirm(int sap,
NSI_Resource_Type type,
FILE *resource_loc
int resource,
NSI_Result result,
char *reason_string);

```

This callback confirms an `NSI_Directory_Create_Device` request positively (if `result` is `SUCCESS`) or negatively (if `result` is `FAILURE`). If successful, the identifier of the new device is contained in `resource`. `reason_string` will contain a descriptive error message upon failure of the request.

```
EXTERN int NSI_Resource_Create_Entity_Confirm(int sap,
                                              NSI_Resource_Type type,
                                              int resource,
                                              NSI_Result result,
                                              char *reason_string);
```

This callback confirms an `NSI_Directory_Create_Entity` request positively (if `result` is `SUCCESS`) or negatively (if `result` is `FAILURE`). If successful, the identifier of the new device is contained in `resource`. `reason_string` will contain a descriptive error message upon failure of the request.

```
EXTERN int NSI_Resource_Open_Confirm(int sap,
                                     int resource,
                                     char *mode,
                                     NSI_Result result,
                                     char *reason_string);
```

This callback confirms an `NSI_Resource_Open` request positively (if `result` is `SUCCESS`) or negatively (if `result` is `FAILURE`). `reason_string` will contain a descriptive error message upon failure of the request.

```
EXTERN int NSI_Resource_Indication(int sap, int resource, char *data);
```

This callback contains asynchronous feedback information in data from the previously opened resource.

```
EXTERN int NSI_Resource_Close_Confirm(int sap,
                                       int resource,
                                       NSI_Result result,
                                       char *reason_string);
```

This callback confirms an `NSI_Resource_Close` request positively (if `result` is `SUCCESS`) or negatively (if `result` is `FAILURE`). `reason_string` will contain a descriptive error message upon failure of the request.

```
EXTERN int NSI_Resource_Remove_Confirm(int sap,  
                                       int resource,  
                                       NSI_Result result,  
                                       char *reason_string);
```

This callback confirms an `NSI_Resource_Remove` request positively (if `result` is `SUCCESS`) or negatively (if `result` is `FAILURE`). `reason_string` will contain a descriptive error message upon failure of the request.

END

1 7 + 1 | 1 - 9 | 5

FIN